Assignment 4: Face detection with a sliding window

Overview

The sliding window model is conceptually simple: independently classify all image patches as being object or non-object. Sliding window classification is the dominant paradigm in object detection and for one object category in particular -- faces -- it is one of the most noticeable successes of computer vision. Because you have already implemented the SIFT descriptor, you will not be asked to implement HoG. You will be responsible for the rest of the detection pipeline, though -- handling heterogeneous training and testing data, training a linear classifier (a HoG template), and using your classifier to classify millions of sliding windows at multiple scales. Fortunately, linear classifiers are compact, fast to train, and fast to execute. A linear SVM can also be trained on large amounts of data, including mined hard negatives.

Details

- For this project, you need to implement the following parts:
- Extracting features: `get_positive_features()`, `get_random_negative_features()`
- Mining hard negatives: `mine_hard_negs()`
- Train a linear classifier: `train_classifier()`
- Detect faces on the test set: `run_detector()`

Get features from positive samples

You will implement `get_positive_features()` to load cropped positive trained examples (faces) and convert them to HoG features. We provide the default feature parameters in `proj5.ipynb`. (You are free to try different parameters, but it doesnt guarantee better performance.) For improved performance, You can try mirroring or warping the positive training examples to augment your training data. Please refer to the documentation for more details.

Get features from negative samples

You will implement `get_random_negative_features()` to sample random negative examples from scenes which contain no faces and convert them to HoG features. The output feature dimension from `get_random_negative_features()` and `get_positive_features()` should be the same. For best performance, you should sample random negative examples at multiple scales. Please refer to the documentation for more details.

Mine hard negatives

You will implement `mine_hard_negs()` as discussed in **Dalal and Triggs**, and demonstrate the effect on performance. The main idea of hard negative mining is that you use the trained classifier to find false-positive examples, and then include them in your negative training data, so you can train the classifier again to improve the performance. This might not be very effective for frontal faces unless you use a more complex feature or classifier. You might notice a bigger difference if you artificially limit the amount of negative training data (e.g. a total budget of only 5000 negatives).

Train a linear classifer

You will implement `train_classifier()` to train a linear classifier from the positive and negative features by using scikit-learn LinearSVC. The regularization constant C is an important parameter that affects the classification performance. Small values seem to work better e.g. 1e-4, but you can try other values.

Detect faces on the test set

You will implement `run_detector()` to detect faces on the testing images. For each image, You will run the classifier with a sliding window at multiple scales and then call the provided function `non_max_suppression_bbox()` to remove duplicate detections.

Your code should convert each test image to HoG feature space for each scale. Then you step over the HoG cells, taking groups of cells that are the same size as your learned template, and classifying them. If the classification is above some confidence, keep the detection and then pass all the detections for an image to `non_max_suppression_bbox()`. The ouputs are the coordinates ([xl, yl, xh, yh]) of all the detected faces and their corresponding confidences and testing image indices. Please See `run_detector()` documentation for more details.

`run_detector()` will have (at least) two parameters which can heavily influence performance:

- how much to rescale each step of your multiscale detector: More scales usually help.
- threshold for a detection: If your recall rate is low and your detector still has high precision at its highest recall point, you can improve your average precision by reducing the threshold for a positive detection.

Using the starter code (`proj5.ipynb`)

The top-level starter code `proj5.ipynb` provides data loading, evaluation and visualization functions and calls the functions in `student_code.py`. If you run the code unmodified, it will return random bounding boxes in each test image. It will even do non-maximum suppression on the random bounding boxes to give you an example of how to call the function. detect random faces in the test images. We provide the following functions to help you examine the performance:

`report_accuracy()`: Show some statistics about your trained classifier. The training accuracy should be really low. Good classification performance doesnt guarantee good detection performance, but its a good sanity check. We also provide the visualization that you can see how well separated the positive and negative examples are at training time.

`visualize_hog()`: Visualize the HOG feature template to examine if the detector has learned a meaningful representation.

`evaluate_detections()`: Compute ROC curve, precision-recall curve, and average precision. Youre not allowed to change this function.

Performance to aim for:

- random (stater code): 0.001 AP
- single scale: ~ 0.3 to 0.4 AP
- multiscale: ~ 0.8 to 0.9 AP

`visualize_detections_by_image()`: Visualize detections in each image (with ground truth). You also can use `visualize_detections_by_image_no_gt()` for test cases which have no ground truth annotations.

Data

The choice of training data is critical for this task. While an object detection system would typically be trained and tested on a single database (as in the Pascal VOC challenge), face detection papers were previously trained on heterogeneous, even proprietary, datasets. As with most of the literature, we will use three databases: (1) positive training crops, (2) non-face scenes to mine for negative training data, and (3) test scenes with ground truth face locations.

You are provided with a positive training database of 6,713 cropped 36x36 faces from the Caltech Web Faces project. We arrived at this subset by filtering away faces which were not high enough resolution, upright, or front facing. There are many additional databases available For example, see Figure 3 in Huang et al. and the LFW database described in the paper. You are free to experiment with additional or alternative training data for extra credit.

Non-face scenes, the second source of your training data, are easy to collect. We provide a small database of such scenes from Wu et al. and the SUN scene database. You can add more non-face training scenes, although you are unlikely to need more negative training data unless you are doing hard negative mining for extra credit.

The most common benchmark for face detection is the CMU+MIT test set. This test set contains 130 images with 511 faces. The test set is challenging because the images are highly compressed and quantized. Some of the faces are illustrated faces, not human faces. For this project, we have converted the test sets ground truth landmark points in to Pascal VOC style bounding boxes. We have inflated these bounding boxes to cover most of the head, as the provided training data does. For this reason, you are arguably training a "head detector" not a "face detector" for this project.

Useful Functions

`vlfeat.hog.hog()`. The main function to extract the HOG feature. For detailed information, please refer to the link or the comments in `student_code.py` and `proj5.ipynb`.

`sklearn.svm.LinearSVC()`. Linear support vector classification. The regularization constant C is the most critical parameter affecting the classification performance.

Banned Functions

You should not use any third-party library that do multi-scale object detection, such as `cv2.HOGDescriptor().detectMultiScale()`. You need to implement the detection function by yourself. You may also not use anyone elses code. The main principle is that y ou should not use any third-party library that can directly perform one of these functions: `get_positive_features()`, `get_random_negative_features()`, `mine_hard_negs()`, `train_classifier()` or `run_detector()`.

Handing in

- Hand in your project as a zip file through BlackBoard.
- Please do not include the data sets in your handin.

Rubric

- +20 pts: Use the training images to create positive and and negative training HoG features.
- +20 pts: Mine hard negatives.
- +10 pts: Train linear classifier.
- +25 pts: Create a multi-scale, sliding window object detector.
- +25 pts: Writeup report with design decisions and evaluation.
- -10 pts: Lose 10 points for your ugly code.