2.9: Hard-negative mining

In our previous lessons, we took a **first shot at training a Linear SVM (https://gurus.pyimagesearch.com/lessons/the-initial-training-phase/)** to detect the presence of cars in images, followed by applying **non-maxima suppression (https://gurus.pyimagesearch.com/lessons/non-maxima-suppression/)** to prune multiple, overlapping bounding boxes.

Our first try at creating our custom object detector worked quite well, but we still had the issue of false-positive detections (i.e., the car being detected in an image when in reality there wasn't a car).

To reduce the number of false-positive detections (and therefore increase detection accuracy), we need to apply ***hard-negative mining***, which is exactly what the remainder of this lesson covers.

Objectives:

In this lesson, we will:

- Learn what hard-negative mining is.
- Implement hard-negative mining in our object detection framework.

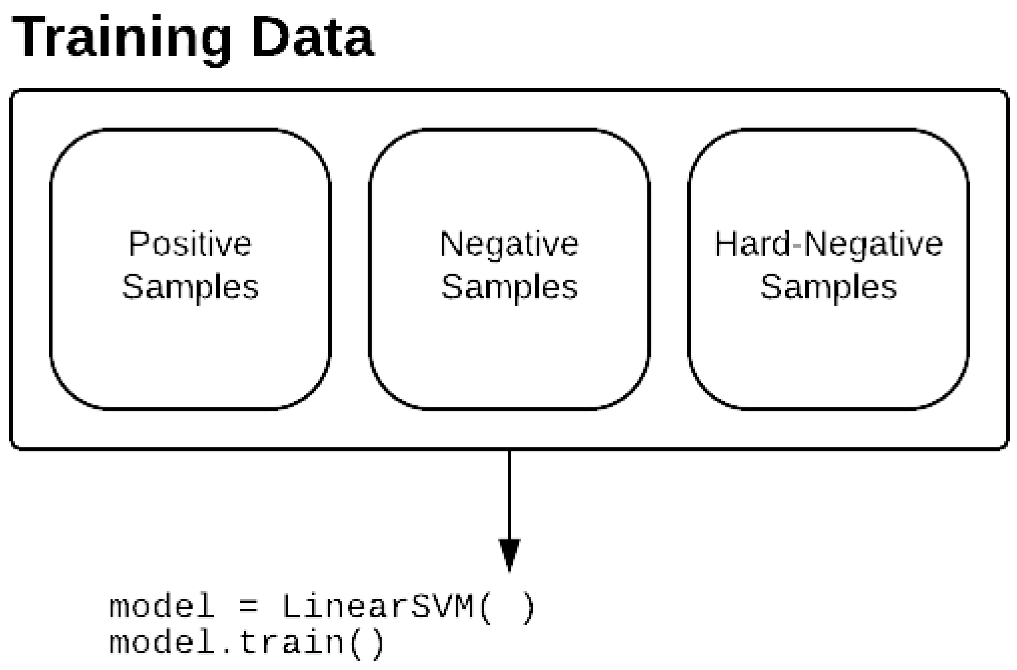
Hard-negative mining

Hard-negative mining is the brute-force process of obtaining additional ***negative samples*** from a training set.

We start by looping over our image dataset of negative images (i.e., the images that *do not* contain examples of the object we want to detect). For each image in this dataset, we construct an image pyramid and apply a sliding window at each layer. HOG features are extracted from each window and passed on to our Linear SVM for classification.

If a patch is (falsely) labeled as an object of interest (such as a patch containing a car, when in reality it does not), we take the HOG feature vector associated with the patch and add it as a hard-negative sample to our training set.

After applying hard-negative mining, we end up with a set of HOG feature vectors that our Linear SVM has misfired on. We then take our *original positive samples*, *original negative samples*, and our *hard-negatives*, and use them to re-train our Linear SVM:



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

FIGURE 1: 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.

After re-training our Linear SVM, the detector should perform better with this added knowledge and not make as many false-positive detections.

This process is called *hard-negative mining* since we are explicitly looping over our negative image dataset and exhaustively searching for image patches that cause our detector to misfire.

Let's see how we can implement hard-negative mining in our object detection framework.

The first step is to update our `cars.json` file:

cars.json JavaScript
```javascript
...
    /****
    * HARD NEGATIVE MINING
    ****/
    "hn_num_distraction_images": 50,
    "hn_window_step": 4,
    "hn_pyramid_scale": 1.5,
    "hn_min_probability": 0.51
}
```

Here, we indicate that we are going to examine 50 images from our negative dataset. This number can easily be increased to mine for more hard-negatives. We'll also supply a sliding window step of four pixels, a pyramid scale of 1.5, and a minimum probability of 0.51. If a given HOG feature vector reported was a positive classification with more than 51% probability, we'll mark the feature vector as a false-positive and add it to our training set.

Now that we have updated our JSON file, it's time to implement hard-negative mining:

hard_negative_mine.py Python
```python
# import the necessary packages
from __future__ import print_function
from pyimagesearch.object_detection import ObjectDetector
from pyimagesearch.descriptors import HOG
from pyimagesearch.utils import dataset
from pyimagesearch.utils import Conf
from imutils import paths
import numpy as np
import progressbar
import argparse
import pickle
import random
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--conf", required=True, help="path to the configuration file")
args = vars(ap.parse_args())
```

We'll start off by importing our necessary packages and parsing our command line arguments to grab the path to our `.json` configuration file.

From here, we can load our Linear SVM from disk, initialize our HOG descriptor, and instantiate the actual `ObjectDetector` :

```python
20 # load the configuration file and initialize the data list
21 conf = Conf(args["conf"])
22 data = []
23
24 # load the classifier, then initialize the Histogram of Oriented Gradients descriptor
25 # and the object detector
26 model = pickle.loads(open(conf["classifier_path"], "rb").read())
27 hog = HOG(orientations=conf["orientations"], pixelsPerCell=tuple(conf["pixels_per_cell"]),
28     cellsPerBlock=tuple(conf["cells_per_block"]), normalize=conf["normalize"], block_norm="L1")
29 od = ObjectDetector(model, hog)
30
31 # grab the set of distraction paths and randomly sample them
32 dstPaths = list(paths.list_images(conf["image_distractions"]))
33 dstPaths = random.sample(dstPaths, conf["hn_num_distraction_images"])
34
35 # setup the progress bar
36 widgets = ["Mining: ", progressbar.Percentage(), " ", progressbar.Bar(), " ", progressbar.ETA()]
37 pbar = progressbar.ProgressBar(maxval=len(dstPaths), widgets=widgets).start()
```

We'll also sample our negative example images on **Lines 32 and 33**.

As you'll undoubtedly note, this code is actually very similar to `test_model.py` from the previous lesson, only this time we are applying object detection to *multiple images*.

We are now ready to perform hard-negative mining:

hard_negative_mine.py Python

```python
39 # loop over the distraction paths
40 for (i, imagePath) in enumerate(dstPaths):
41     # load the image and convert it to grayscale
42     image = cv2.imread(imagePath)
43     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
44
45     # detect objects in the image
46     (boxes, probs) = od.detect(gray, conf["window_dim"], winStep=conf["hn_window_step"],
47         pyramidScale=conf["hn_pyramid_scale"], minProb=conf["hn_min_probability"])
48
49     # loop over the bounding boxes
50     for (prob, (startX, startY, endX, endY)) in zip(probs, boxes):
51         # extract the ROI from the image, resize it to a known, canonical size, extract
52         # HOG features from the ROI, and finally update the data
53         roi = cv2.resize(gray[startY:endY, startX:endX], tuple(conf["window_dim"]),
54             interpolation=cv2.INTER_AREA)
55         features = hog.describe(roi)
56         data.append(np.hstack([[prob], features]))
57
58     # update the progress bar
59     pbar.update(i)
```

We start off by looping over our sampled distraction images on **Line 40**.

For each of these images, we apply our object detector on **Lines 46 and 47** using the supplied sliding window step, pyramid scale, and minimum required probability.

If we end up (falsely) detecting objects in a distraction image, we loop over the bounding boxes associated with the regions on **Line 50**. For each of these regions, we extract HOG features and update our `data` list with the feature vector itself, along with its associated probability.

The last step of hard-negative mining is to take these feature vectors, sort them by probability, and dump the data to disk:

```python
hard_negative_mine.py                                                     Python
61 # sort the data points by confidence
62 pbar.finish()
63 print("[INFO] sorting by probability...")
64 data = np.array(data)
65 data = data[data[:, 0].argsort()[::-1]]
66
67 # dump the dataset to file
68 print("[INFO] dumping hard negatives to file...")
69 dataset.dump_dataset(data[:, 1:], [-1] * len(data), conf["features_path"], "hard_negatives",
70     writeMethod="a")
```

We sort the hard-negative feature vectors so that vectors with *larger probability* are placed *closer to the front of the list*. We do this just in case *all* of our positive + negative + hard negative examples do not fit into main memory, allowing us to easily take only the *most confident* false-positive detections.

Finally, **Lines 68-70** update our HDF5 feature database by creating a new dataset (in append mode to ensure we don't overwrite our original dataset) named `hard_negatives` and dumping the features to this dataset.

To apply hard-negative mining, just execute the following command:

```shell
hard_negative_mine.py                                                      Shell
$ python hard_negative_mine.py --conf conf/cars.json
```

On my machine, applying hard-negative mining to *only 50 images* takes **16m 39s**.

As you can see, the biggest issue with hard-negative mining is that *it's extremely slow and tedious*. The slowness is also compounded by the fact that we don't know exactly how many false-positives we need to obtain optimal accuracy.

Given this, it becomes very hard to answer the questions, *"How many hard-negatives are needed to increase accuracy?"*, and *"How many hard-negatives is 'too many', resulting in deteriorated detector performance?"*

In reality, the only way to know is to run experiments and evaluate the results. As I've mentioned in previous lessons, parameter tuning is a bit of a "black art", but gets easier the more you experiment and work with datasets.

Finally, it's worth noting that both object detection and hard-negative mining can be made to run in parallel. In order to make object detection run in parallel, you can distribute each of the image pyramid scales to the cores of your processor. Doing this can dramatically decrease the amount of time it takes to perform object detection, but you'll always be limited by the amount of time it takes to process the upper layers (i.e., spatially larger layers) of the image pyramid.

However, introducing multi-processing code can complicate your codebase quite a bit and is left as an exercise to the reader.

Summary

In this lesson, we learned how to perform hard-negative mining, the process of "mining" your negative example images for false-positive detections. The HOG feature vectors associated with these false-positive detections are then used as additional training data for your Linear SVM.

By applying hard-negative mining, we can reduce the number of false-positive detections and thus increase the overall detection accuracy. In our next lesson, we'll learn how to take the results of our hard-negative mining script and re-train our Linear SVM using them.

Feedback

Downloads:

Download the SceneClass13 Dataset
(https://gurus.pyimagesearch.com/protected/code/object_detector/sceneclass13

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

Quizzes		Status
1	Hard-negative Mining Quiz (https://gurus.pyimagesearch.com/quizzes/hard-negative-mining-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/non-maxima-suppression/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/re-training-and-running-your-classifier/)

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)

Feedback

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