Seeking the strongest rigid detector

Rodrigo Benenson*†‡ Markus Mathias*† Tinne Tuytelaars† Luc Van Gool†
† ESAT-PSI-VISICS/IBBT,
Katholieke Universiteit Leuven, Belgium
firstname.lastname@esat.kuleuven.be

‡ Max Planck Institut für Informatik
Saarbrücken, Germany
benenson@mpi-inf.mpg.de

Abstract

The current state of the art solutions for object detection describe each class by a set of models trained on discovered sub-classes (so called "components"), with each model itself composed of collections of interrelated parts (deformable models). These detectors build upon the now classic Histogram of Oriented Gradients+linear SVM combo.

In this paper we revisit some of the core assumptions in HOG+SVM and show that by properly designing the feature pooling, feature selection, preprocessing, and training methods, it is possible to reach top quality, at least for pedestrian detections, using a single rigid component.

We provide experiments for a large design space, that give insights into the design of classifiers, as well as relevant information for practitioners. Our best detector is fully feed-forward, has a single unified architecture, uses only histograms of oriented gradients and colour information in monocular static images, and improves over 23 other methods on the INRIA, ETH and Caltech-USA datasets, reducing the average miss-rate over HOG+SVM by more than 30%.

1. Introduction

Many of the current top performing methods for object detection are variants of the deformable part models work of Felzenszwalb *et al*. [7]. Through components and parts, these models are built upon sets of non-deformable, weaker, classifiers. These rigid classifiers are built using the now classic HOG+SVM (histogram of oriented gradients, plus linear support vector machine) detector, introduced by Dalal and Triggs [3].

Fast and high quality detection enables a vast range of applications, which has motivated a large amount of research on the topic. In this paper we focus on pedestrian detection, since it is a challenging class of great practical interest. We believe that most of the results and conclusions obtained here do apply to other classes as well. For pedestrian detection HOG+SVM, although not state-of-the-art anymore, is still surprisingly competitive [6].

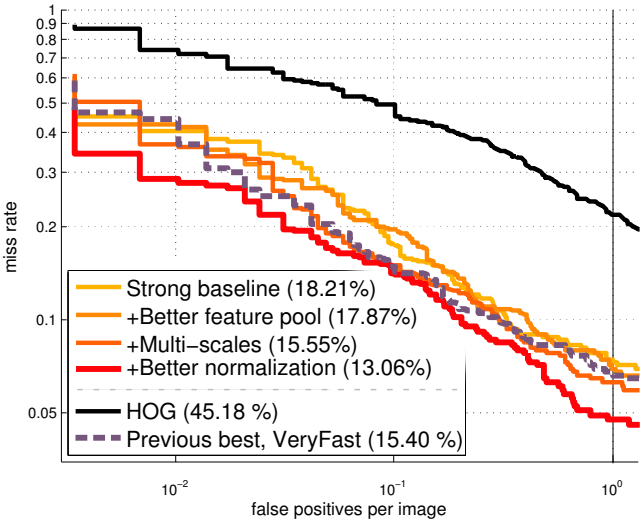


Figure 1: Progress obtained from each section of the paper. From an already strong baseline up to our final detector, on the INRIA dataset. See also figure 7a.

In this paper we propose to revisit this low-level rigid detector. By reconsidering some of its assumptions and design choices, we show that it is possible to have significant quality improvements, reaching state of the art quality on par with flexible part models, while still using only HOG + colour information, in a single rigid model.

In the spirit of the original work on HOG+SVM [3], we provide a new extensive set of ~40 experiments, revisiting the design space of classifiers based on oriented gradient histograms. We provide results per image (instead of per window [3, 5]), and perform evaluations on large datasets. Our final single component rigid classifier reaches record results on INRIA, ETH and Caltech-USA datasets, providing an average miss-rate reduction of more than 30% over HOG+SVM.

Our work is based on the integral channel features detector family, introduced by Dollár *et al*. [5], an extension

*Indicates equal contribution

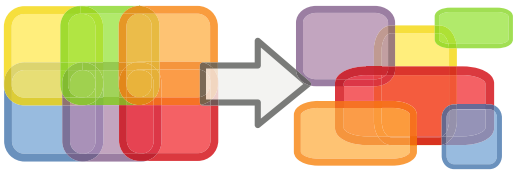


Figure 2: In traditional HOG the histograms are computed over cells in a regular pattern (left side), while we learn in a discriminative fashion an irregular pattern (right side).

of HOG+SVM. Before detailing how our detector works, we give an appetizer by highlighting how our approach differs from HOG+SVM [3] and how it relates/contrasts to more recent work.

Irregular cells The HOG classifier builds its descriptor by computing histograms over regular square cells of fixed size. In our detector, similar to [5, 9, 16], the set of rectangular cells is selected during learning (see figure 2). Section 4 shows evidence of the benefit of irregular cells. Learning the set of rectangles is significantly better than using a hand-designed pattern.

Feature pre-processing Low level details have an important impact on the final quality. Which kind of low level transformation/normalization is used varies significantly amongst methods. Some use no normalization [2], some use local normalization [1, 3, 8]. In HOG+SVM [3] the importance of using local normalization at multiple scales is emphasized. In section 5 we show the effect of such factors in our setup. We discover that global normalization can be surprisingly effective.

Feature channels The integral channel features framework [5] (see section 2) enables the use of multiple kinds of "channels" (low level pixel-wise features). We constrain ourselves to using only HOG and LUV colour channels, because these are still close to the original HOG+SVM work, and at the same time were validated as best performing [5].

Non-linear classifier The HOG+Linear SVM obtains its score via a linear combination of the constructed feature vector. The classifier is non-linear with respect to the input image due to the HOG features and their normalization. In our case the classifier is non-linear due to the use of decision trees as weak learners over HOG features. Multiple kinds of non-linearities on top of HOG have been explored in the past, including neural network sigmoid functions (and variants) [14], as well as different kinds of stumps and decision trees [5, 10]. In section 6 we evaluate the impact of such choices.

Multi-scale model Using multiple scales has been shown to improve quality [12], and also speed [2]. In [12] both low and high-resolution models are used to evaluate a single candidate window. Similarly, all components of [7] are evaluated for each candidate window. Instead, following [2], we evaluate a single model per candidate window. Using multi-scale models has a significant positive effect on our final model. See section 9.

Learning algorithm We use boosting instead of linear SVM. It has been previously shown how different boosting approaches give different results [5, 17]. We revisit the question in section 7.

Speed Although HOG+SVM is not particularly slow, the specific normalization mechanism used, and the use of high dimensional vectors hinder speed. Our approach is compatible with the VeryFast approach [2] (where pixels are processed only once at the original image scale), and with the sophisticated cross-talk cascades [4] for fast evaluation.

Single rigid template In this paper we focus on the lower level of more sophisticated part and component-based classifiers. Just as in the original HOG+SVM we use a single rigid template per candidate detection window. For detection we only use a single static colour image. We do not use motion, stereo, or geometric priors in the scene.

1.1. Contributions

- We perform a detailed investigation of the different factors affecting the integral channel features detectors family. We provide context and contrast our work with the "mainstream" HOG+Linear/NonLinear SVM classifiers.
- Dollár *et al*. did ~80 experiments exploring the design space of the channel features detector [5, and addendum]. We add here another ~40 complementary ones, and provide additional guidelines for practitioners. In our experiments we evaluate false positives per image (FPPI), while Dollár's are done per window (FPPW), which is flawed, as argued in [6].
- We push the state-of-art forward with new record quality, using a single rigid template detector using HOG+colour only. We obtain top performance on INRIA, ETH and Caltech-USA, improving over 23 other methods including non-linear SVMs, more sophisticated features, geometric priors, motion information, deformable models, or deeper architectures.

Sections 2 and 3 describe our base classifier and the evaluation setup. Sections 4, 5, 6, 7, 8 explore different aspects affecting its quality. In section 9 we use the learned lessons to build our final strong classifier. We conclude and delineate future work in section 10.

2. Integral channel features classifier

Our starting point is the Integral Channel Features detector [5]. In a sense, it can be seen as a combination of

the classic Viola and Jones work (VJ) [16] with Dalal and Triggs' (HOG+SVM) [3].

Given an input image, a set of gradient and colour "channels" are computed (pixel-wise transforms). As for HOG+SVM, quantized oriented gradients are extracted. Then low level features are built by summing over rectangular regions. Like VJ these rectangular regions are then selected and assembled in a set of weak classifiers using boosting. The final strong classifier is a linear combination of the weak classifiers. Without further specifications, the Integral Channel Features detector describes a family of detectors.

ChnFtrs detector Typically VJ provides low quality (see figure 7) because it is applied directly over the image intensity channels. Dollár *et al*. showed that by applying a similar approach over oriented gradients, the quality improves drastically. Amongst the different designs explored, they propose to select the so called ChnFtrs detector [5]. HOG and LUV channels are computed (10 channels in total), and 30 000 random rectangles are used as feature pool. The features are assembled into a linear combination of 2 000 level-2 decision trees (containing 3 stumps), using Adaboost.
Training is done in 3 stages. The first stage randomly samples 5 000 negative samples, the second and third stages use bootstrapping to add 5 000 additional hard negatives each.
The learned strong-classifier is applied at test time as a sliding window over the image. To make things faster (with no significant impact on quality), the coordinates of the feature pool and of the candidate detection windows can be quantized by a factor 4 (so called "shrinking factor"). To obtain the final detections, a non-standard greedy non-maximal suppression method is used (greedy⋆). For more details please consult [5, and its addendum].

VeryFast detector One peculiarity of the VJ approach, is that the detector runs directly on the input image, without the need to resize the image nor recompute features (and integral images) multiple times at different scales. The VeryFast detector proposed by Benenson *et al*. [2] achieves this too, while at the same time improving on the quality of the ChnFtrs. By learning models at multiple canonical scales, this detector better exploits the information available in the image, leading to better detection. Our final detector will use this strategy too.
The starting point for our work is the open source release of the VeryFast detector [2]. To enable our experimental analysis, we have improved the already multi-threaded and GPU enabled training code to accelerate it by a factor of 3.

Our strong baseline The training of ChnFtrs includes a randomness factor in the selection of candidate features. To avoid this source of variability we build a deterministic baseline named SquaresChnFtrs, where the feature pool is composed of all the squares that fit inside the model window. Everything else is identical to ChnFtrs. Our baseline already beats a dozen of others approaches (see section 9). This baseline is kept as a reference in all our experiments (using a deep blue colour).

Despite their good results the ChnFtrs and VeryFast detectors still leave a large number of free design parameters, such as: how to select the rectangular features? how many of them to use? how should these features be normalized? which kind of weak classifier to use? which training method? which training data? We attempt to answer all these questions in the next sections.

3. Experimental setup

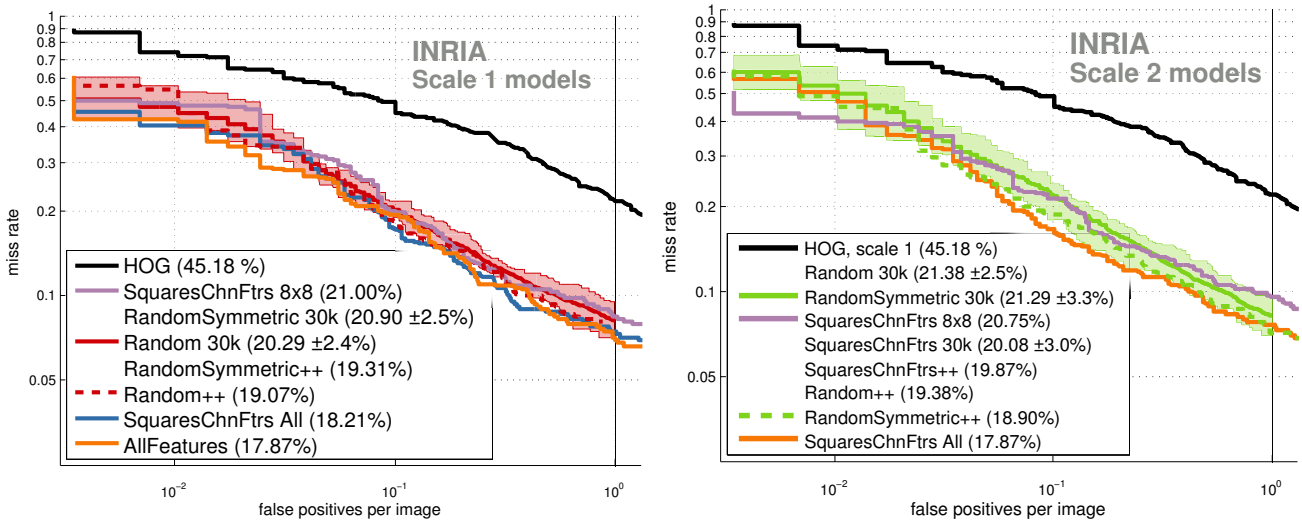
For evaluation we use the Caltech pedestrian detection benchmark, version 3.0.1. This benchmark provides software for evaluation and comparison of a dozen of state of the art methods over multiple datasets. This software takes care of the delicate issues arising when evaluating detection algorithms [6]. We use for evaluation the INRIA (diverse scenes, ~ 500 annotated pedestrians), ETH (sidewalks, $12k$ pedestrians), and Caltech-USA dataset (car view, $1k$ pedestrians). Thanks to direct correspondence with the authors we also include results for the competitive EBLearn [14], VeryFast [2], and MLS [10] methods.

All our models are trained using the INRIA pedestrians dataset [3]. The evaluations of the experiments are done per image (not per window) [6]. We use the INRIA test set as validation set. ETH is only used for the experiments of section 5, all other results on ETH (in the supplementary material) were done after fixing the parameters. Caltech-USA was used solely for the final evaluation.

For evaluation we use the corrected INRIA annotations provided by [14], which include additional "ignore" windows, to avoid penalizing methods that detect very small pedestrians (we saw no change in the trends when changing annotations). The mean miss-rates reported in the legend of each figure (e.g. figure 3) are based on 200 samples of the curve (in the range 0 to 1 FPPI, evenly spaced in log-space), instead of only 9 as used in the original benchmark (the average is thus more precise).

For all implementation details, we follow as precisely as possible the methodology described in [2, 5]. Unless specified otherwise, we use models of size 64×128 pixels, named "scale 1", although the final classifier in section 9 uses different models for each scale.

It should be noted that we took particular care of handling the sources of randomness. All experiments are done using the same set of random seeds, to make the curves comparable.



(a) Here all models use a model window of size 64×128 pixels.

(b) Here all models (but HOG) use a window of size 128×256 pixels.

Figure 3: Detector quality on INRIA using different feature pool settings. Lower average miss-rate is better. All curves available in the supplementary material.

4. Which feature pool?

The feature pool (set of rectangles) used to construct the weak learners is one of the key design choices during the training of `ChnFtrs`. It is known that having a proper feature pool impacts quality [9]. A priori, there is no particular reason to believe that the regular pattern used by `HOG+SVM` is optimal. On the other hand, simply using random features is bound to lead to sub-optimal results.

Using too many features puts a heavy memory load during training time, usually reaching the machine limits. Using too few features will impoverish the capacity of the classifier, and lead to bad quality at test time. Unfortunately, the space of rectangles inside a 64×128 pixels model is very large (even when using a shrinking factor of 4). Even worse, when training a multi-scales detector (such as `VeryFast`), models of twice and four times the size need to be trained, making the set of all possible rectangles explode in size.

Experiments at scale 1 We first perform a set of experiments at scale 1 – see figure 3a. Some curves are omitted for clarity. All curves of all experiments on both INRIA and ETH can be found in the supplementary material.

`Random 30k`: As a baseline we train 10 models using 30 000 random rectangles, measure the average performance, and the max-min envelope.

`RandomSymmetric 30k`: We hypothesise that the detector might benefit from comparing the same feature across different channels, or in the same channel using reflection symmetry across the vertical axis. We generate 150 random rectangles on a single channel, mirror them and copy these 300 features in all 10 channels. We train 10 such models.

`SquaresChnFtrs-8x8`: Instead of using random rectangles, we imitate `HOG+SVM` by using only squares of 8×8 pixels, positioned regularly each 4 pixels. Everything else is just as in `ChnFtrs`. Please note that this experiment has the same symmetry as `RandomSymmetric`. In this setup only 5 120 features are considered.

`SquaresChnFtrs All`: This is the strong baseline described in section 2. We sample squares of all sizes, positioned regularly each 4 pixels. A similar pattern was suggested in [10]. In this setup 38 400 features are considered.

`AllFeatures`: Using 90 Gbyte, 16 cores, on a GPU enabled server, we were able to run an experiment using all rectangles for a 64×128 pixels model (with shrinking factor 4). In this setup 718 080 features are considered.

`Random++`: The model learned in `Random 30k` depends on the random seed used. To reduce the randomness effect and build a stronger classifier, we aggregate features selected by Adaboost in 10 runs of `Random 30k` (at scale 1 many of them are repeated). This new bootstrapped feature pool is then employed to initialize an 11th training, named "++".

*++: Similar method as `Random++` applied to the corresponding set of random models.

Experiments at scale 2 Since the problem of finding good features is more acute at larger scales, we think it is relevant to also present experiments when computing `AllFeatures` is not an option anymore. At scale 2 the model to learn has twice the size of scale 1. See figure 3b.

`SquaresChnFtrs 30k`: Like `SquaresChnFtrs`

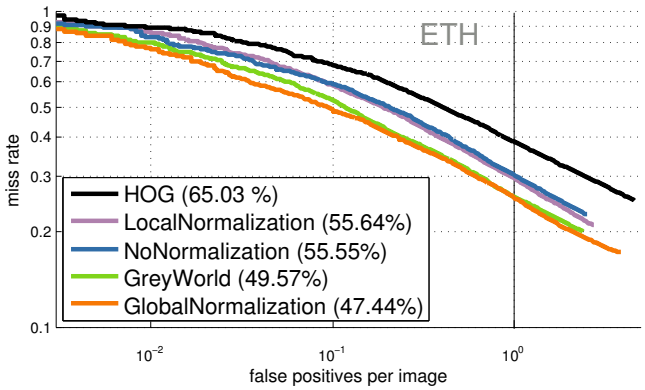


Figure 4: Detector quality when using different feature normalization schemes.

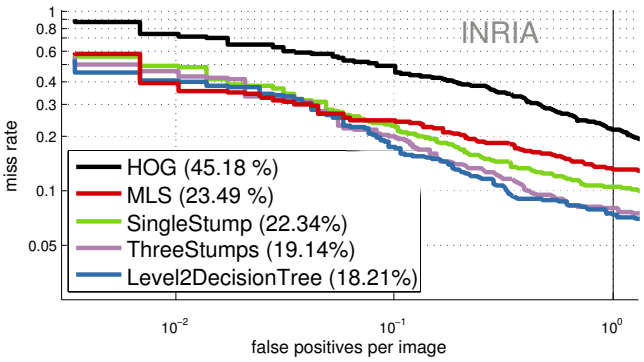


Figure 5: Which weak classifier to use?

`All` but only a subset of 30 000 random features is considered.

Analysis We observe in figure 3 that although `Random 30k` seems like a reasonable choice at scale 1, it becomes increasingly worse with increasing scales. As expected, using `AllFeatures` is the best choice when possible, `SquaresChnFtrs All` being a close second best. When neither of these two are tractable, `Random++` provides the best choice (see also supplementary material). The "++" variants seem to systematically improve over the mean of their random pool. Notice that the "++" approaches trade-off exploding growth in memory usage, with an increase in training time (but low memory needs).
Note that all the curves are significantly better than `HOG+SVM`, and that `SquaresChnFtrs-8x8` is not particularly effective. In the supplementary material we provide additional plots showing statistics on the selected features for `AllFeatures`.

Conclusion Having a good coverage of the feature space seems directly related to the quality of the final detector. Depending on your available computing resources we recommend `AllFeatures>SquaresChnFtrs All>Random++`.

5. Which feature normalization?

In the re-implementation of the `ChnFtrs` detector done by Benenson *et al*. [2] no illumination normalization is used. In [5, addendum] it was shown that normalizing the gradient magnitude by considering a small neighbourhood (4×4 pixels) provides a small improvement. In [3] it is argued that representing the same value with multiple normalizations is a key element for quality. Since our baseline detector is based solely on thresholds, injecting some invariance to illumination changes seems reasonable.

Experiments All experiments are done on top of the `SquaresChnFtrs` classifier (see section 2). The INRIA dataset was captured using consumer cameras, which perform on-board colour and illumination adjustment. Thus this dataset seems rather well behaved illumination-wise compared to datasets that are acquired with a mobile setup. Therefore, we choose to report experiments on ETH, where the effect of normalization is more pronounced. The corresponding INRIA curves are in the supplementary material.
`NoNormalization`: alias for `SquaresChnFtrs`, as described in section 2.
`GlobalNormalization`: As a sanity check, we consider first doing a global image normalization, before computing the channels. We use the "automatic colour equalization" algorithm (`ACE`) which provides better results than a simple `GreyWorld` equalization [13].
`LocalNormalization`: For local normalization we look for a more principled approach than the proposal of [5, addendum]. We follow the normalization employed by [1, equation 19], where the gradient orientation features are normalized by the gradient magnitude in the same area.
Our attempts to implement Dollár *et al*.'s normalization produced results worse than `LocalNormalization`.

Analysis As expected, normalization improves over using no normalization. Surprisingly even a simple global normalization such as `GreyWorld` already provides an important gain. In our setup, standard normalization schemes are puzzlingly ineffective. In other experiments we have validated that global normalization improves detections even when using only monochromatic images.

Conclusion Simple global normalization such as `GreyWorld` are very effective and should not be disregarded. The slower `ACE` algorithm [13] is even better.

6. Which weak classifier?

Boosting methods construct and compound weak classifiers. Choosing the kind of weak classifier and its weak learner, are an important design decision.

Experiments
`SingleStump`: we train `SquaresChnFtrs`, using

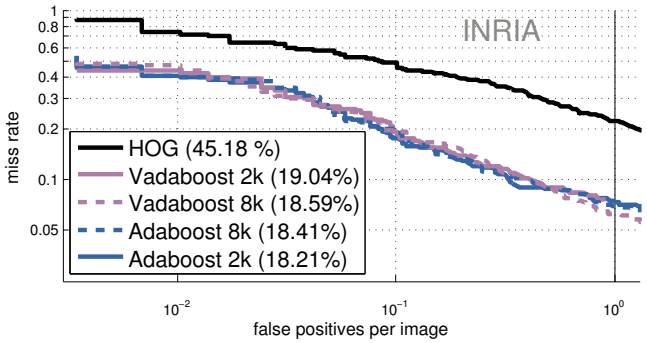


Figure 6: Which training method to use?

6 000 stumps (same number of stumps as in the baseline). Stumps are the simplest weak classifiers. In [5] only a single point of the FPPW evaluation is given and no FPPI curve is presented.
`Level2DecisionTree`: alias for `SquaresChnFtrs`, as described in section 2. Each tree contains three stumps.
`ThreeStumps`: In modern parallel machines, having two parallel threads selecting different sides of a level-2 decision tree has a cost similar to evaluating both sides. Inspired by the work on ferns [11], we transform the level-2 decision tree in a three bits vector, and use it to index a table with 2^3 entries. With a similar computational cost, we obtain a (slightly) more discriminative weak classifier.

Analysis The results in figure 5 seem to indicate that `Level2DecisionTree` are a sweet spot for Adaboost. Using slightly more discriminative weak classifiers seems not to improve quality.
The `MLS` method [10] explored constructing boosted classifiers on top of HOG cells, using different arrangements. On the INRIA test set, our approach performs significantly better.

Conclusion Without changing the learning method, `Level2DecisionTree` seems to be the adequate choice for this architecture.

7. Which training method?

Since the introduction of Adaboost, the boosting principle has been instantiated in a myriad of variants, each of them claiming to be superior to the others. In the context of image classification it is unclear which method is best in practice. The experiments in [5], comparing Adaboost, Realboost, and LogitBoost, showed insignificant quality differences.

Experiments
`Adaboost`: alias for `SquaresChnFtrs`, as described in section 2.
`VadaBoost`: A regularized Adaboost variant that minimizes not only the margin average, but also its variance [15].
The variants `2k`/`8k` indicate the number of weak learners used in each case.

Analysis The results of figure 6 show that none of the considered training methods is significantly better than our baseline. Making the classifier longer seems not to improve the quality either.

Conclusion It seems that changing features and weak classifiers (e.g. figures 3a and 5) matters more than which greedy boosting is used. The baseline `Adaboost 2k` still is the best choice.

8. Which training set?

It is well-known that the data used to train an algorithm is just as important as the algorithm itself [18]. The INRIA dataset has been regularly used for training pedestrian detectors [6, table 2], despite being quite small by today's standards. During the bootstrapping stages of learning, we observe that the baseline fails to find the desired amount of false positives (5 000 per round), pointing out the need for a richer set of negative images.

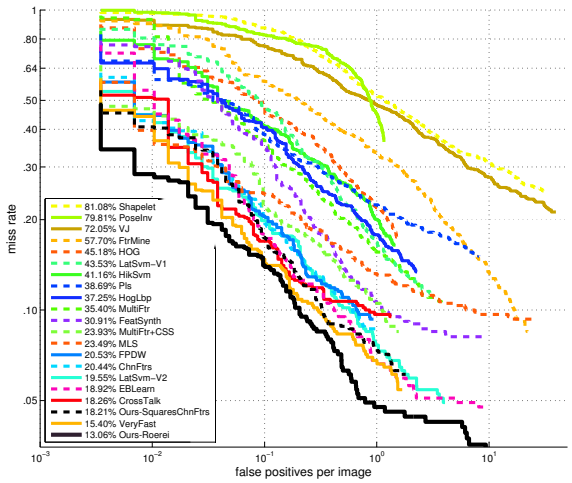
Experiments
`InriaOnly`: alias for `SquaresChnFtrs`, as described in section 2.
`SmallJitter`: augmentation of positive samples via mirroring, scaling, or jittering are a common practice when training neural networks. In our setup with shrinking factor 4, displacements of less than ± 2 pixels should appear as new "perfectly centred" samples. This is the level of jitter we test, using 9 random samples per training pedestrian.
`PedestriansNegatives`: we have observed that false positives occur frequently on the legs of large pedestrians. To mitigate this effect we propose to augment the negative set using crops of pedestrians with low overlap (based on the `greedy*` overlap criterion and threshold). We add 9 negative samples per each positive sample.

Analysis The result figure is available in the supplementary material. It indicates that `PedestriansNegatives` has no significant effect and that `SmallJitter` slightly worsens the quality ($-2\,\%$ on INRIA). It seems that the original annotations already contain enough natural jitter and adding more only hurts performance.

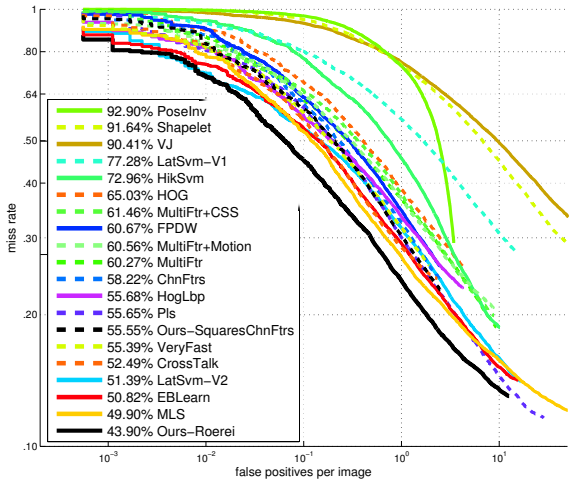
Conclusion The vanilla INRIA training data seems to hold as a good choice for top performance.

9. The `Roerei` detector

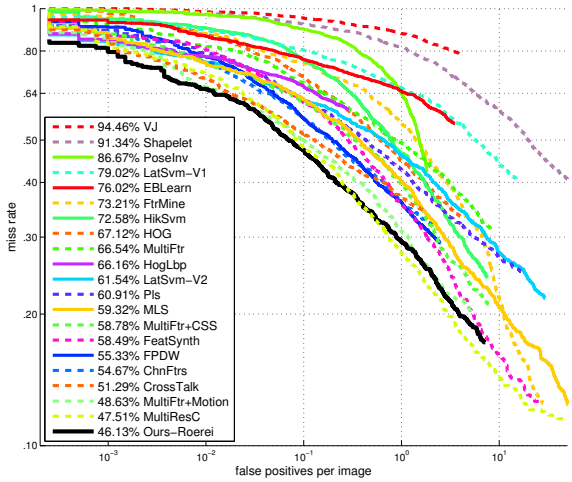
Given the lessons learned from the previous sections, we proceed to build our final strong classifier. We name it the `Roerei` detector (Dutch for "scrambled eggs", pronounced [rurɛi]), in allusion to the fact that, in contrast to the orderly pattern used in `HOG+SVM`, our learned detector has



(a) INRIA results



(b) ETH results



(c) Caltech-USA test set results

Figure 7: Comparison of different methods on different datasets. See also figure 1 and table 1.

Detector aspect	Average miss-rate	
	INRIA	ETH
Strong baseline (§2)	18.21%	55.55%
+ `AllFeatures` (§4)	17.87%	55.50%
+ Multi-scales (§2)	15.55%	53.17%
+ `GlobalNormalization` (§5)	13.06%	43.90%
= `Roerei` detector	**13.06%**	**43.90%**
`HOG+SVM`	45.18%	65.03%
Previous best, `VeryFast`/`MLS`	15.40%	49.90%

Table 1: How does quality improve at each stage?

"scrambled cells" of different dimensions and positions (see figure 2).

In table 1 we show the results obtained at each stage of improvement (see figure 1). Note that our *baseline* already beats 18 methods on INRIA, and 13 on ETH; including the original `ChnFtrs` detector. Improving the feature pool, using multi-scales (training models for scales 1, 2, 4 and 8), and using global normalization, leads us to our top performing `Roerei` detector. For scale 1 we use `AllFeatures`, for scale 2 `SquaresChnFtrs All`, and for the last two scales `RandomSymmetric++`.

The results of figure 7 correspond to a fixed model evaluated on the different datasets. We use 55 scales in all cases, and only change the search range amongst each dataset to reflect the different sizes of pedestrians on each scenario.

Our final detector shows a significant improvement over the previous best methods on these datasets, having an homogeneous (single stage) architecture, and evaluating a single rigid model for each candidate window. On INRIA we reach 96% recall at 10 FPPI. On Caltech-USA we improve over `MultiResC` [12] which also uses multi-scale models (but evaluates multiple models per window), and uses deformable parts, a weak geometric prior of the ground plane, and was trained over the Caltech training data using a sophisticated latent-SVM.

Training time As discussed in section 4, training with the best feature pool puts pressure on the memory usage. Training all 4 models for each scale increases the training time. The training time corresponds to 2.5 days on a GPU enabled, multi-core, large RAM work-station.

Test time speed Since in this paper we focus on quality, we use the multi-scale model without the speeding-up approximation used by `VeryFast` [2], obtaining thus the quality improvement but not all the speed benefits. The `ACE` global normalization is rather slow (5 seconds per frame), but the `GreyWorld` normalization is much faster (can be computed in less than 10 milliseconds) and provides similar benefits. All other proposals only affect training time. Using the soft-cascade employed in [2] or the cross-talk cascade of [4], the `Roerei` detector should run comfortably in

the range $5 - 20$ Hz. Without soft-cascade and after global normalization, the detections currently run at about 1 Hz.

Note for practitioners If there is one thing we have learned in our quest for high quality detections, it is how much low level details matter. We have observed up to 10% average miss-rate differences depending on how border conditions are handled, how exactly the features are computed, how the training data is cropped, and other similar implementation details.

Link to deep networks In figure 7 it can be noted that our method outperforms the deep convolutional neural network `EBLearn` [14]. Already our strong baseline `SquaresChnFtrs` obtains better results. Despite usual claims on "hand-designed features", we believe that the design space of convolutional networks is not smaller than that of the integral channel features classifiers. It should be noted however that both approaches are quite similar in spirit. Deep networks alternate linear and non-linear operators, ending with a simple classifier at the top layer. Roughly, our architecture first applies linear operators (oriented gradient filters convolutions and sum over rectangles), followed by non-linear look-up tables (decision trees). The final layer is a simple linear combination. Due to the similarity between these architectures, we believe that some of the observations in this paper can be used in the context of deep learning.

10. Conclusion

Throughout the paper we have revisited the design of a low level, rigid, HOG+colour detector. We have provided extensive experiments and identified the aspects that improve quality over our strong baseline (feature pool, normalization, use of multi-scale model). On the INRIA, ETH and Caltech-USA datasets, our new `Roerei` detector improves over methods using non-linear SVMs, more sophisticated features, geometric priors, motion information, deformable models, or deeper architectures. We think that combining these ideas will provide even further improvements.

The experiments show that overall we have moved a long way from the classic `HOG+SVM`. We hope that some of the practices presented here will serve future generation classifiers.

Acknowledgement Work partly supported by the Toyota Motor Corporation, and the ERC grant COGNIMUND. We thank P. Dollár and R. Appel for the insightful discussions, and S. Aihara for his `colorcorrect` python package.

References

[1] K. Ali, F. Fleuret, D. Hasler, and P. Fua. A real-time deformable detector. *PAMI*, 2011. 2, 5

[2] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *CVPR*, 2012. 2, 3, 5, 7

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 1, 2, 3, 5

[4] P. Dollár, R. Appel, and W. Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *ECCV*, 2012. 2, 7

[5] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009. 1, 2, 3, 5, 6

[6] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *TPAMI*, 2011. 1, 2, 3, 6

[7] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010. 1, 2

[8] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009. 2

[9] Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *CVPR*, 2012. 2, 4

[10] W. Nam, B. Han, and J. Han. Improving object localization using macrofeature layout selection. In *ICCV, Visual Surveillance Workshop*, 2011. 2, 3, 4, 6

[11] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *PAMI*, 2010. 6

[12] D. Park, D. Ramanan, and C. Fowlkes. Multiresolution models for object detection. In *ECCV*, 2010. 2, 7

[13] A. Rizzi, C. Gatta, and D. Marini. A new algorithm for unsupervised global and local color correction. *Pattern Recognition Letters*, 2003. 5

[14] P. Sermanet, K. Kavukcuoglu, and Y. LeCun. Traffic signs and pedestrians vision with multi-scale convolutional networks. In *Snowbird Machine Learning Workshop*, 2011. 2, 3, 8

[15] P. Shivaswamy and T. Jebara. Variance penalizing AdaBoost. In *NIPS*, 2011. 6

[16] P. Viola and M. Jones. Robust real-time face detection. In *IJCV*, 2004. 2, 3

[17] C. Wojek, S. Walk, and B. Schiele. Multi-cue onboard pedestrian detection. In *CVPR*, 2009. 2

[18] X. Zhu, C. Vondrick, D. Ramanan, and C. Fowlkes. Do we need more training data or better models for object detection? In *BMVC*, 2012. 6