How to train a sliding-window object detector

Stephen Gould
sgould@stanford.edu

January 31, 2009

1 Background

Sliding-window object detection is a popular technique for identifying and localizing objects in an image. The approach involves scanning the image with a fixed-size rectangular window and applying a classifier to the sub-image defined by the window. The classifier extracts image features from within the window (sub-image) and returns the probability that the window (tightly) bounds a particular object. The process is repeated on successively scaled copies of the image so that objects can be detected at any size. Usually non-maximal neighborhood suppression is applied to the output to remove multiple detections of the same object.

There are two main drawbacks to sliding-window object detectors:

- In the vanilla version, the detectors fail to take into account contextual cues;
- The window has a fixed aspect ratio making it difficult for articulated objects or objects with large intra-class variation.

This article describes how to train a sliding-window object detector using the STAIR Vision Library. Our detectors are based on the patch-based detectors introduced by Torralba (see references below).

2 Step-by-step Guide

You will need to prepare a dataset of positive and negative training examples (jpg images). These can be collected from image databases, Google Image Search, or extracted from images of scenes collected in-house. Experience has shown that around 200-500 positive examples and 10000-20000 negative example results fairly decent performance.

- The positive examples should contain cropped images that leave a small border around the edge so that the detector can take advantage of edge features. They should also contain a wide variety of instances from the object class in the same orientations in which you want them to be detected.
- All positive examples should have approximately the same aspect ratio. However, you do not need to scale the images to the same size (the training code will do that automatically). If some objects in the class (or some orientations) exhibit a different aspect ratio then you can crop the images to be square with the object centered.
- The negative example can be randomly snipped out of a training video or image sequence. Be sure to collect examples at different scales. Note that by constructing a negative set this way there will be high correlation in these examples so you will need lots of them. The same negative set can be used for training multiple object classes.

The positive and negative examples should be placed in separate directories. As mentioned above, the images need not be scaled, but training will run faster if they are already at the base scale.

If you have collected a sequence of images and annotated them with ground-truth labels (see *How to label ground-truth objects for training and evaluating object detectors*), then you can create the positive and negative training examples by clipping out from these images:

```
mkdir data/objects/<OBJECT>
mkdir data/objects/negative
bin/buildTrainingImageDataset -object <OBJECT> -resize 32 32 \
    -directory data/objects <IMAGE_SEQUENCE>.xml <GT_LABELS>.xml
```

2.1 Building a Patch Dictionary

The first step to training the detector is to build a patch dictionary. You need to decide on the base size for your classifier. This is the smallest scale at which objects will be detected. A typical size is 32-by-32 (if your object has a square aspect ratio), but it can be larger for bigger objects. A small number of patch features (typically 10 per example) are randomly selected from the positive training images to construct the initial dictionary. If this dictionary is too large you will run out of memory during training. The dictionary will eventually be trimmed to remove patches which don't contribute to the classifier.

```
bin/buildPatchDictionary -n 10 -o models/<OBJECT>.dictionary.xml data/objects/<OBJECT> 32 32
```

2.2 Computing the Patch Response Cache

Although the object detectors could be trained directly from images, it is best to first compute a cache of all the patch responses. This is because the patch response calculations take a long time (especially when the initial dictionary is large) and caching the responses allows you to recover from crashes (or compute responses in parallel).

A response cache should be created separately for the positive and negative training examples. Here is an example command-line:

```
bin/buildPatchResponseCache -maxImages 1000 data/objects/<OBJECT> \
    /tmp/cache-<OBJECT>-pos models/<OBJECT>.dictionary.xml
bin/buildPatchResponseCache -maxImages 20000 data/objects/negative \
    /tmp/cache-<OBJECT>-neg models/<OBJECT>.dictionary.xml
```

Training on such a large set of examples is a computational challenge, especially with a large initial dictionary. One computational trick is to train on a smaller set of negative examples first (say, 2000) to reduce the dictionary size (see Trimming the Dictionary) and then re-train on the full set of examples. This is implemented in the `trainObjectDetector.pl` script.

2.3 Training the Boosted Detector

Once the patch response cache is built, the boosted detector can be trained quickly and simply:

```
bin/trainObjectDetector -o models/<OBJECT>.model /tmp/cache-<OBJECT>-pos /tmp/cache-<OBJECT>-neg
```

The code will output the accuracy on the training dataset. Note that this does not reflect how well the detector will perform during testing, but can indicate potential problems (for example, very poor performance).

2.4 Trimming the Dictionary

Computing patch responses is expensive so the final stage in training is to remove the patches which are not actually used by the detector. The Perl script trimDictionary.pl will remove unused patches from the dictionary and renumber the features indexes in the learned model file accordingly.

```
svl/scripts/trimDictionary.pl models/<OBJECT>.dictionary.xml models/<OBJECT>.model
```

2.5 Evaluating Performance

Object detectors are usually evaluated in terms of their precision (percentage of detections that were correct) and recall (percentage of true positives detected), although a number of other metrics can also be used. In general there is a trade-off between precision and recall. This trade-off can be traced out on a PR-curve (by varying the threshold at which the detector decides it has found an object).

A different set of images to the ones used for training should be used for evaluation. Evaluation occurs in two stages. First, the object detectors are run over the set of test images with low threshold (say, 0.01). This ensures that the high recall part of the PR-curve can be recovered. Second, the PR-curve is generated by sorting the detections and counting false and true positives as the detection threshold is varied. The following commands illustrate the process.

```
bin/classifyImages -o <DETECTIONS>.xml -t 0.01 -v models/<OBJECT>.dictionary.xml \
    models/<OBJECT>.model <IMAGE_SEQEUNCE>.xml
bin/scoreDetections -pr experiments/pr -v <GT_LABELS>.xml <DETECTIONS>.xml
```

2.6 Re-training on False-Positives

Often performance can be improved by re-training the object detector with a set of negative examples that has been augmented with false-positives from an initial run of the object detector. Intuitively, this makes a much better negative training set than the random patches chosen initially. Note that the false-positives should only come from the training set of images (otherwise you will be training on your test set!).

The `buildTrainingImageDataset` that was used to create the initial set of training images (see Data Preparation) can also be used to extract image windows containing false-positive detections by specifying the `-detections` flag.

```
bin/buildTrainingImageDataset -object <OBJECT> -resize 32 32 -detections <DETECTIONS>.xml \
    -directory data/objects <IMAGE_SEQUENCE>.xml <GT_LABELS>.xml
```

3 Automating the Process

The script trainObjectDetector.pl implements the main steps:

- Building a Patch Dictionary
- Computing the Patch Response Cache
- Training the Boosted Detector
- Trimming the Dictionary

of the above pipeline for training an object detector.

```
svl/scripts/trainObjectDetector.pl -c -w 32 -h 32 -r 100 -s 2 \
    -n 10 -m 2000 -M 20000 -N negative <OBJECT>
```

4 References

- P. Viola and M.J. Jones, "Robust Real-Time Face Detection," IJCV, 2004.
- J. Winn and A. Criminisi and T. Minka, "Object Categorization by Learned Universal Visual Dictionary," ICCV, 2005.
- A. Torralba and K.P. Murphy and W.T. Freeman, "Sharing Visual Features for Multiclass and Multiview Object Detection," PAMI, 2007.
- Dalal, N. and Triggs, B., "Histograms of Oriented Gradients for Human Detection," CVPR, 2005.