Discriminatively trained deformable parts models
Short report

Jorge Almeida

September 27, 2013

1 Introduction

The `voc_release5` is a complete learning-based system for detecting and localizing objects in images. The system represents objects using mixtures of deformable part models. These models are trained using a discriminative method that only requires bounding boxes for the objects in an image. This system achieved state of the art results on the PASCAL dataset.

This document presents an overview of the main functions and their interaction.

The `voc_release5` package is accessible in `http://www.cs.berkeley.edu/~rbg/latent/`

2 Configuration

To prepare Matlab to use the `voc_release5` use the function `startup()`. This function adds all necessary paths.

The `voc_release5` uses `C++ mex` functions to increase performance. The `mex` functions can be compiled by calling the function `compile()`. To successfully compile, additional `CXXOPTIMFLAGS` are needed, these flags are `-msse -msse2 -msse3 -mfpmath=sse`.

Package parameters are configured by `voc_config()`. This function provides a simple overwrite mechanism, the user must only define the global variable `VOC_CONFIG_OVERRIDE` with a handler to an additional user defined configuration file. All parameters already defined in the user configuration file will **not** be redefined.

3 Training

The top level training function is `pascal()`, this function trains and evaluates a model using the PASCAL database. This function calls `pascal_train()` for the training phase.

`pascal_train()` starts by obtaining the configuration using `voc_config()` and loading the training information (headers only) by calling `pascal_data()`.

The `pascal_train()` function performs multiple rounds of training using the `train()` function. First, it starts by training only the root filters, figure 1, the user specifies how many root filters must be trained for each class, the positives are sorted by aspect ratio and n root filters are initialized. Then mirrored versions of the initialized root filters are added to create mixture models, and subsequently trained, figure 2. Following that, all root filters are combined into one mixture model and undergo another round of training. The last step is the inclusion of part filters, 8 parts for each root model, and retraining, figure 3.

Function `root_model()` is used for root model initialization; `lr_root_model()` is used to add mirrored root filters; `model_merge()` combines all root models into a single mixture model and `model_add_parts()` adds parts to a model.

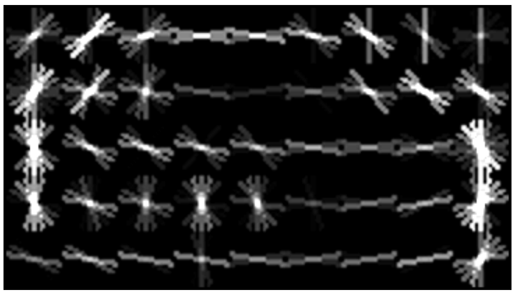

Figure 1: Asymmetrical root filter trained with warped positives and random negatives. This is just one of the root filters created for each class, in this case a car.

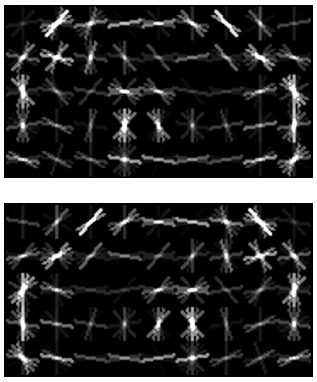

Figure 2: Mixture of two mirrored root filters trained with latent positives and hard negatives. In this stage there are still multiple models created from the initial positives.

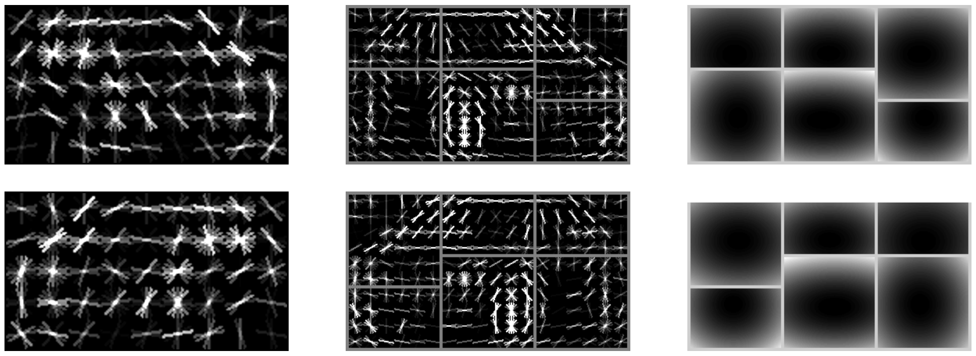

Figure 3: Part of a final model, only one of the mirrored root filters. On the left the root filters; in the middle the part filters and on the right the cost of placing the part filters away from their initial position, relative to the root.

`train()` implements the main training routine, this function uses a `C++` implemented feature cache. The examples are prepared using specialized functions and put in the cache: `poswarp()` prepares warped positive examples; `poslatent()` prepares the latent positives; `neghard()` prepares the hard negatives and finally `negrandom()` prepares the random negatives.

The preparing functions use the `C++` implemented `features()` function to calculate HOG features. Both `poswarp()` and `negrandom()` call `features()` directly. `poslatent()` calls `gdetect_pos_prepare()` to create a feature pyramid (using `featpyramid()`) and prepare it for latent detection, while the `neghard()` calls `featpyramid()` directly.

`train()` uses the `fv_obj_func()` to calculate the objective function value and gradient, this operation is also performed on the cache. L-BFGS is used to minimize the objective function by calling `minConf_TMP()` (function from an external library included in the package).

All cache operations are performed by calling `fv_cache()` with different arguments (see the `handlers` structure in `fv_cache.cc` for a list of possibilities and the corresponding handling functions).

4 Detection

To perform a detection using a previously trained model the function `process()` is used. This function calls `imgdetect()`, that calculates the feature pyramid, `featpyramid()`, and calls `gdetect()`. `gdetect()` performs detections by first computing dynamic programming tables using `gdetect_dp()` and obtaining detections by calling `gdetect_parse()`.

Detections are output as bounding boxes for both the root filter position and parts positions.

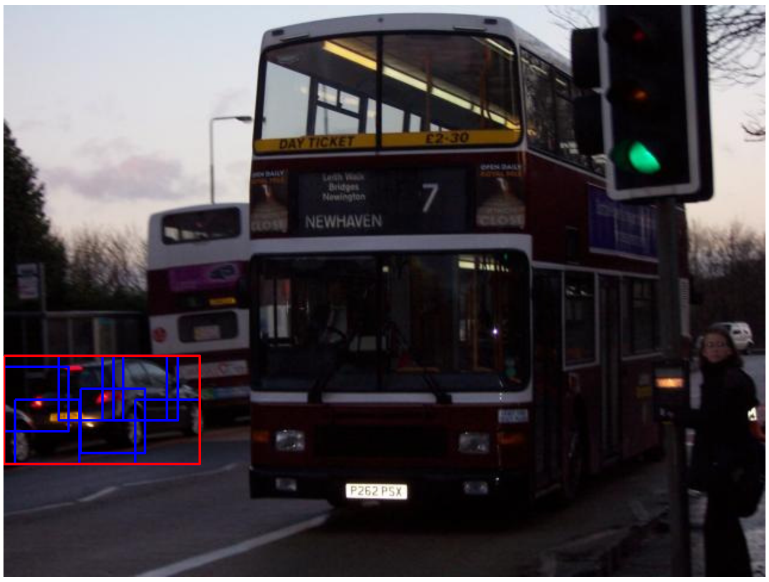

Figure 4: Example of a vehicle detection. The red box denotes the root filter detection while the blue boxes are part filters.