detectorPLS

version 0.1.1

William Robson Schwartz
http://www.umiacs.umd.edu/~schwartz

October 30, 2009

Contents

Chapter 1

Introduction

This manual describes some functionalities of the software **detectorPLS**, which contains an implementation of the paper

> [1] *Human Detection Using Partial Least Squares Analysis.*
> W.R. Schwartz, A. Kembhavi, D. Harwood, L. S. Davis.
> In proceedings of the ICCV. Kyoto, Japan, 2009 [PDF].

if you use this software in your research, please cite the paper above. **Note:** this software cannot be used for commercial purposes. Further information about this research might be found on the author's webpage: `http://www.umiacs.umd.edu/~schwartz/research.html`.

The goal of this implementation is to allow researchers to perform detection using already learned models for applications such as human and face detection and also provide a simple way of learning new object models for detection by providing exemplars of the positive and negative samples.

As described in our paper [1], the detection method is based on the extraction of a rich set of features based on edges, colors and textures analyzed by partial least squares (PLS). The experimental results have shown that this approach outperforms state-of-art methods for human detection in three standard datasets: INRIA Pedestrian Dataset [2], ETHZ Pedestrian Dataset [3], and DaimlerChrysler Dataset [4].

The current implementation provides two modules: *detection* and *learning*. The *detection module* performs object detection in multiple scales where the user is allow to input a single image, a directory containing multiple images specified by a extension, or a video. While the *learning module* allows the user to perform training of new object models by providing directories containing negative and positive exemplars of an object class to be learned.

Note: the user should be aware that the features currently implemented in our system are location dependent, therefore the detector will not work for objects presenting wide intra-class variation such as those in *Caltech-256 Object Category Dataset* [5]. Using the features currently implemented, the system works best when the objects have small intra-class variation such as humans, faces, heads, and cars.

If you have found bugs or problems in this software or you have suggestions to improve or make it more user friendly, please send an e-mail to schwartz@cs.umd.edu.

Chapter 2

Performing Object Detection

The detection module allows the user to provide different inputs such as a single image, a directory containing multiple images, input video, or a streaming from a camera. Then, the detection is performed using a sliding detection window considering multiple scales and using different strides in horizontal and vertical directions, both defined by the user.

Our implementation allows that the user specify multiple objects to be detected simultaneously such that the multiple computation of features might be avoided if the object share the same features. In addition, multiple stages are allowed for detecting an object. In this case, the earlier fast stages reject detection windows according to user-defined thresholds and then the remaining detection windows are considered in late and slower stages, composed of a larger number of features.

The setup of the detection is composed by two parts: a *configuration file* and *command line parameters*.

2.1 Configuration File

The configuration file contains information regarding the objects to be detected such as the file containing the learned model, the thresholds used in each stage, and the strides in the horizontal, vertical and scale that each object should be considered. Figure 2.1 shows an example of a configuration file considering the detection of two objects simultaneously.

The example shown in Figure 2.1 consists of detectors for body and face. The body detector, identified as Body, consists of two stages. In the first stage, a sliding detection window scans the whole image searching for

```
# comment

# body detector
model <Body, 0, 0.2, learnedFullBody64x128FastHOG.ret02.yml>
model <Body, 1, 0.999999, All.64x128Learning.ret03.yml>
scale <Body, 65, 400, 0.15, 0.05, 1.1>

# face detector
model <Face, 0, 0.9999, learnedFaceInternet.ret03.yml>
scale <Face, 30, 140, 0.15, 0.05, 1.1>
```

Figure 2.1: Example of the configuration file used to run body and face detection simultaneously, where the body detection consists of two stages.

humans with height between 65 and 400 pixels. For a given location, if the response is smaller than 0.2, this window is rejected and there is no human of this size at that particular location, otherwise, the second stage is considered and a human is outputted it the response (probability) is higher than 0.999999. On the other hand, the face detector has only one stage.

The parameters for each detector consists of two sections, the first define the object models to be used, having the fields defined as

```
model <object name, stage ID, threshold, model file>
```

where

object name	Unique ID used to identify the object being considered in multiple stages and match to scale parameters.
stage ID	Sequential ID for multiple stages. The ID must start in 0.
threshold	Threshold used to reject samples in a given stage. If the response (probability or regression value) obtained in that stage is smaller than the threshold, they detection window is discarded. In the last stage, only the detection windows with response greater or equal than the threshold will be outputted.
model file	.yml file containing the learned model for an object for a given stage.

the second set of parameters defines the scale and strides in which the objects will be detected, the format is defined as follows

```
scale <object name, min size, max size, delta_X, delta_Y, scale>
```

where

object name	Unique ID used to identify the object being considered in multiple stages and match to scale parameters.
min size	Minimum object height. **Note:** if the minimum height is set to a value smaller than the field *objH* in the .yml model file, the image will need to be increased in size to search for object with that size.
max size	Maximum object height.
delta_X	Delta for the horizontal stride of the sliding detection window. The stride (in pixels) is defined by s * delta_X * objW, where *objW* is defined in the .yml model file and s is the current value of the scale parameter.
delta_Y	Delta for the vertical stride of the sliding detection window. The stride (in pixels) is defined by s * delta_Y * objH, where *objH* is defined in the .yml model file and s is the current value of the scale parameter.
scale	Delta for the scale. To the object go from min to max size, several iterations are performed, where for each iteration, s = s * scale, where starts as 1 and the object size is h = h * scale, where h starts with min size. **Note:** to avoid an infinite loop, scale must have value higher than 1.

2.2 Command Line Parameters

The command line parameters provide information regarding the input and output of the detector. An example is shown below.

```
./detectorPLS -d -c ParamsExecutionNew64x128.txt -i wm2.ppm \
    -o output -s -b
```

In this example, the detection is performed (-d) in a single image file named wm2.ppm (-i) and the configure file used is ParamsExecutionNew64x128.txt (-c). The detection result is saved in the directory output (-o). In addition, a text file is outputted saving the bounding boxes locations (-b) and the non-maximum suppression is performed (-s).

The general command line with its possible parameters has the following format.

```
./detectorPLS -d -c <filename> -o <directory> \
    -i <file/directory> [-k <string>] [-s] [-g] [-b] [-e] \
    [-v <first,last,step>] [-a] [--version] [-h] <x,y,w,h>
```

The following table describe the user options for the detection (the second column indicates if an argument is required (r) or optional (o)).

`-d, --detection`	r	Indicates that detection should be performed.
`-c, --configfile`	r	Specify the configuration file to be used for the detection (see section 2.1 for more details regarding the configuration file).
`-i, --input`	r	Set the input for the detector. The input can be one of the following. Single image, directory containing multiple image files (see parameter -k). A video file (those that can be read by OpenCV 1.0).
`-o, --output`	r	String containing the directory where the results will be store. If the input files are images, a text file will be outputted have the bounding boxes for the detection windows that have response greater than the threshold for the final stage. Each bounding box is displayed in a row in the following format. `x y w h r` (x,y: top-left corner, w: width of the bounding box, h: height of the bounding box, r: response for this detection window (probability or regression response)). See example below.
`-k, --filemask`	o	Mask used to locate multiple image files in the input directory. The default file mask is *.png,*.jpg.

`-s, --nonMaxSup`	o	Perform non-maximum suppression to select only the detection windows that the response is maximum.
`-g, --stageNonMaxSup`	o	Option not yet implemented.
`-b, --drawBB`	o	Saves image files overlayed with rectangle bounding boxes around the object. If the input is a file, the output will have the same name. If the input is a video, the output will consist of files with the same name of the video appended by the frame number. If flag -a is set and the input is from a camera, the output filename will be VideoFrame with the frame number appended. **Note:** all outputs will be .png files.
`-e, --drawEllipse`	o	Instead of using rectangle bounding boxes, uses ellipses
`-v, --videoInfo`	o	Parameter to specify the first, last frames of a video and also define how many frames should be skip. The format is first,last,skip
`-a, --camera`	o	This flag specify that the input is read from a stream provided by a camera attached to the computer.
ROIs	o	When a single image is set as input, multiple regions of interest can be passed as parameter. For each ROI one need to specify x,y of the top-left position and width and height of the ROI. The format is $x_1, y_1, w_1, h_1 \; x_2, y_2, w_2, h_2 \; \ldots \; x_k, y_k, w_k, h_k$

Example of the output text file:

```
163 203 23 66 0.9995995759964
457 203 25 72 1
300 218 34 97 0.9999936819077
221 182 36 104 0.999884724617
```

Chapter 3

Learning New Object Models

This implementation provides routines to learn new object models by presenting positive and negative samples of an object class. The learning procedure is flexible such that the user can easily change the features, block sizes, and number of features.

3.1 Configuration File

The configuration file is used to setup how the new object model should be build. It specifies the features, their parameters, detection window size, object size, number of factors, and block size and strides. After the object model is learned, its parameters are saved in a .yml format file. Figure 3.1 shows an example of a configuration file.

In this example, the features considered are HOG, color frequency and co-occurrence matrix for color bands H, S, and V (`COOCH`, `COOCS`, `COOCV`). The features computed using co-occurrence matrix are been cached (`params <COOCPARAM, cache, 1>`). The block sizes considered for both HOG and co-occurrence matrix are 16×16 and 32×32 pixels with stride of 4 and 8, respectively. The classifier used is QDA with 20 factors. The detection window is 42×48 and the object size is 30×34, centered inside the detection window.

The feature extraction methods to be used are defined as following.

```
method <method name, method ID, cache, parameters ID>
```

```
# internal identifier (ignore this)
ModelID <FaceInternet42x48AllFeat>

# feature methods used
method <HOG,HOG,0,HOGPARAM>
method <COOCH,COOC,1,COOCPARAM>
method <COOCS,COOC,1,COOCPARAM>
method <COOCV,COOC,1,COOCPARAM>

# feature parameters
params <COOCPARAM,bins,16>
params <COOCPARAM,distance,1>
params <COOCPARAM,cache,1>

params <HOGPARAM,UseColorFrequency,1>
params <HOGPARAM,UseHOG,1>
params <HOGPARAM,cache,0>

# block sizes and strides for each feature
block <HOG,16,16,4,4,5>
block <HOG,32,32,8,8,5>

block <COOC,16,16,4,4,5>
block <COOC,32,32,8,8,5>

# classifier to be used and number of factors
classifier <qda>
factors <20>

# detection window and object size
window <42,48>
object <5,6,30,34>

# region inside the detection window to be considered
region <0,0,41,47>
```

Figure 3.1: Example of the configuration file used to learn an object model considering HOG, co-occurrence matrix, and color frequency.

where

method name	Name of the feature extraction method. These names are defined in the implementation. Currently available methods are COOCH, COOCS, COOCV (co-occurrence matrix for color bands H, S, and V) and HOG.
method ID	ID used to link this method to the block setup (see `blocks`).
cache	Obsolete parameter.
parameters ID	ID used to setup the parameters for this method (see `params`).

Each feature extraction method has a few parameters that can be set by the user. They have the following format.

```
params <parameters ID, name, value>
```

where

parameters ID	identifier to link to the feature extraction method (same as in `method`).
name	parameter name.
value	value for the current parameter.

The parameters available for co-occurrence matrix are: bins, distance, cache. Please, do not change the first two parameters. If the value of cache is 1, the features extracted will be cached, avoiding multiple computations of features for a given block.

The parameters available for HOG method are: UseColorFrequency, UseHOG, cache. If UseColorFrequency has value 1, the color frequency feature will be computed. If UseHOG has value 1, HOG will be compute (**Note:** if only UseColorFrequency has value 1, set parameter -f (section 3.2) to at most 3). If cache is set to 1, the HOG features are cached (**Note:** since HOG uses integral histograms, the cache does not save much computational time, but, in fact, it only increases the memory consumption).

The block setup is done by the following command.

```
block <method ID, width, height, stride_X, stride_Y, factors>
```

where

method ID	identifier to link to the feature extraction method (same as in `method`).
width	width of the block.
height	height of the block.
stride_X	stride of the block in the horizontal direction.
stride_Y	stride of the block in the vertical direction.
factors	obsolete parameter.

The classifier to be considered for the PLS subspace is defined by the following command. Currently, only the QDA classifier is considered (string to be used in the name is qda).

```
classifier <name>
```

The number of factors (dimensionality of the PLS subspace) to be used by the classifier is defined by the following command. This command will not be considered if the cross-validation is performed (parameter -C described in section 3.2).

```
factors <value>
```

The following command defines the size of the detection window to be used to detect the object.

```
window <width, height>
```

Usually, it is important that the detection window size be greater than the object size because some information from the background can be used to improve the detection performance. This way, the object size is defined in the next command, where x,y indicates the location of the top-left corner of the object inside the detection window. width and height indicate the size of the object.

```
object <x, y, width, height>
```

Sometimes one is interested on extracting features from sub-regions of the detection window (used for part-based detectors). The following command allows the user to specify which those regions are. Multiple entries of this command can be set to covering different regions of the detection window.

```
region <x, y, width, height>
```

3.2 Command Line Parameters

The general command line with its possible parameters has the following format.

```
./detectorPLS -t -c <filename> -n <directory> \
     -p <directory> [-k <string>] [-m <filename>] \
     [-z <number>] [-w] [-f <number>] [-S <number>] \
     [-T <number>] [-C] [-F <number>] [-B <number>] \
     [-R <number>] [-I <number>] [-M <number>] \
     [--version] [-h]
```

The description of each parameter is given in the following table. In the second column, (r) indicates that the argument is required and (o) indicates optional.

`-t, --training`	r	Indicates that a new model will be learned (called training).
`-c, --configfile`	r	Specify the configuration file used to learn a new model (see section 3.1 for more details regarding the configuration file used during the training).
`-n, --negative`	r	Specify the directory containing the negative exemplars of the object (images that do not contain the object). Set parameter -k to define the file mask. This directory might contain images of any size. A sliding window with size of the detection window (see section 3.1) and stride defined by parameter -S will scan the image to sample negative exemplars.

`-p, --positive`	r	Specify the directory containing the positive exemplars of the object (images containing samples of the object). Set parameter -k to define the file mask. The images in this directory must have the detection window size define in the configuration file (see section 3.1), with only one object sample per image.
`-k, --filemask`	o	Mask used to locate multiple image files in the negative and positive directories. The default file mask is *.png,*.jpg.
`-m, --modelFile`	o	Specify the filename where the model learned will be saved (default: learnedModel).
`-z, --seed`	o	Select initial seed for the pseudo-random generator (used to choose a subset of negative samples randomly). If set to -1, srand(time(NULL)) is used to initialize pseudo-random generator. (default: 0).
`-w, --rawFeatures`	o	Instead of build PLS models for each block then concatenate the resulting latent variables, the raw features are used to learn the PLS model (requires much more memory and the quality of the detector might not improve significantly).
`-f, --nfactBlock`	o	Specify the number of latent variables used for each block when raw features are not used (default: 5).
`-S, --shift`	o	Specify the stride (the same on the horizontal and vertical) to slide a detection window to sample negative exemplars (default: 16).
`-T, --ThRetraining`	o	Specify the threshold used to select negative samples for the retraining (default: 0.10).
`-C, --cv`	o	Run cross-validation to find the best number of PLS factors (weight vectors) according to the training set.
`-F, --nfolds`	o	Specify the number of folds used in the cross-validation (default: 20).

`-B, --nblockIt`	o	Specify the number of blocks to be considered to extract features. When raw features are not used, this parameter indicates the number of blocks that will have features extracted before building local PLS models. A small number allow machines with little memory to perform the training, a large number allow the training to be faster, but it will require more memory (default: 330).
`-R, --samplesIt`	o	Specify the number of negative samples that might be added per retraining iteration (default: 1000).
`-I, --nRetIt`	o	Specify the maximum number of rounds of the retraining (default: 4).
`-M, --nSampIm`	o	Specify the number of exemplars sampled from each negative image (default: 1).

3.2.1 Memory Consumption

The user should be aware that if raw features are used (parameter -w), the memory consumption for the training will be high because features from all block must be in memory to build the PLS model. Therefore, if your computer does not have much memory (for example: 1 or 2GB) and the number of training samples is large, you should not use raw features.

If you do not use raw features, you can control the memory consumption by setting the number of blocks per iteration (parameter -B) to a small value. In this case only features for that number of blocks must be in memory at once. In addition, the number of latent variables (parameter -f) may also be used to reduce the memory consumption, but it is not recommended because the detection quality may be reduced.

Bibliography

[1] W. R. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis, "Human detection using partial least squares analysis," in *In International Conference on Computer Vision*, 2009.

[2] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *CVPR 2005*, 2005, pp. 886–893.

[3] A. Ess, B. Leibe, and L. V. Gool, "Depth and appearance for mobile scene analysis," in *ICCV*, October 2007.

[4] S. Munder and D. Gavrila, "An experimental study on pedestrian classification," *PAMI*, vol. 28, no. 11, pp. 1863–1868, 2006.

[5] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," California Institute of Technology, Tech. Rep. 7694, 2007. [Online]. Available: http://authors.library.caltech.edu/7694