HOG-based Pedestriant Detector Training

eVS embedded Vision Systems Srl

c/o Computer Science Park, Strada Le Grazie, 15 Verona- Italy

http://www.embeddedvisionsystems.it

Abstract

This paper describes how to train a pedestrian detection classifier based on HOG and how to use the training results to confgure the logiPDET IP core. logiPDET is HOG/SVM-based pedestrian detector optimized for Xilinx®devices. The core is developed for real-time vision-based embedded applications such as Automotive Driving Assistance. The algorithm follows a discriminative approach combining a HOG descriptor and a SVM classifier. logiPDET is a configurable detector suitable for different classes of objects. Pedestrian is just the most popular example. The training process presented in this paper is valid in general, for different objects, and not only to train the logiPDET IP core. An example of C++ source code for training the pedestrian detector is also provided.

The IP core is sourced from eVS embedded Vision Systems Srl and is part of the logicBRICKS™ IP core library provided by Xylon doo. It can be evaluated on the logiADAK Automotive Driving Assistance Kit . It is a Xilinx Zynq-7000 All Programmable SoC based development platform for advanced vision-based automotive Driver Assistance (DA) applications. For more information about the logiPDET and logiADAK visit `http://www.logicbricks.com/`.

Keywords: logiPDET, logiADAK, Pedestrian Detection, HOG, Histogram of Oriented Gradient, SVM, Support Vector Machines, Machine Learning, Driving Assistance, FPGA IP core, Zynq-7000 All Programmable SoC

1. Elements of an HOG pedestrian detector

The approach to pedestrian detection we are considering in this paper has three main components.

A feature descriptor. We describe an image region with a high-dimensional descriptor: in this case, the descriptor is the histogram of gradients (HOG) feature [1]. The feature is based on evaluating normalized local histograms of image gradient orientations on a dense grid (see Figure 1). The image window is divided into small spatial regions (cells). A local 1D histogram of gradient orientations over the pixels of each cell is estimated. Grouping neighboring cells into larger spatial regions, called blocks, and performing local contrast normalization on each block allows to gain a certain invariance to illumination and shadowing. The combination of all the block normalized histograms forms the descriptor.

A learning method We learn to classify an image region (described using HOG features) as a pedestrian or not. For this, we will be using support vector machines (SVMs) and a large training dataset of image regions containing pedestrians (positive examples) or not containing pedestrians (negative examples).

A sliding window detector Using the classifier, we can tell if an image region looks like a pedestrian or not. The final step is to run this classifier as a sliding window detector on an input image in order to detect all instances of pedestrians in that image. The process is repeated on successively scaled copies of the image so that objects can be detected at various sizes and ranges. Usually, non-maximal neighborhood suppression is applied to the output to remove multiple detections of the same object.

2. Pedestrian dataset

The first step is creating the dataset, which consists of a set of images of positive and negative examples.

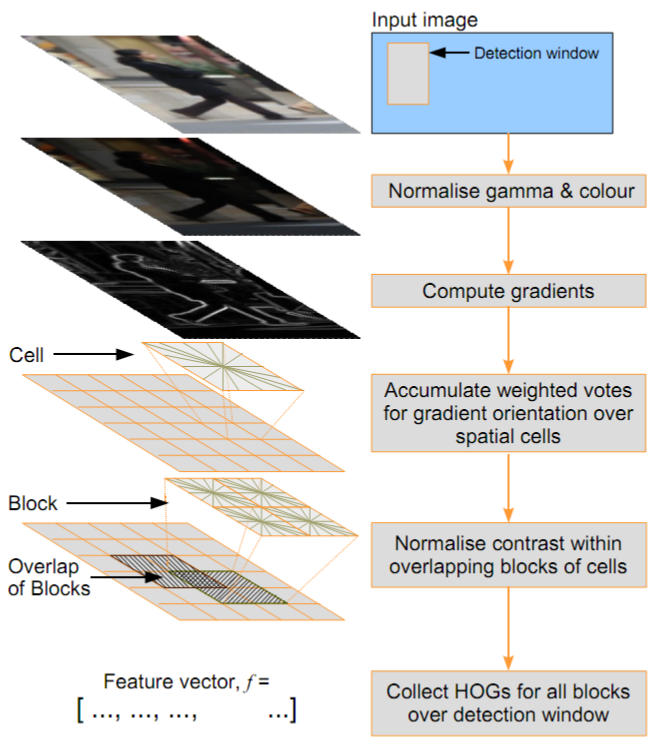

Figure 1: HOG descriptor

2.1. Positive images

Positive images are those containing pedestrians. For a successful training, positives must well represent the complexity of the object. In this case, we need images of pedestrians in different contexts and illumination conditions, with different appearances, poses and clothes. Partial occlusions may also be included. In addition, adding left-right reflections of the selected images is an easy way to enrich the dataset.

On the other hand, there is no need to complicate training if not necessary. Therefore, since independence from scale is achieved scaling the input image, it is best if pedestrians inside positive images appear to be the same size and centered.

You may consider adding images from different points of view, such as top and side views. However, if the appearance of the object varies too much the final classifier may behave poorly. If you run in a case like that, please do consider to train a classifier for each point of view. For example, cameras mounted on the top of trucks will have a different perspective of pedestrians than cameras mounted on passenger cars. Therefore, a special training for truck mounted cameras may be in order.

A very good example of dataset for pedestrians is the INRIA person dataset, which can be downloaded at the website `http://pascal.inrialpes.fr/data/human/`. It is composed by 1239 images of pedestrians, together with their left-right reflections (2478 images total), cropped from a varied set of photos. The people are usually standing, but appear in any orientation and against a wide variety of backgrounds, including crowds. Some positives images of the dataset are shown in Figure 1. These images have 64128 pixels resolution, and include about 16 pixels of margin around the person on all four sides. As explained by the authors of HOG paper [1], this border provides a significant amount of context that helps detection.

Figure 2: Some positive images from INRIA person dataset

2.2. Negative images

Negative images are those not containing pedestrians. Theoretically, they should contain anything but pedestrians, since the classifier must be able to distinguish pedestrians from any other objects. Of course, this set of images is potentially infinite. In practice, we can restrict to contexts where pedestrians are likely to appear, considering the final application. If pedestrian detection is applied in the automotive, for example, a set of images taken from road scenarios are good candidates.

The set of negatives of the INRIA person dataset is composed of 1218 photos of different scenes, urban and not urban, and of generic objects (wheels, flowers, etc. see Figure 3). A set of negatives is compulsory to generate a pedestrian detector that works in any context. To improve performances, it is often necessary to augment the set of negatives of the INRIA person dataset with images from the final application.

Figure 3: Some negative images from INRIA person dataset

3. Pedestrian detection training process

The training process involves a series of steps.

First, a preliminary classifier is estimated using all positive images and a fixed set of patches sampled randomly from the person-free training photos in the dataset. The HOG descriptors of the samples are projected as points in a high dimensional space, and then a support vector machine constructs a hyper-plane in this space, where positive and negative samples are separated. A good separation is achieved by the hyper-plane that has the largest distance from the nearest training data point of each class, so-called functional margin (see Figure 4).

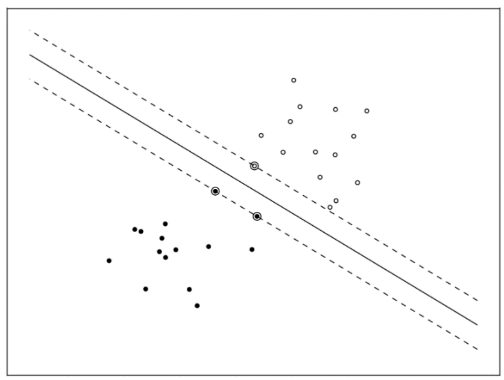

Figure 4: Example of linear Support Vector Machine. White dots represent samples from positives and black dots samples from negatives.The dashed line indicates the maximized margin.

Second, with this preliminary classifier, the complete set of negatives in the training dataset is exhaustively searched for false positives (called hard examples). The classifier is then re-trained using this augmented set (initial number + hard examples) to produce a more accurate detector. The operation of selection of hard examples and retraining is called *bootstrap*.

According to [2], less than two bootstrap rounds would lead to performances that depend heavily on the initial training set. Moreover, at least two bootstrap rounds are necessary to reach the full performance of the standard combination HOG + linear SVM.

4. Pedestrian detection training code

To help in training logiPDET and perform the processes described above, we make available a software bundle downloadable at [enter website]. The code is released as a Qt project. For installation instructions, see the README file included into the bundle. The bundle contains:

1. A revised version of HOG feature extractor function, released as a dll library, that includes all the approximations used in the logiPDET core. It's necessary to be consistent with the FPGA implementation for both training and testing.
2. A Qt project including the source code for SVM training and testing (*trainHog* utility).
3. Example batch files (running under MS Window OS) showing how to parametrize the training tool.

The project is developed using the ver 6.1 of *SVM light* (`http://pascal.inrialpes.fr/soft/olt`). It is a variant of the original *SVM light* by Thorsten Joachims: it adds the capability of handling binary files instead of text files, with a consequent relevant speed up of the performance.

Please note that, to operate correctly, the bundled code requires the following libraries installed on your PC:

- *OpenCV 2.4.10* for HOG descriptor
- *Qt library* version greater than 4

OpenCV is downloadable at `http://opencv.org/downloads.html` while *Qt library* is downloadable at `https://www.qt.io/download/`.

The compilation of the project generates the executable program *trainHog.exe* providing a command line user interface. The utility can be used for training, testing and measuring the performance of the trained classifier. It can be configured with several parameters that controls the training process. The most important ones are reported below:

Number of positive examples. It is always best to use all positives available.

Number of initial negative examples. This is the number of negatives of the initial classifier. Usually we take 5 times the number of positives.

Number of hard negatives. This is the maximum number of samples that the classifier estimates incorrectly, to be extracted from the negative set. These samples will be inserted in the dataset for re-training. You should see that moving forward with bootstrap rounds, it will take much longer to find the required number of hard examples. If, after two bootstraps, a relevant number of false positives is still extracted for each image it means that the training is not working.

Number of bootstrap runs. At least two bootstrap rounds are necessary to get a classifier [2]. Moreover, we experienced that more than three bootstrap rounds usually do not improve performances. Thus we recommend to set this parameter to three runs.

Regularization parameter (C). A soft-margin SVM is used: we allow the classifier decision to make a few mistakes (some samples - outliers or noisy examples - are inside or on the wrong side of the margin). This regularization parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C (>1), the optimization will pick a smaller-margin hyper-plane if that hyper-plane gets all the training points classified correctly. Conversely, a very small value of C (< 0.001) will cause the optimizer to look for a larger-margin separating hyper-plane, even though that hyper-plane misclassifies more points. Since the dataset is usually very big, the default value we recommend is 0.01.

Cost factor. Describes how much training errors on positive examples outweigh errors on negative examples. This parameter is used to compensate the fact that the number of negatives is much bigger than the number of positives. A value of 1 means that positives and negatives have the same weight, a value of m means that each positive weights m-times more than the negatives. Usually it is best to raise its value as the number of negatives increases.

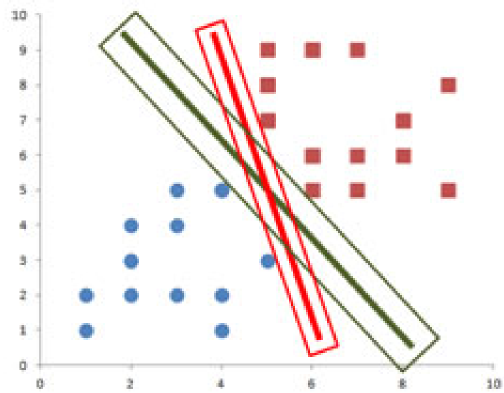

Figure 5: Different hyper-planes are picked for the same training datatset depending on the C value

The result of the training is a text file containing all the coefficients (C0,..,Cn,) of the hyper-plane, plus the bias term. The detection window size of logiPDET is 128x64 pixels corresponding to 15x7 blocks. So, n = 15x7x32 = 3360.

For the logiADAK (ver 2.0 or 3.0) users it is pretty easy to configure the logiPDET IP core with their own trained classifier. They have just to rename the generated text file in SVM.txt and copy it into the /logiPDET folder of the SD card. Switching on the board the pedestrian detection demo application will automatically initialize the logiPDET core with the user's classifier loaded from file.

To help the user in parameterizing the *trainHog* we provide a set of batches files as example for training and testing the classifier:

train.bat Example of how to use the *trainHog* utility in training mode.

testDisplay.bat Example of how to use the *trainHog* utility in test mode. The output is the set of input images with the result of detection in overlay.

testPrecisionRecall.bat Example of how to use the *trainHog* utility to compute the precision and recall metrics (described later).

5. Pedestrian detector performances estimation

To evaluate the performances of a detector one requires ground truth data, i.e. a set of images or videos labeled with pedestrian information. The evaluation works in terms of precision and recall, which are explained in the following paragraph. For classification tasks, the terms true positives (tp), true negatives (tn), false positives (fp), and false negatives (fn) compare the results of the classifier under test with the ground truth. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the ground truth (sometimes known as the observation). Precision and recall are then defined as:

- **Precision** $= tp/(tp + fp)$
- **Recall** $= tp/(tp + fn)$

In simple terms, high recall means that the classifier returned most of the relevant results, while high precision means that the classifier returned substantially more relevant results than irrelevant. The best classifier is the one that maximizes both recall and precision.

6. Pedestrian detector: how to improve performances

In this paper, we provided an example code for training a pedestrian detector based on HOG features and linear SVM. The performances of the resulting classifier may not fit the final application. Is it possible to improve performance? The answer is yes.

Here are the items you may want to try:

- Add positive and negative examples from the scenario of your application.

- Test the classifier on the target scenario and select positive and negative examples that the classifier is not able to classify correctly (hard examples), then add these examples for re-training.

- Modify the parameters of the training process.

- Use all constraints you may have on the scene, or due to the setup. For example, if the camera is mounted fixed on a vehicle you can add region of interests, i.e. test the classifier only in the regions of the image where pedestrians are likely to be present. In fact, due to perspective, pedestrians far away from the camera can only appear on a restrict area.

After some trials, you may find that the classifier does not improve any longer, or it is actually getting worse. This probably means that you reached the maximum performance of the classifier.

References

[1] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: In CVPR, 2005, pp. 886–893.

[2] S. Walk, N. Majer, K. Schindler, B. Schiele, New features and insights for pedestrian detection., in: CVPR, IEEE, 2010, pp. 1030–1037.