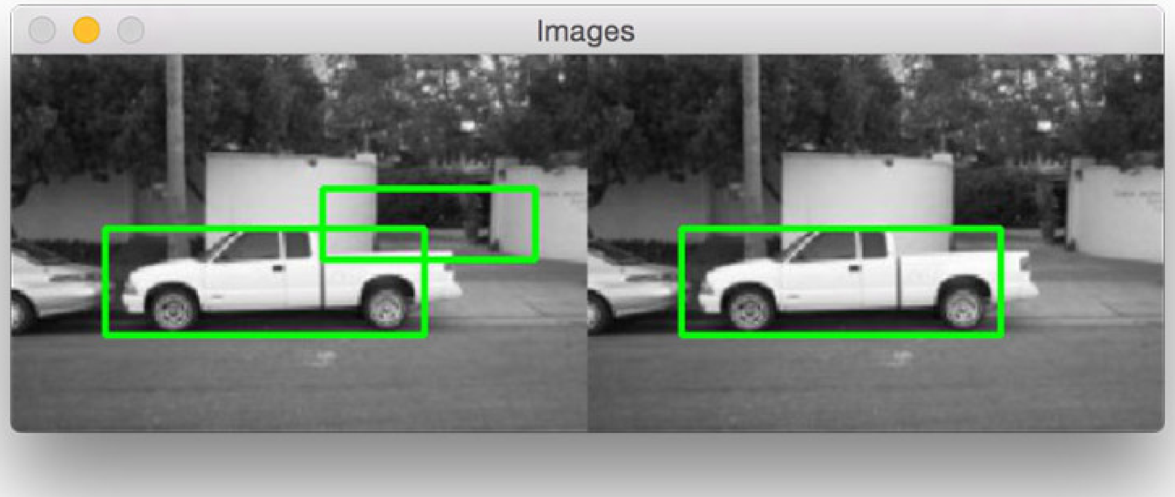
2.10: Re-training and running your classifier



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/hog_retraining_image_0016.jpg)

In our previous lesson, we applied **hard-negative mining (https://gurus.pyimagesearch.com/lessons/hard-negative-mining/)** to obtain additional negative training examples. Now that we have our extra negative examples, we need to re-train our classifier using all three sets of training data:

1. Original positives

2. Original negatives
3. Hard-negatives

After re-training our Linear SVM on this data, the detector should report *less false-positives* while still *retaining the true-positive detections*.

Objectives:

In this lesson, we will:

- Use hard-negative mining examples mined from our previous lesson to re-train our object detector.
- Test our newly re-trained object detector and see if our false-positive detection rate has been reduced.

Re-training and running your classifier

Now that we have applied hard-negative mining, the next step is to re-train our classifier. To accomplish this, we'll use the same `train_model.py` from our **initial training phase lesson (https://gurus.pyimagesearch.com/lessons/the-initial-training-phase/)**. We have already reviewed this script earlier, but I have included it here as a matter of completeness:

```
train_model.py                                                                   Python
```

```
 1 # import the necessary packages
 2 from __future__ import print_function
 3 from pyimagesearch.utils import dataset
 4 from pyimagesearch.utils import Conf
 5 from sklearn.svm import SVC
 6 import numpy as np
 7 import argparse
 8 import pickle
 9
10 # construct the argument parser and parse the command line arguments
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-c", "--conf", required=True,
13     help="path to the configuration file")
14 ap.add_argument("-n", "--hard-negatives", type=int, default=-1,
15     help="flag indicating whether or not hard negatives should be used")
16 args = vars(ap.parse_args())
17
18 # load the configuration file and the initial dataset
19 print("[INFO] loading dataset...")
20 conf = Conf(args["conf"])
21 (data, labels) = dataset.load_dataset(conf["features_path"], "features")
22
23 # check to see if the hard negatives flag was supplied
24 if args["hard_negatives"] > 0:
25     print("[INFO] loading hard negatives...")
26     (hardData, hardLabels) = dataset.load_dataset(conf["features_path"], "hard_negatives")
27     data = np.vstack([data, hardData])
28     labels = np.hstack([labels, hardLabels])
29
30 # train the classifier
31 print("[INFO] training classifier...")
32 model = SVC(kernel="linear", C=conf["C"], probability=True, random_state=42)
33 model.fit(data, labels)
34
35 # dump the classifier to file
36 print("[INFO] dumping classifier...")
37 f = open(conf["classifier_path"], "wb")
38 f.write(pickle.dumps(model))
39 f.close()
```

We can see on **Line 21** that we load the *original positive and negative examples* from disk. The *hard-negative examples* are then loaded on **Line 26** (provided the `--hard-negatives` switch is supplied) and used to update the `data` and `label` lists, respectively (**Lines 27 and 28**).

Lines 32 and 33 train our classifier, which is then serialized to disk on **Lines 37-39**.

To re-train our object detector using the hard-negative examples, just issue the following command:

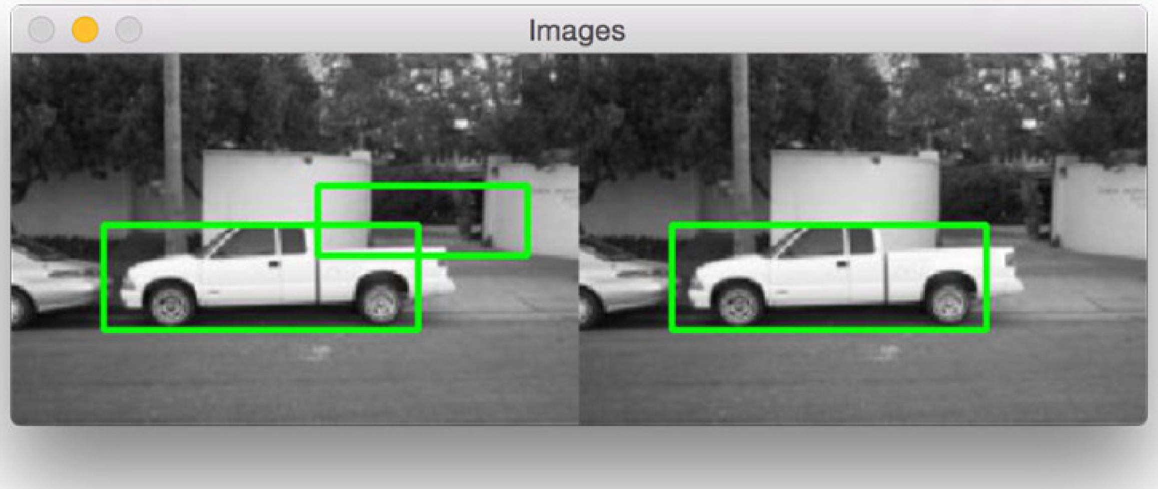
```
train_model.py                                                              Shell
$ python train_model.py --conf conf/cars.json --hard-negatives 1
```

Again, notice how the `--hard-negatives` switch is used to indicate that we want to include the hard-negative examples in the training process.

To test out our newly re-trained model, execute `test_model.py` , only this time, our newly re-trained object detector will be automatically used:

test_model.py Shell

```
$ python test_model.py --conf conf/cars.json \
    --image datasets/caltech101/101_ObjectCategories/car_side/image_0016.jpg
```

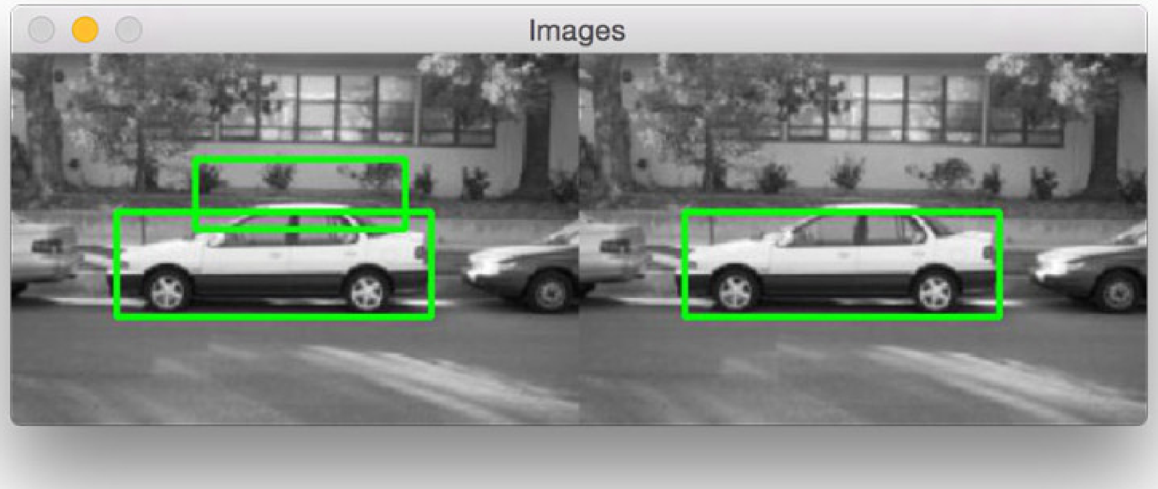


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/hog_retraining_image_0016.jpg)

FIGURE 1: *(LEFT)* BEFORE APPLYING HARD-NEGATIVE MINING. *(RIGHT)* AFTER APPLYING HARD-NEGATIVE MINING AND RE-TRAINING OUR CLASSIFIER. NOTICE HOW THE CLASSIFIER NO LONGER MIS-FIRES AND FALSELY REPORTS A CLASSIFICATION.

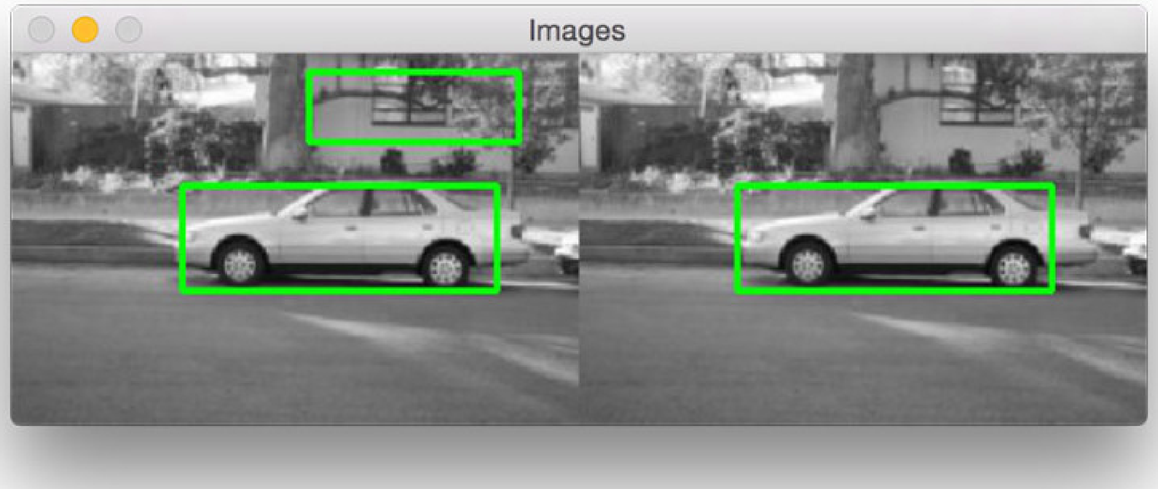
On the *left*, we have our original detections (after non-maxima suppression [NMS]) with *no hard-negative mining*. Then, on the *right*, we have the re-trained object detector results (after NMS) *after applying hard-negative mining*. As you can see, the false-positive detection has been removed.

The same is true for the following images as well:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/hog_retraining_image_0009.jpg)

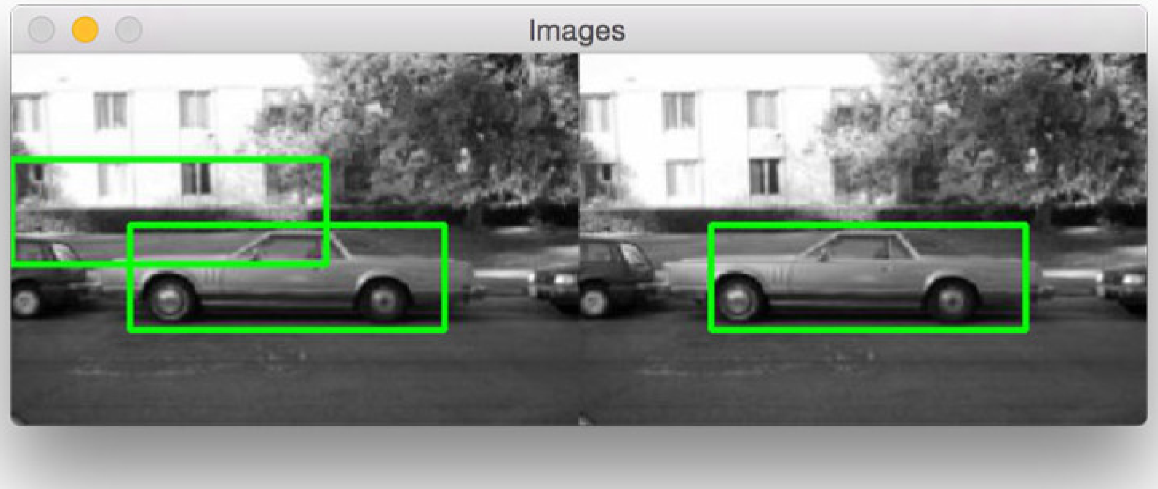
FIGURE 2: ANOTHER EXAMPLE OF HOW HARD-NEGATIVE MINING CAN BE USED TO REDUCE THE FALSE POSITIVE DETECTION RATE.



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/hog_retraining_image_0010.jpg)

FIGURE 3: REDUCING FALSE-POSITIVE DETECTIONS WITH HARD-NEGATIVE MINING.

Feedback



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/hog_retraining_image_0025.jpg)

FIGURE 4: A FINAL EXAMPLE OF HOW WE CAN USE HARD-NEGATIVE MINING TO GIVE EXTRA "INFORMATION" TO OUR CLASSIFIER AS TO WHAT IS A POSITIVE DETECTION AND WHAT IS NOT A POSITIVE DETECTION.

As all these results demonstrate, applying hard-negative mining has lead to a reduction in false-positive detections, with no loss in true-positive detections.

In our next lesson, we'll review the complete object detection pipeline to train a *custom face detector* from front-to-back.

Feedback

Summary

In this lesson, we re-trained our custom object detector using our hard-negative examples. By using our hard-negative examples as additional training data, we were able to reduce the false-positive detection rate of our classifier while still retaining the true detections.

Applying hard-negative mining ***nearly always*** increases detection accuracy, so it's something that should be performed (or at the very least tested) for your own custom object detectors.

Coming up next, we'll review our complete object detection framework and use the code we have developed in this module to construct a *face detector* using only a JSON configuration file and the command line.

Downloads:

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

Quizzes		Status
1	Re-training and Running Your Classifier Quiz (https://gurus.pyimagesearch.com/quizzes/re-training-and-running-your-classifier-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/hard-negative-mining/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/training-your-custom-object-detector/)

Course Progress

Feedback

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/)

Search

Feedback