Pedestrian detection at 100 frames per second

Rodrigo Benenson, Markus Mathias, Radu Timofte and Luc Van Gool
ESAT-PSI-VISICS/IBBT, Katholieke Universiteit Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract

We present a new pedestrian detector that improves both in speed and quality over state-of-the-art. By efficiently handling different scales and transferring computation from test time to training time, detection speed is improved. When processing monocular images, our system provides high quality detections at 50 *fps.*

We also propose a new method for exploiting geometric context extracted from stereo images. On a single CPU+GPU desktop machine, we reach 135 *fps, when processing street scenes, from rectified input to detections output.*

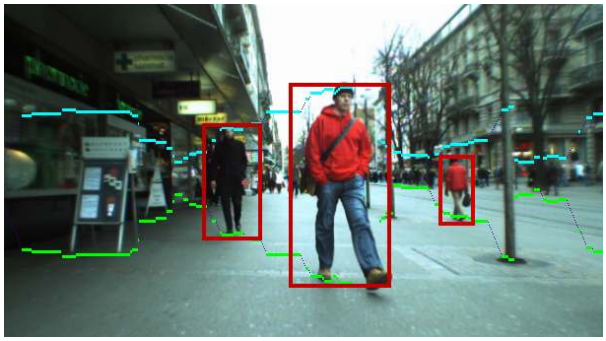


Figure 1: Example result on the Bahnhof sequence. Green line indicates the stixels bottom, blue line the stixels top and the red boxes are the obtained detections.

1. Introduction

Visual object detection is under constant pressure to increase both its quality and speed. Such progress allows for new applications. A higher speed enables its inclusion into larger systems with extensive subsequent processing (e.g. as an initialization for segmentation or tracking), and its deployment in computationally constrained scenarios (e.g. embedded system, large-scale data processing).

In this paper we focus on improving the speed of pedestrian (walking persons) detection, while providing state-of-the-art detection quality. We present two new algorithmic speed-ups, one based on better handling of scales (on monocular images), and one based on better exploiting the depth information (on stereo images). Altogether we obtain speed-ups by a factor ~ 20, without suffering a loss in detection quality. To the best of our knowledge, this is the first time that pedestrian detections at 100 fps (frames per second) has been reached with such high detection quality.

1.1. Related work

Providing an exhaustive overview of previous, fast object detection work is beyond the scope of this paper. Yet, most of the work on improving detection speed (without trading-off quality) exploits one or more of the following ideas:

Better features Having cheap to compute features that capture best the input image information is crucial for fast and good detections.
Viola and Jones popularized the use of integral images to quickly compute rectangular averages [18]. Later, Dalal and Triggs popularized the idea that gradient orientation bins capture relevant information for detections [4]. In the same vein, bag-of-words over dense SIFT features has been used [12].
It has also been shown multiple times that exploiting depth and motion cues further improves the detection quality [5, 2, 11], but so far usually at the cost and not the benefit of speed.

Better classifier For a given set of features, the choice of classifier has a substantial impact on the resulting speed and quality, often requiring a trade-off between these two. Non-linear classifiers (e.g. RBF SVMs) provide the best quality, but suffer from low speed. As a result, linear classifiers such as Adaboost, linear SVMs, or Random/Hough Forests are more commonly used. Recent work on the linear approximation of non-linear kernels seems a promising direction [16].

Better prior knowledge In general, image processing greatly benefits from prior knowledge . For pedestrian detection the presence of a single dominant ground

plane has often been used as prior knowledge to improve both speed and quality [9, 14, 17, 3].

Cascades A frequently used method for speeding up classifiers is to split them up into a sequence of simpler classifiers. By having the first stages prune most of the false positives, the average computation time is significantly reduced [18, 20, 10].

Branch and bound Instead of accelerating the evaluation of individual hypotheses, branch-and-bound strategies attempt to reduce their number [12, 13]. By prioritizing the most promising options, and discarding non-promising ones, the speed is increased without a significant quality loss.

Coarse-to-fine search is another popular method to arrive at fast object detection [15]. It can be considered a specific case of cascades, where the first stages take decisions on the basis of a coarser resolution. Coarse-to-fine approaches trade-off quality for speed. If the coarse resolution were good enough, then all computations could be done at this level, otherwise we are guaranteed to incur a quality loss.

1.2. Contributions

Our main result is a novel object detector, which we here demonstrate to provide high quality pedestrian detection at 135 fps. This detector is 20 times faster than previous results of equal quality [6, 8], and has only half as many false positives on the INRIA dataset compared to previous results obtained at equal speed [17]. This result is based on two core contributions.

Object detection without image resizing Common detection methods require resizing the input image and computing features multiple times. Viola and Jones [18] showed the benefits of "scaling the features not the images", however such approach cannot be applied directly to HOG-like feature because of blurring effects. To the best of our knowledge, we present the first detector based on orientation gradients that requires no image resizing (see section 3). This can be seen as a "better classifier".
This improvement provides a $\sim$ 3.5 times algorithmic speed-up on the features computation stage. We show state-of-the-art (monocular) detections running at 50 Hz on GPU; this is 10 times faster than previous `HOG+SVM` GPU results and two times better quality-wise (see section 6).

Object detection using stixels Depth information is known to be a strong cue for detections. However depth maps are too slow to compute. For the first time, we show that a recently introduced fast depth information method ("stixels world", see figure 1) [3] can be used to accelerate objects detection in practice (see section 5).

Using the stixel world ("better prior knowledge"), the detection search space is reduced by a factor 44 (factor 5 with respect to using ground plane constraints only), enabling a significant speed-up in practice. We obtain (stereo-based) detections at 135 Hz without quality loss, while running on a single CPU+GPU desktop machine. The same algorithm runs at 80 Hz on a high-end laptop (see section 6).

In the next section, we succinctly describe our baseline detector and proceed to describe our contributions in sections 3 and 5. Section 6 discusses our experiments, and spells out the effects on the speed and quality of pedestrian detection that the different novelties bring. Section 7 concludes the paper with some general considerations.

2. Baseline detector

Our baseline detector is `ChnFtrs` of Dollar et al. [7]. Based on the exhaustive evaluations presented in [8], this detector yields state-of-the-art results, on par with the popular part-based detectors [10]. Since then, we are aware of only two quality improvements over this baseline. The release of `LatSvmV4` [10] seems to present significant improvements over `LatSvmV2`. Also, Park et al. [14] presented a multi-scales detector that evaluates multiple detectors (including part-based models) at multiple scales to improve results. We are comfortable claiming that `ChnFtrs` provides state-of-the-art results for single part templates (significantly outperforming `HOG+SVM` [4]), and is competitive with the initial versions of part-based detectors. Our own detector, will improve on top of `ChnFtrs`.

The `ChnFtrs` detector is based on the idea of "Integral Channel Features", which are simple rectangular features that sum a filter response over a given image area. For pedestrian detection it was shown that using 6 quantized orientations, 1 gradient magnitude and 3 LUV color channels was enough to obtain state-of-the-art results (see figure 5, upper row). On top of these rectangular features a set of level-two decision trees (three stump classifiers per tree) are constructed and then linearly weighted to obtain a strong classifier. The set of decision trees and their weights is learned via discrete Adaboost.

Unless specified otherwise, we use the same setup as used for the best results in the original paper [7]. The strong classifier consists of 2000 weak classifiers, the features are selected from a random pool of 30000 rectangles. The training starts with a set of 5000 random negative samples and then bootstraps twice, each time adding 5000 additional hard negative sample. The classifier is trained and evaluated using the INRIA pedestrians dataset. For faster training and evaluation we also shrink the features by a factor 4 (after computing the feature responses, before creating the integral image), as described in [7, addendum].

The results presented here correspond to a complete re-implementation of the original method. Details of the fea-

ture computation and the bootstrapping procedure came out to have a significant impact on the final results. An all too naive implementation may lead up to about $10\ \%$ in performance loss when compared to the reported results.

We produced two implementations of the `ChnFtrs` detector, a CPU version and a compatible GPU version. Evaluated over 640×480 pixels images (evaluating all shrunk pixels, all 55 scales, and without using a cascade), the CPU version runs at about 0.08 Hz when running on an 8 cores machine Intel Core i7 870; the GPU version runs roughly 15 times faster, at 1.38 Hz on an Nvidia GeForce GTX 470. At test time the GPU code spends roughly half of the time resizing the images and computing the integral channels, and half of the time computing the feature responses and detection scores.

Another relevant feature of this detector is that the training is fairly fast. In our implementation the full training from raw training dataset to final classifier (including the bootstrapping stages) takes about three hours, on a single CPU + GPU machine. Importantly, the training time and memory consumption is stable even if the learned model has larger dimensions; this is a enabler for the approach described in section 3.

Comparison with `HOG+SVM` At a first glance it may be surprising to see that such a simple classifier may be able to compete with sophisticated approaches such as HOG part-based models [10]. A key difference is the use of learned features versus hand designed features. Whereas Dalal and Triggs [4] chose to place the HOG cells uniformly, the `ChnFtrs` detector instead learns where to place the features so as to maximize its discriminating power.

3. Improving multi-scale handling

Ideally, a class-specific object detector yields the correct number of object instances, as well as their positions and scales. A naive approach would create a classifier for each position and scale, and make them compete against each other. The strongest responses would then be selected. Since responses overlap, some non-maximum suppression should determine the number of instances. This is an abstract description of the most commonly used object detector architecture (sliding-windows type).

Due to the pixel discretization, the object appearance at different scales changes by more than just scaling. In small scales objects appear "blurry", while bigger scales provide more detailed images. Assuming that object appearance is invariant to translations in the image, to implement the naive approach we should train as many models as there are scales, see figure 2a. The number of scales N is usually in the order of ~ 50 scales.

Training 50 models seems like a daunting task. The traditional approach for object detection at multiple scales (used by [7, 10]), is to train a single model for one canonical scale, and then rescale the image N times, see figure 2b. A detection with the canonical model scale on a resized image becomes equivalent to a detection on a different scale.

This traditional approach has been shown to be effective, but nonetheless it poses two problems. First, training a canonical scale is delicate, as one needs to find the optimal size and learn a model that will trade-off between the rich high resolution scales and the blurry low resolution scales. Secondly, at run-time one needs to resize the input image 50 times, and recompute the image features 50 times too. In the rest of the section we will explain how to sidestep these issues, without having to naively train N models.

3.1. Approximating nearby scales

Recently, Dollar et al. [6] proposed a new approach for fast pedestrian detections, named `FPDW`. Instead of rescaling the input image N times, they propose to rescale it only N/K times, see figure 2c. Each rescaled image is used to compute the image features, and these image features are then in turn used to approximate the feature response in the remaining $N - N/K$ scales. By reducing the number of image resizing and feature computations by a factor K (~ 10), the total detection time is significantly reduced.

The core insight of the `FPDW` approach is that the feature responses of nearby scales can be approximated accurately enough (up to half an octave). This empirical approximation can be described as follows (see [6] for details),

$$r(s) = \begin{cases} a_u \cdot s^{b_u} & \text{if } s > 1 \\ a_d \cdot s^{b_d} & \text{otherwise} \end{cases} \quad (1)$$

where s is the scaling factor (the new height of the detection window is equal to the old height times s), $r(s)$ is the ratio between a feature response at scale 1 versus scale s, and a_u, b_u, a_d, b_d are parameters empirically estimated for the up-scaling and down-scaling case.

In our implementation we use $a_u = 1$, $b_u = 0$, $a_d = 0.89$, $b_d = 1.586$, for the orientation channels, and $a_u = a_d = 1$, $b_u = b_d = 2$ for the LUV color channels; following the empirical evaluation from [6].

3.2. Object detection without image resizing

The core idea of our paper it to move the resizing of the image from test time to training time. To do so we will use the insight of the `FPDW` detector and *reverse it*. Since we can approximate the feature responses across scales, we can decide how to adjust a given stump classifier to classify correctly, as if the feature response had been computed at a different scale.

The strong classifier is built from a set of decision trees, with each decision tree containing three stump classifiers. Each stump classifier is defined by a channel index, a rectangle over such a channel, and a decision threshold τ. When

N models, 1 image scale
(a) Naive approach

1 model, N image scales
(b) Traditional approach

1 model, N/K image scales
(c) FPDW approach

N/K models, 1 image scale
(d) Our approach

Figure 2: Different approaches to detecting pedestrians at multiple scales.

rescaling a stump by a relative scale factor s, we keep the channel index constant, scale the rectangle by s and update the threshold as $\tau' = \tau \cdot r(s)$.

We can now take a canonical classifier, and convert it into K classifiers for slightly different scales. Based on this, we then proceed to train N/K (~ 5) classifiers, one for each octave (scale 0.5, 1, 2, etc.), see figures 2d and 5. Given that training our baseline detector takes three hours beginning to end in a single desktop computer, we can easily train our five classifiers in a few hours.

At test time we use the described approximation to transform our N/K classifiers into N classifiers (one per scale), we compute the integral channel features on the original input image, and then compute the response for each scale using the N classifiers. The proposed approach effectively enables to use the naive approach initially described (figure 2a). We name our "no image rescaling approach" the `VeryFast` detector.

Algorithmic speed-up Being able to skip the effort of computing multiple times the features, is clearly interesting speed-wise. Assuming half of the time is spent computing features for 50 scales and half of the time evaluating classifier responses, computing features only once would provide at best a speed-up of 1.9 times. Compared to `FPDW`, assuming canonical scales 0.5, 1, 2 and 4; avoiding image resizing reduces by a factor 3.75 the features computation time. Then using `VeryFast` instead of `FPDW`, provides theoretical $1.57\times$ speed-up.

Measured speed-up In our GPU code the `VeryFast` method is twice as fast as using the `ChnFtrs` detector (2.68 Hz versus 1.38 Hz), while at the same time showing a slight quality improvement (results presented in section 6). As expected, our `VeryFast` detector is also faster than `FPDW` (2.68 Hz versus 1.55 Hz).

After modifying the handling of scales, the GPU code now spends only 5% of the time computing features and the remaining 95% is solely dedicated to computing the features responses at different scales and positions. Even more, now the code does not need anymore to alternate between computing detection scores and computing the features; having a more streamlined execution path has significant impact in practice. This creates ideal conditions to further speed-up our detector, as described in section 4.

3.3. Training the multi-scale classifier

To train the N/K classifiers we rescale the positive training images to fit the desired object size. All large pedestrians can be converted into small training examples, however when rescaling small examples into large sizes blurring artefacts appear. The INRIA Persons dataset contains only few examples (< 600) of pedestrians taller than 256 pixels, so training the larger scales using only appropriate example sizes risks leading to poor classifiers for these larger scales.

In the experiments presented here we rescaled all examples to all scales, without taking any measure to lessen the negative effect of blurred examples. We acknowledge that a better handling of scales during training will certainly lead to a further improved quality. However we focus on speed more than quality.

Another issue to handle during training is calibrating the different detectors amongst themselves. Here, again, we take a simplistic approach that leaves room for improvement. We simply normalize the maximum possible score of all detectors to 1.

4. Soft cascade design

Up to now we have discussed speed(-up) results when using all of the 2000 stages of our base Adaboost classifier. Dollar et al. suggested to use a soft-cascade to accelerate the detections. The soft-cascade aborts the evaluation of non-promising detections if the score of a given stage drops below a learned threshold. The suggested method [19] sets such stage threshold at the minimal score of all accepted detections on a training or validation set.

In our experiments building such cascade over the training set leads to over-fitting of the thresholds and poor detections at test time. Instead we adjusted quality results without using soft-cascade, and then tuned the soft-cascade to keep the exact same quality results, but provide the desired speed-up. In practice, we use the INRIA test set as a val-

idation set to adjust the soft-cascade that will be used to accelerate the results on the Bahnhof dataset (see section 6).

After using the INRIA test set, adding a small offset (10% of the lowest threshold) allowed to make the cascade run as desired (higher speed, same quality). In the `VeryFast` method, each model has its own specific soft-cascade thresholds.

Algorithmic speed-up The speed-up of a soft-cascade is content dependent and hard to predict, however a ten times speed-up is expected. The soft-cascade should equally benefit the detection scores stage of `ChnFtrs`, `FPDW`, and `VeryFast`. Since the latter spends a larger portion of the time computing score (and a lower portion computing features), we expect it to benefit the most from the soft-cascade.

Measured speed-up When using the cascade our `VeryFast` implementation has a $20\times$ speed gain, reaching 50 Hz (see section 6 for speed evaluation details). In comparison `ChnFtrs` and `FPDW` barelly reach a $5\times$ speed gain ($\sim$ 10 Hz). This is mainly due to the need to alternate between features computation and detection scores which significantly hinders the GPU speed. Even if this was not a factor, `VeryFast` would still be faster, since it requires less computation by design (as seen in section 3).

5. Exploiting geometry

Previous work has shown that using scene geometry as prior for object detection can improve both the quality (by re-weighting the detection scores) [9, 14] and speed (by reducing the detections search space) [17, 3].

Common approaches based on processing dense stereo depth maps are a no-go, since producing depth maps at 100 Hz is a challenge in itself. Instead we follow the approach of Benenson et al. [3], where objects above the ground are modelled using the so-called "stixel world model" (stixel $\approx$ sticks above the ground in the image) [1]. For each column in the image, the bottom pixel, top pixel and distance to the (unclassified) object are estimated (see figure 1). The key feature of this approach is that the stixel world model can be estimated directly from the stereo images quickly, without having to compute the full depth map.

In our implementation we are able to estimate the ground plane and the stixels at about 135 Hz, using only a CPU, 80 disparities, and a fixed stixel height of 1.75 m.

Although Benenson et al. [3] presented the idea of coupling a detector with stixels, they did not realize such coupling. Showing the actual speed impact of tightly coupling stixels estimation and objects detection is a contribution of this paper.

In section 6.2 we compare the unconstrained detections using our detector, those constrained by a ground plane estimated for each stereo-frame, and those using stixels to constrain the detections.

When using a ground plane estimate, the detections are required to touch the ground plane with their bottom within a margin (e.g. ± 30 pixels). When using stixels, they are required to fit the bottom of the stixel crossing the detection centre (up to the same margin as in the ground plane case). We also limit the scales of detection to a small range around the scale indicated by the central stixel distance (e.g. ± 5 scales, when scale step is 1.05).

Algorithmic speed-up When processing a 640×480 pixels image over 55 scales, using stixels with ± 30 pixels and ± 5 scales provides a $44\times$ reduction in search space (since we only consider 640×60 pixels $\cdot$ 10 scales). In comparison using ground plane constraints only provides a $8\times$ reduction in search space (since we only consider 640×60 pixels $\cdot$ 55 scales). We show in section 6.2 that these parameters values provide no relevant degradation in the detection quality.

Measured speed-up Unconstrained detections run at 50 Hz (see section 4), however we may still want faster detections when additional computations are done in the real-time system. When using the ground plane we reach 100 Hz (ground plane computation itself runs at 300 Hz on CPU). When using stixels, the detections run at 145 Hz on GPU, but the stixel estimation itself runs at 135 Hz on CPU, making detections CPU bound at 135 fps.

For both ground plane and stixel constraints the speed-ups obtained (see table 1) are lower than the candidate window search space reduction because the discarded areas also are those where soft-cascade (section 4) is most effective.

The details of the speed evaluation and the detection quality are presented in section 6.

6. Pedestrian detection at 100 fps

In the past sections we have presented the evolution from a baseline CPU detector running at 0.08 Hz up to GPU detections at 135 Hz. We resume this evolution in the table 1.

Speed-wise our monocular results at 50 Hz are more than 7 times faster than the reported 6.5 Hz on CPU from Dollar et al. [8]. Also our result is 10 times faster than the `cudaHOG` results reported from the GPU implementation in [17], and at the same time the quality is twice as good than `cudaHOG` (see section 6.1).

Speed measurement We measure the speed taken by the CPU+GPU starting when the rectified stereo images are

Detector aspect	Relative speed	Absolute speed
Baseline detector (§2)	1×	1.38 Hz
+Single scale detector (§3)	2×	2.68 Hz
+Soft-cascade (§4)	20×	50 Hz
+Estimated ground plane (§5)	2×	100 Hz
+Estimated stixels (§5)	1.35×	135 Hz
Our monocular detector	-	50 Hz
Our stereo (stixels) detector	-	135 Hz

Table 1: Relative speed-up of each aspect of the proposed detector, with respect to the baseline detector.

available both to the CPU and the GPU. The measured time, does include all CPU computations, GPU computations and the time to download the GPU results and run the non-maximum suppression on CPU. The ground plane and stixels are estimated at frame $t-1$ and fed to the GPU computations at frame t. All speed results are given when computing over the Bahnhof images (640×480 pixels) over 55 scales (unless otherwise specified), averaged over the 1000 frames of the sequence.

As previously indicated our desktop computer is equipped with an Intel Core i7 870 and an Nvidia GeForce GTX 470. Our fastest result `VeryFast+stixels` is CPU bound (GPU runs at 145 Hz, CPU at 135 Hz), however the current CPU stixels code is sub-optimal and we believe it should be amenable for further speed-up (to match the GPU speed).

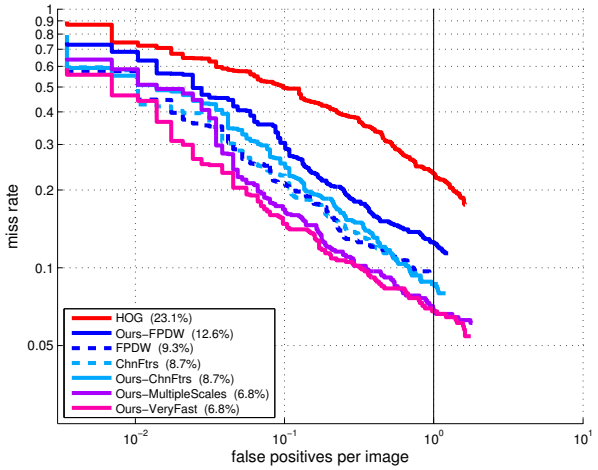
When running on a high end laptop (Intel Core i7-2630QM @ 2.00GHz, Nvidia GeForce GTX 560M), we reach 20 Hz for `VeryFast`, 38 Hz for `VeryFast+ground plane`, and 80 Hz for `VeryFast+stixels`.

6.1. INRIA Persons dataset results

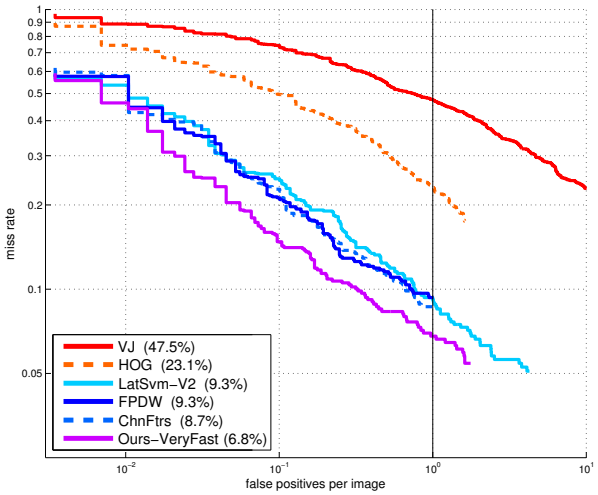
We use the INRIA dataset to train our detector and to evaluate its quality. Although this dataset is rather small, the diversity of its content helps to highlight the differences in performance of various methods. As a matter of fact, the relative ordering of methods seems roughly preserved across different pedestrian datasets [8].

In figure 3a we present the results of the different detector variants discussed in section 3. We also evaluate using the $^N/_K$ detectors, while still rescaling the input image to compute the feature responses at different scales (i.e. we do not use the `FPDW` approximation), this variant is named `MultipleScales` detector.

Figure 3b compares our detector with other state-of-the-art methods. Our detector is competitive in terms of the detection quality with respect to `ChnFtrs` and provides significant improvement over `HOG+SVM`.



(a) Quality of our detector variants (and reference detectors)



(b) Comparison with other methods

Figure 3: Results on the INRIA persons dataset.

6.2. Bahnhof sequence results

The Bahnhof sequence presents a challenging stereo sequence, acquired from a stroller moving along a crowded side-walk. This sequence allows us to evaluate the benefits of using stereo information and its impact on detection quality. We use the PASCAL VOC evaluation criterion.

The evaluation from Dollar et al. [8], showed that on this sequence the results between different methods is significantly reduced (due to the low intra-variance of the dataset). On this sequence we expect `ChnFtrs` to be only marginally better than `HOG+SVM` from Dalal and Triggs [4].

In figure 4 we present the results obtained from the methods described in section 5. We observe that the quality of our detector stays roughly constant when using ground plane and stixels, despite the 2.7× speed-up and reaching 135 fps. Equally important, we show that our `VeryFast`

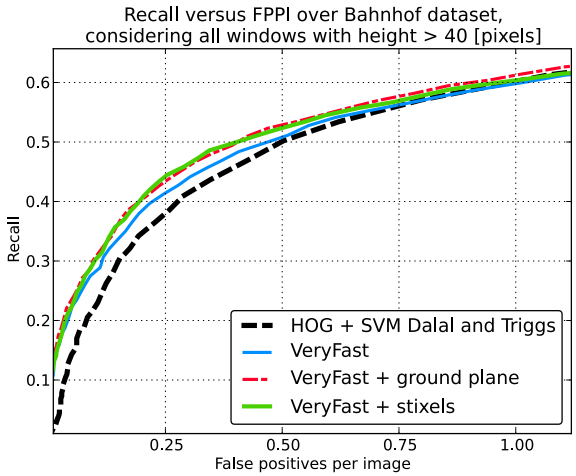


Figure 4: Results on the Bahnhof stereo sequence.

detector is above the HOG+SVM, confirming the expected quality gain.

In [3] the authors presented higher quality detections when using stixels than when using ground plane. We do not observe this in figure 4 (ground plane and stixels have overlapping quality). This can be explained by a few factors. First, our detector makes different kind of errors than HOG+SVM, which changes the stixels related gain. Second, we use the stixels to limit the scales search (to gain more speed), while [3] did not consider this in their black box. Thirdly and more importantly, to reach our desired speed we use the stixels from $t-1$ to guide the detections are frame t, this was a slight negative impact on quality. All factors together, using stixels provides a pure speed gain, with no noticeable quality loss.

7. Conclusion

We presented a novel pedestrian detector running at 135 fps in one CPU+GPU enabled desktop computer. The core novelties of our approach are reverting the FPDW detector of Dollar et al. [6] in order to avoid resizing the input image at multiple scales and using a recent method to quickly access to geometric information from stereo [3].

Our approach tallies with the Viola and Jones idea of "scale the features not the images" [18], applied to HOG-like features.

Given the high parallelism of our solution, it will directly benefit from future hardware improvements. We wish to improve the quality of the classifier training (see section 3.3), and extend the current system to the multi-class/multi-view detection of cars, bikes and other mobile objects. We are also interested in exploring a monocular equivalent of the current stereo stixels estimation.

Acknowledgement Work partly supported by the Toyota Motor Corporation, the EU project EUROPA (FP7-231888) and the ERC grant COGNIMUND.

References

[1] H. Badino, U. Franke, and D. Pfeiffer. The stixel world - a compact medium level representation of the 3d-world. In *DAGM*, 2009. 5

[2] M. Bajracharya, B. Moghaddam, A. Howard, S. Brennan, and L. H. Matthies. A fast stereo-based system for detecting and tracking pedestrians from a moving vehicle. *IJRR*, 28:1466–1485, 2009. 1

[3] R. Benenson, R. Timofte, and L. Van Gool. Stixels estimation without depthmap computation. In *ICCV, CVVT workshop*, 2011. 2, 5, 7

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, CA, USA, 2005. 1, 2, 3, 6

[5] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *ECCV*, 2006. 1

[6] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010. 2, 3, 7

[7] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009. 2, 3

[8] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *TPAMI*, 2011. 2, 5, 6

[9] A. Ess. *Visual Urban Scene Analysis by Moving Platforms*. PhD thesis, ETH Zurich, October 2009. 2, 5

[10] P. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010. 2, 3

[11] C. Keller, D. Fernandez, and D. Gavrila. Dense stereo-based roi generation for pedestrian detection. In *DAGM*, 2009. 1

[12] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: object localization by efficient subwindow search. In *CVPR*, 2008. 1, 2

[13] A. Lehmann, P. Gehler, , and L. Van Gool. Branch&rank: Non-linear object detection. In *BMVC*, 2011. 2

[14] D. Park, D. Ramanan, and C. Fowlkes. Multiresolution models for object detection. In *ECCV*, 2010. 2, 5

[15] M. Pedersoli, A. Vedaldi, and J. Gonzalez. A coarse-to-fine approach for fast deformable object detection. In *CVPR*, 2011. 2

[16] V. Sreekanth, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Generalized RBF feature maps for efficient detection. In *BMVC*, 2010. 1

[17] P. Sudowe and B. Leibe. Efficient use of geometric constraints for sliding-window object detection in video. In *ICVS*, 2011. 2, 5

[18] P. Viola and M. Jones. Robust real-time face detection. In *IJCV*, 2004. 1, 2, 7

[19] C. Zhang and P. Viola. Multiple-instance pruning for learning efficient cascade detectors. In *NIPS*, 2007. 4

[20] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR*, 2006. 2

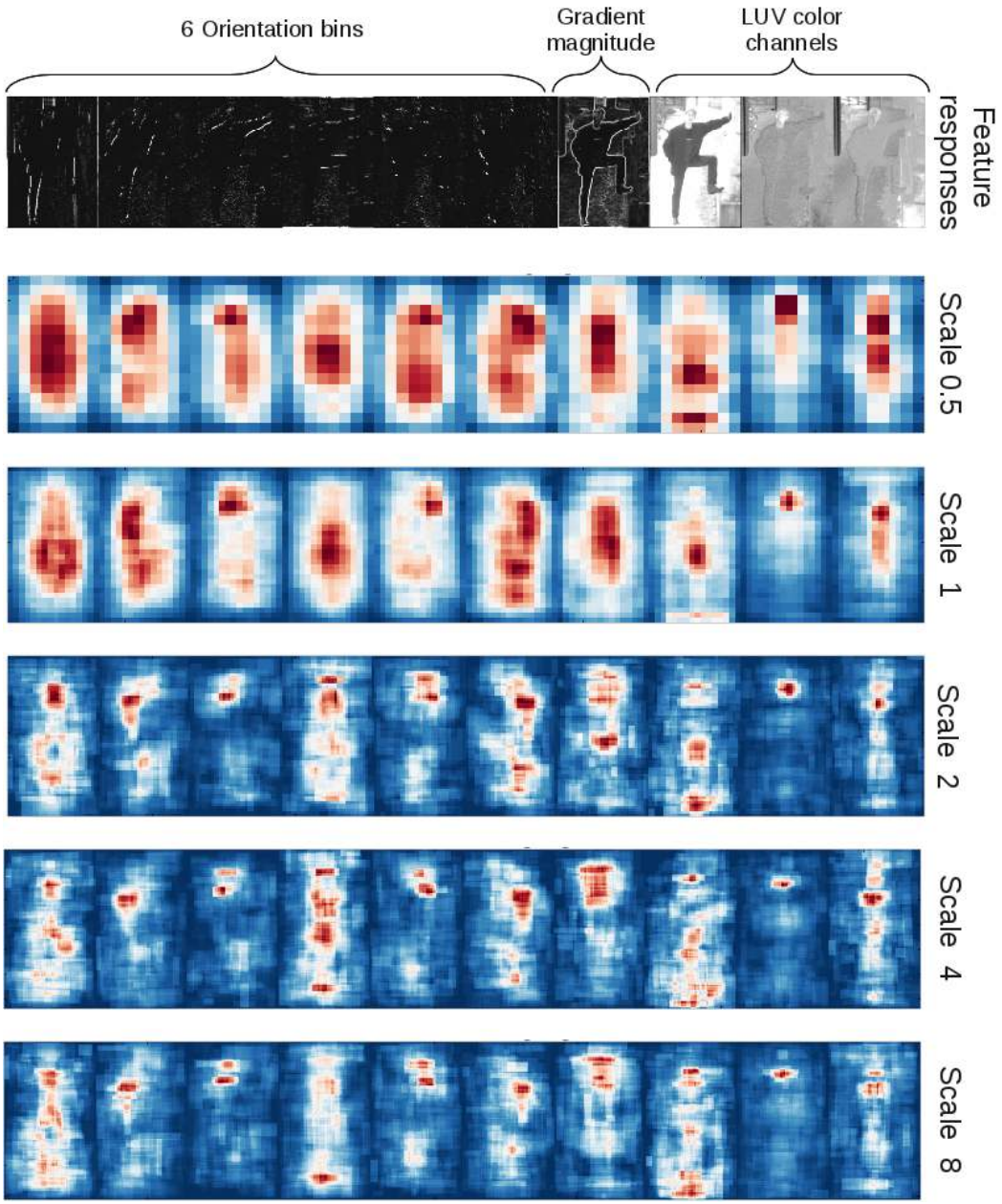


Figure 5: Visualization of the trained multi-scales model. First row shows an example of the different features used, one per column. Each row below shows the trained model, one scale per row. Each model column is individually normalized to maximize contrast (relative influence not visible). Red and blue indicate positive and negative contributions to the detection score, respectively. Scale one has size 64×128 pixels.