2.4: The 6-step framework

This is it. This is where we start building our own Python framework to rapidly construct object detectors using **Histogram of Oriented Gradients (https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/)** and **Linear Support Vector Machines (https://gurus.pyimagesearch.com/topic/support-vector-machines/)**.

But why bother with HOG + Linear SVM? Doesn't OpenCV already come with methods to train your own custom **Haar classifiers (https://gurus.pyimagesearch.com/lessons/face-detection-in-images/)**?

Yes, that's true — but the Haar classifiers shipped with OpenCV (and even the ones you train yourself) leave much to be desired. The problems include, but are not limited to:

- **Problem #1:** Haar cascades are *extremely slow* to train, taking days to work on even small datasets.
- **Problem #2:** Haar cascades tend to have an *alarmingly high* false-positive rate (i.e. an object, such as a face, is detected in a location where the object does not exist).
- **Problem #3:** What's worse than falsely detecting an object in an image? Not detecting an object that actually *does exist* due to sub-optimal parameter choices.
- **Problem #4**: Speaking of parameters: it can be *especially challenging* to tune, tweak, and dial in the optimal detection parameters; furthermore, the *optimal parameters can vary on an image-to-image basis!*

Because of these reasons, the Haar cascade model for object detection has fallen out of favor to the more robust HOG + Linear SVM approach, which has become the widely accepted method to training your own custom object detectors.

In the remainder of this lesson, we'll review the 6-steps required to build your own custom object detection framework. Then, in the rest of this module, we'll implement each of these steps one-by-one. By the end of this module, you'll have a ***fully functioning*** object detection framework that you can use to train your own custom object detectors.

Objectives:

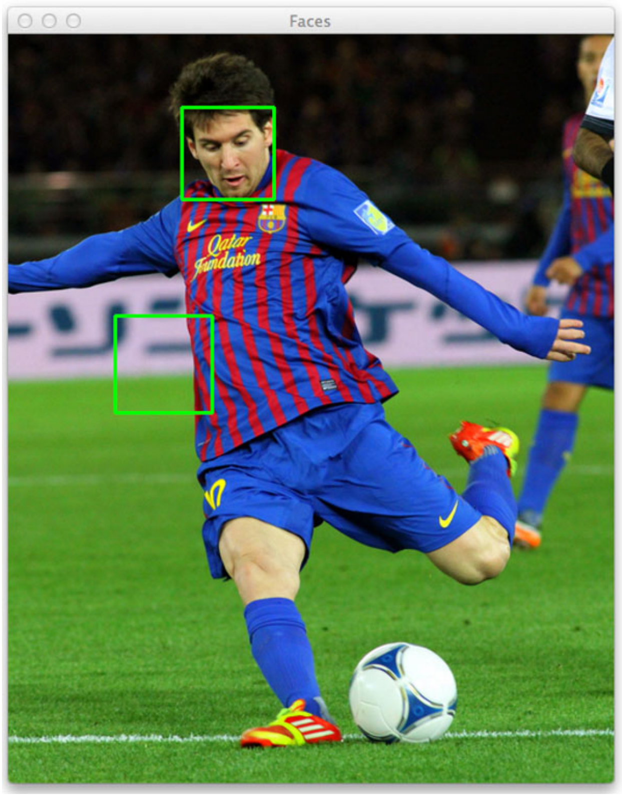
In this lesson, we will:

- Review the 6-steps required to build a custom object detection framework.
- Briefly discuss the difference between the standard 6-step approach proposed by Dalal and Triggs versus the Davis (i.e. dlib) method.

The problem with Haar cascades

Honestly, I really can't stand using the Haar cascade classifiers provided by OpenCV (i.e. the Viola-Jones detectors) — and that is why we are working on our own suite of classifiers. While cascade methods are extremely fast, they leave much to be desired. If you've ever used OpenCV to detect faces (or worked through the **face detection** lesson), you'll know exactly what I'm talking about:

Feedback



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_false_detection.jpg)

FIGURE 1: EXAMPLE OF FALSELY DETECTING A FACE IN AN IMAGE. THIS IS A COMMON PROBLEM WHEN USING HAAR CASCADES AND THE CV2.DETECTMULTISCALE FUNCTION.

In order to detect faces/humans/objects/whatever in OpenCV (and remove the false positives), you'll spend a lot of time tuning the `cv2.detectMultiScale` parameters. And again, there is no guarantee that the exact same parameters will work from image-to-image. This makes batch-processing large datasets for face detection (or any other type of object detection) a tedious task, since you'll be very concerned with either (1) falsely detecting faces or (2) missing faces entirely, simply due to poor parameter choices on a per image basis.

There is also the problem that the Viola-Jones detectors ***are nearing 15 years old.*** If this detector were a nice bottle of Cabernet Sauvignon, I might be pretty stoked right now. But the field has advanced substantially since then. Back in 2001, the Viola-Jones detectors were state-of-the-art, and they were certainly a huge motivating force behind the incredible new advances we have in object detection today.

Now, the Viola-Jones detector isn't our only choice for object detection. We have object detection using **keypoints (https://gurus.pyimagesearch.com/lessons/keypoint-detectors/)**, **local invariant descriptors (https://gurus.pyimagesearch.com/lessons/local-invariant-descriptors/)**, and **bag of visual words models (https://gurus.pyimagesearch.com/lessons/bag-of-visual-words-for-classification/)**. We have Histogram of Oriented Gradients. We have deformable parts models. Exemplar models. And we are now utilizing Deep Learning with pyramids to recognize objects at different scales!

All that said, even though the Histogram of Oriented Gradients descriptor for object recognition is nearly a decade old, it is still heavily used today — and with fantastic results. The Histogram of Oriented Gradients method suggested by Dalal and Triggs in their seminal 2005 paper, *Histogram of Oriented Gradients for Human Detection* (https://gurus.pyimagesearch.com/wp-content/uploads/2015/05/dalal_2005.pdf), demonstrated that the Histogram of Oriented Gradients (HOG) image descriptor and a Linear Support Vector Machine (SVM) could be used to train highly accurate object classifiers — or in their particular study, human detectors.

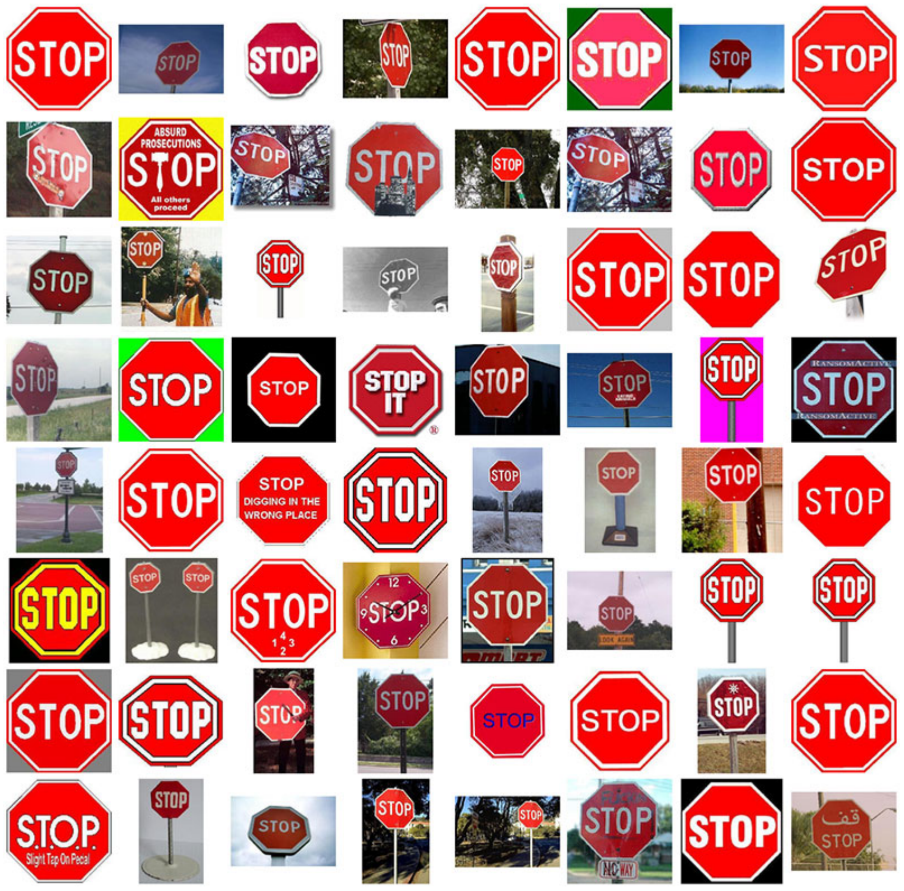
The 6-step framework

We'll review each of the following steps in detail later in this module. The primary goal of this lesson is to familiarize ourselves with the general process for training an object detector using Histogram of Oriented Gradients and a Linear SVM. The algorithm goes a little something like this:

Step 1:

Sample *P* positive samples from your training data of the object(s) you want to detect, and extract HOG descriptors from these samples.

For example, given our training dataset of CALTECH-101 stop signs below:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_stop_signs.jpg)

FIGURE 2: STOP SIGN IMAGES FROM THE CALTECH-101 DATASET.

We would (1) extract the bounding box (included with the training data for the images) of the object, followed by (2) computing HOG features over this ROI. These HOG features would serve as your *positive examples* (i.e. the examples containing the object we want to detect) that we'll later feed into our Linear SVM.

Step 2:

Sample *N* negative samples from a *negative training set* that ***does not contain*** any of the objects you want to detect, and extract HOG descriptors from these samples as well. In practice *N* >> *P*.

Continuing our example of building a custom stop sign detector, our first step is to find a dataset of images that *do not contain* any stop signs. A common choice when building custom object detectors is to use the **13 Natural Scene Categories (http://vision.stanford.edu/resources_links.html)** dataset of nature images:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_natural_scenes.jpg)

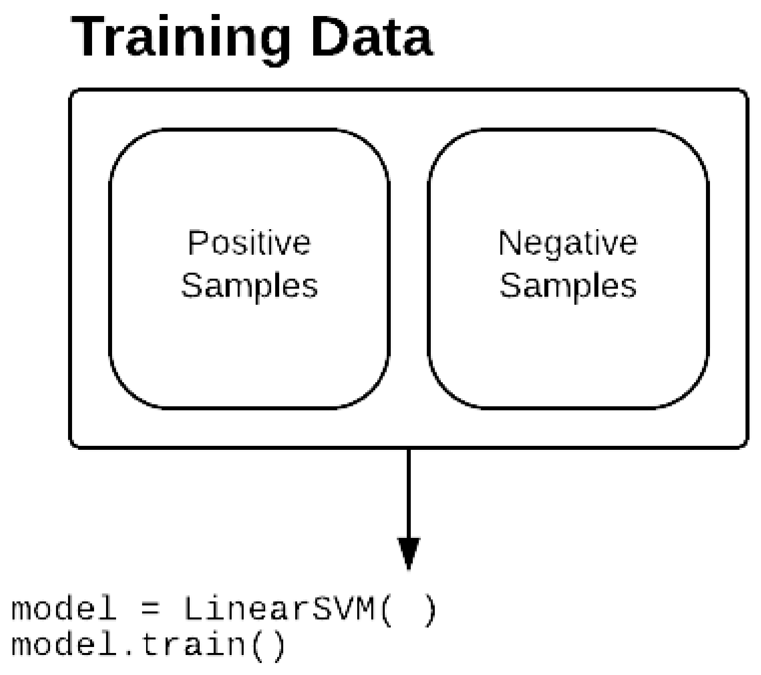
FIGURE 3: SAMPLES FROM THE 13 NATURAL SCENE CATEGORIES DATASET, A POPULAR DATASET USED FOR *NEGATIVE SAMPLES* WHEN TRAINING A CUSTOM OBJECT DETECTOR.

It's ***extremely unlikely*** that any image from this dataset will contain a stop sign. From there, all we need to do is *randomly sample* image patches from this dataset and extract HOG features from them. These HOG features will be our *negative examples* (i.e. examples that *do not contain* the object we want to detect) that we'll use to train our Linear SVM.

Step 3:

Train a Linear Support Vector Machine on your positive and negative samples.

Feedback



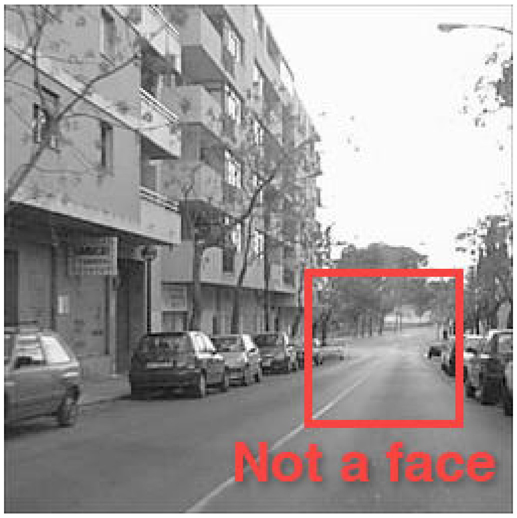
(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_initial_training.png)

FIGURE 3: TO TRAIN OUR LINEAR SVM, WE NEED TO SUPPLY THE TRAINING DATA (I.E. HOG FEATURE VECTORS) ASSOCIATED WITH BOTH THE POSITIVE AND NEGATIVE SAMPLES

Step 4:

Apply hard-negative mining. For each image and each possible scale (i.e. **image pyramid (https://gurus.pyimagesearch.com/topic/image-pyramids/)**) of each image in your negative training set, apply the **sliding window technique (https://gurus.pyimagesearch.com/topic/sliding-windows/)** and slide your window across the image. At each window, compute your HOG descriptors and apply your classifier. If your classifier (incorrectly) classifies a given window as an object (and it will; there will absolutely be false-positives), record the feature vector associated with the false-positive patch along with the probability of the classification.

In the example below, the red bounding box indicates an image patch that a classifier *false reported* as a face. In order to make a detector more robust (and reduce these types of false positives), we'll store the HOG feature vector associated with this bounding box and use it to improve our classifier.



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_hard_negative.jpg)

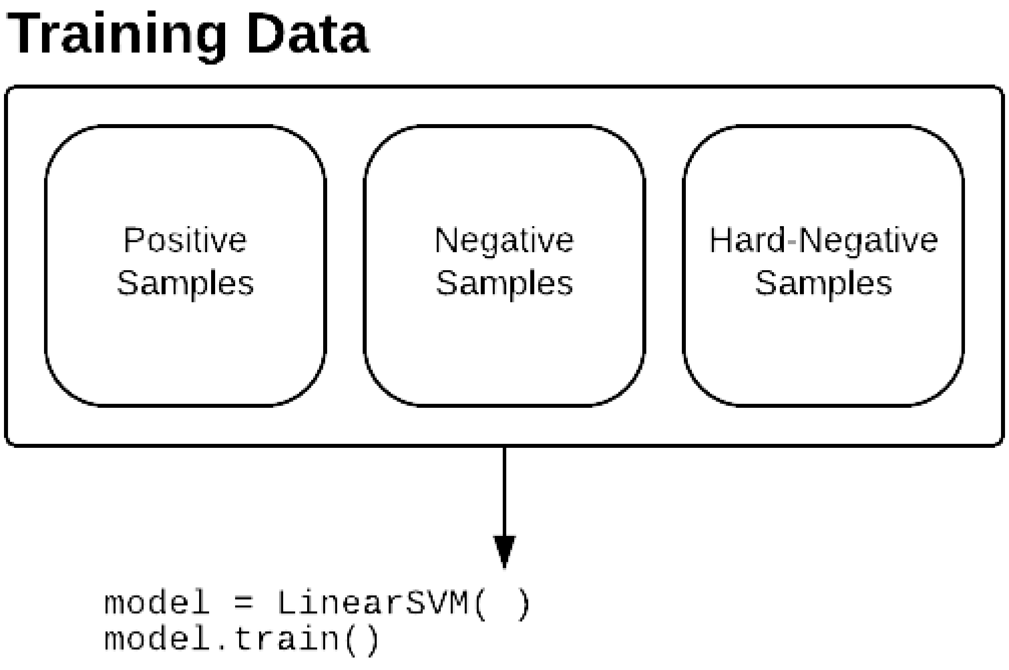
FIGURE 4: AN EXAMPLE OF APPLYING HARD NEGATIVE MINING. WE APPLY A SLIDING WINDOW AT EACH SCALE OF AN IMAGE, EXTRACT HOG FEATURES, AND PASS THEM OFF TO OUR LINEAR SVM. IF THE SVM REPORTS A POSITIVE CLASSIFICATION, THEN WE STORE THE HOG FEATURE VECTOR ASSOCIATED WITH THE IMAGE AS ADDITIONAL TRAINING DATA.

This approach is called *hard-negative mining* and allows us to reduce the number of false positives in our final detector.

Feedback

Step 5:

Take the false-positive samples found during the hard-negative mining stage, sort them by their confidence (i.e. probability), and re-train your classifier using these hard-negative samples:



[(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_retrain.png)](https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_retrain.png)

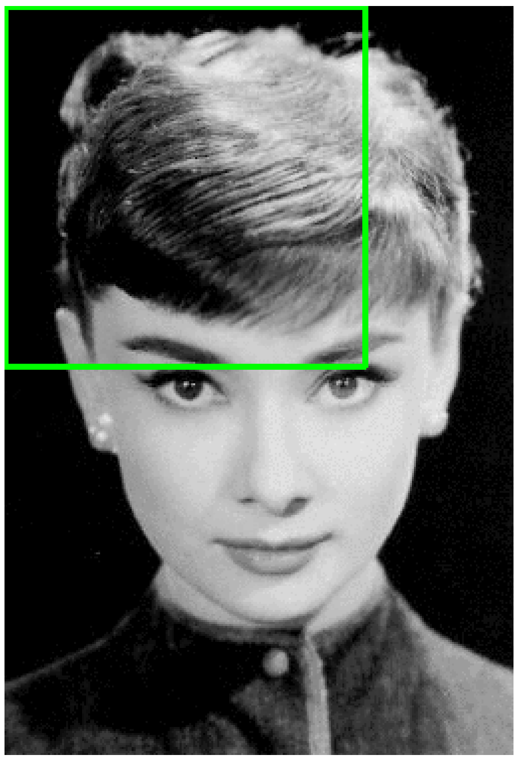
FIGURE 5: TO IMPROVE THE ACCURACY OF OUR OBJECT DETECTOR, WE RE-TRAIN IT USING THE HOG FEATURE VECTORS ASSOCIATED WITH THE POSITIVE, NEGATIVE, AND HARD-NEGATIVE DATA.

Note: You can iteratively apply **Steps 4 and 5**, but in practice, one stage of hard-negative mining usually (but not always) tends to be enough. The gains in accuracy on subsequent runs of hard-negative mining tend to be minimal.

Feedback

Step 6:

Your classifier is now trained and can be applied to your test dataset. Again, just like in **Step 4**, for each image in your test set, and for each scale of the image, apply the sliding window technique:

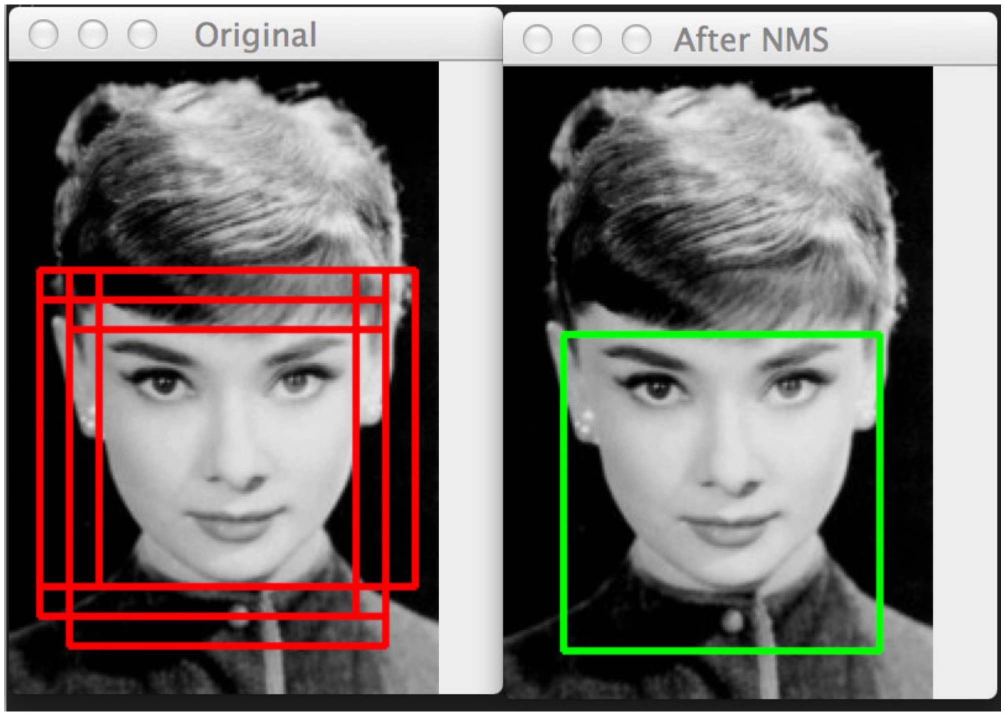


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/sliding_window_example.gif)

FIGURE 6: EXAMPLE OF THE SLIDING WINDOW APPROACH WHERE WE SLIDE A WINDOW FROM LEFT-TO-RIGHT AND TOP-TO-BOTTOM. AT EACH STOP ALONG THE WAY, WE EXTRACT HOG FEATURES AND PASS THEM ON TO OUR CLASSIFIER TO "DETECT" THE PRESENCE OF A GIVEN OBJECT (IN THIS CASE, A FACE).

Feedback

At each window, extract HOG descriptors and apply your classifier. If your classifier detects an object with sufficiently large probability, record the bounding box of the window. After you have finished scanning the image, apply non-maximum suppression to remove redundant and overlapping bounding boxes:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/6step_framework_nms.jpg)

FIGURE 7: *(LEFT)* DETECTING MULTIPLE OVERLAPPING BOUNDING BOXES AROUND THE FACE WE WANT TO DETECT. *(RIGHT)* APPLYING NON-MAXIMUM SUPPRESSION TO REMOVE THE REDUNDANT BOUNDING BOXES.

Feedback

Notice on the *left*, we have 6 overlapping bounding boxes that have correctly detected Audrey Hepburn's face. However, these 6 bounding boxes all refer to the same face — we need a method to suppress the 5 smallest bounding boxes in the region, keeping only the largest one, as seen on the *right*.

This is a common problem, no matter if you are using the Viola-Jones based method or following the Dalal-Triggs paper.

There are multiple ways to remedy this problem. Triggs et al. suggests to use the Mean-Shift algorithm (http://en.wikipedia.org/wiki/Mean-shift) to detect multiple modes in the bounding box space by utilizing the *(x, y)* coordinates of the bounding box, as well as the logarithm of the current scale of the image.

I've personally tried this method and wasn't satisfied with the results. Instead, you're much better off relying on a *strong classifier* with *higher accuracy* (meaning there are very few false positives) and then applying non-maximum suppression to the bounding boxes. We'll be reviewing a very fast and efficient non-maximum suppression algorithm later in this module.

Extensions and other approaches

It's worth noting that the HOG + Linear SVM method we used in the **object detection made easy (https://gurus.pyimagesearch.com/topic/object-detection-made-easy/)** lesson takes a slightly different approach than the standard 6-step framework proposed above. In his 2015 paper, *Max-Margin Object Detection (https://gurus.pyimagesearch.com/wp-content/uploads/2015/10/davis_2015.pdf)*, Davis details three changes that are made inside the dlib library to improve object detection.

The first change is in relation to the HOG sliding window and non-maximum suppression approach. Instead of extracting features from *both* a positive and a negative dataset, the dlib method optimizes the number of mistakes the HOG sliding window makes on *each* training image. This implies that the *entire* training image is used to both (1) extract the positive example, and (2) extract *negative samples* from all other regions of the image. This *completely alleviates* the need for a negative training set and the requirement of hard-negative mining. This is one of the reasons why the *Max-Margin Object* detection method is so fast.

Secondly, dlib also takes into account non-maxima suppression during the actual training phase. We normally only apply NMS to obtain our final set of bounding boxes, but in this case, we can actually use NMS during the training phase. This helps reduce false-positives *substantially* and again alleviates the need for hard-negative mining.

Finally, the dlib uses an extremely accurate algorithm to find the optimal hyperplane separating the two image classes. This method obtains much a higher accuracy (with a lower false-positive rate) than many other state-of-the-art object detectors.

The mathematics behind Davis' method are outside the scope of this course, but if you're interested, please see his original paper (https://gurus.pyimagesearch.com/wp-content/uploads/2015/10/davis_2015.pdf). We'll be revisiting the dlib method for object detection later in this module.

Summary

In this lesson, we had a little bit of a history lesson regarding object detectors and why Haar cascades, while highly popular, are less used than the more robust HOG + Linear SVM approach. We also had a sneak peek into the Python framework that we will be constructing to build object detection in images.

Last, we looked at the 6-steps required to build a custom object detector using the Histogram of Oriented Gradients descriptor and a Linear Support Vector Machine.

In our next lesson, we'll be preparing our image datasets for training, followed by constructing our HOG descriptor, extracting features, and performing the initial training phase of our classifier.

Quizzes		Status
1	The 6-Step Framework Quiz (https://gurus.pyimagesearch.com/quizzes/the-6-step-framework-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/sliding-windows-and-image-pyramids/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/preparing-your-training-data/)

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)

Feedback

Search

Feedback

/