Boosting a Heterogeneous Pool of Fast HOG Features for Pedestrian and Sign Detection

Gary Overett,Lars Petersson,Lars Andersson and Niklas Pettersson
National ICT Australia
Locked Bag 8001, Canberra, Australia
{gary.overett,lars.petersson,lars.andersson,niklas.pettersson}@nicta.com.au

***Abstract*—This paper presents a fast Histogram of Oriented Gradients (HOG) based weak classifier that is extremely fast to compute and highly discriminative. This feature set has been developed in an effort to balance the required processing and memory bandwidth so as to eliminate bottlenecks during run time evaluation. The feature set is the next generation in a series of features based on a novel precomputed image for HOG based features. It contains features which are more balanced in terms of processing and memory requirements than its predecessors, has a larger and richer feature space, and is more discriminant on a per feature basis.**

In terms of computational complexity it is a heterogeneous feature set. I.e. it has fast and slow variants. In order to optimize our feature selections between the faster and slower features available we implement a recently proposed modification to the RealBoost feature selection rule. This modification provides an additional means to balance processing and memory bandwidth on ordinary PC architectures.

This feature set is suitable for use within typical boosting frameworks. It is compared to Haar and Rectangular HOG features, as well the related feature HistFeat. The new feature set contains two variants, LiteHOG and LiteHOG+, which we compare. Both LiteHOG and LiteHOG+ show promising results on road sign and pedestrian detection tasks.

I. Introduction

Real-time visual object detection from a moving platform is a popular problem in vehicle based computer vision. In many such systems, both *error rates* and *time to decision* determine the value of a given solution. In the field of computer vision there are many successful detection approaches [1] [2] [3] [4]. The use of precomputed datatypes, such as the integral image in [1], facilitates a much faster feature evaluation process than previous methods. In fact, evaluation requires so little processing time that these features are limited by bandwidth to memory, not CPU bandwidth [5]. In response to this problem, Pettersson et al. [5] proposed a novel precomputed datatype, *the histogram image*, and a new feature, *HistFeat*, which is resource efficient in terms of memory and computation. In this paper, we will present two feature types which both make use of the memory efficient histogram image, the LiteHOG and LiteHOG+ features.

In the time since the widely lauded Viola & Jones [1] object detector, there has been a trend toward improved detection performance using ever more computationally complex features. A strong emphasis on error-rates has favored more discriminant features at the expense of computational performance (time to decision). This may be quite reasonable for offline applications but for tasks such as vehicle based pedestrian and road sign detection it is not suitable. In the case of pedestrian detection, Histograms of Oriented Gradients (HOG features) [2][3] and Edgelet features [6] have surpassed the error rate performance of the Haar feature, however, these features are both significantly slower. More recently, very high pedestrian detection rates have been attained using Region Covariance features [7] [8]. Detection performance is impressive but the computation and memory requirements of the Region Covariance feature are comparably huge.

For a time constrained problem we require an analysis not just of the error rates of different features but a comparison of their achievable error rates in a given amount of time. Of course, time constrained detection performance does not only depend on the chosen feature type but also on the chosen learning algorithm. AdaBoost [9], and its Real-Valued counterpart RealBoost [10], are common choices. However, researchers have found that for some problems, alternatives such as LogitBoost [11], decrease the classification error more quickly [7]. There are also notable learning algorithms that directly attempt to minimize time to decision, such as the WaldBoost algorithm [12].

Lastly, we note that per feature processing time may vary across a given feature space, with some features taking more processor time than others. Most popular boosting algorithms, including 'time conscientious' algorithms like WaldBoost, do not consider the possibility that the features available may have a different computational penalty. A small modification to the AdaBoost feature selection rule, is shown in [13]. This aids the selection of features with different computational speeds on a level playing field, see Section III-C. This small modification, which we apply to RealBoost [10], selects the feature which most rapidly decreases the error rather than selecting the most discriminant feature regardless of the time taken.

II. Balancing Processor and Memory Bandwidth

The arrival of precomputed datatypes has dramatically reduced the processing power required to evaluate features. Pettersson et al. [5] showed that a significant bottleneck for such systems is memory bandwidth. It was observed that

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program

978-1-4244-3504-3/09/$25.00 ©2009 IEEE

memory access to the precomputed datatype for Haar features was faster than the memory bandwidth allowed. This problem was further exacerbated by the semi-random pattern of memory accesses. This leads to poor CPU cache performance, forcing the CPU to idle until the lengthy main memory access returned.

For typical rectangle based features (such as, Haar [1], Rectangular Histograms of Oriented Gradients (RHOG) [2], and Region Covariance [7] [8]), memory access is spread out over their respective precomputed datatypes. This causes a high rate of cache misses and slow waits for memory, during which time the CPU remains idle.

Pettersson et al. [5], introduced a novel precomputed image which they called the *histogram image*. The histogram image is used in a similar way to the precomputed images for other features, but has one major advantage in terms of speed. A feature based on the histogram image requires only a single read of a 32-bit 'pixel' value for feature evaluation.

The HistFeat feature proposed by Pettersson et al. greatly reduced the memory bandwidth requirements for evaluating a single feature. However, they also reduced the processing requirements required to evaluate a single feature. This meant that even with just one access to the histogram image for each feature, memory bandwidth is still the bottleneck. To further improve such a feature would require somehow reducing the memory bandwidth required (which is difficult with just one access!) or by creating a more discriminant feature via the use of the spare processing resources.

In response to this we have developed two feature types. First is the LiteHOG feature which is designed to make use of the spare computational resources. While LiteHOG performs comparatively well on several problems, it was found that its additional computational complexity meant that instead of being memory bandwidth limited it is processor limited. The second feature, LiteHOG+, is a modified version of the LiteHOG feature space which is more *balanced* in terms of memory and processor bandwidth. LiteHOG+ is designed so that processing is done in about the same time taken to receive memory from the histogram image. Our testing of the LiteHOG+ feature shows that it is often not only a faster feature than LiteHOG but also a more discriminant one.

III. Implementation

A. The Histogram Image

The process of computing the histogram image is outlined in [5]. However, since most readers may not be familiar with its form, we provide an outline of the content of each histogram image pixel. An overview of the computations needed to produce the histogram image is shown in Figure 1. Figures 1, 2 and 3 are used with the permission of Pettersson et al. [5].

1) Orientation and magnitude binning: Any HOG feature implementation must determine how many *orientation bins* to use in the histograms and how to deal with the *magnitude* of the gradients. In the histogram image only 8 separate gradient directions are considered, see Figure 2.

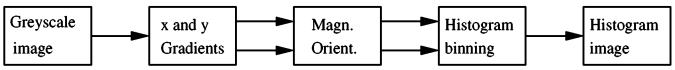

Fig. 1. Given a greyscale image, we first compute the x and y gradients at each pixel in the image. These gradients are then used to compute the magnitude and orientation at each pixel. We then use these magnitudes and orientations to create histograms at every possible 4×4 image patch. Each of these histograms is packed as a pixel in the histogram image.

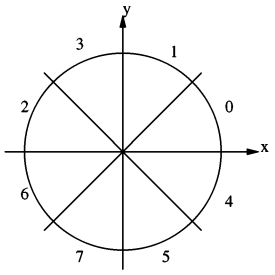

Fig. 2. The orientation space is divided into 8 bins. Each pixel in the grey image is assigned to one of these orientations. The unconventional encoding of the orientations is as defined in [5]. This encoding is designed to avoid the evaluation of slower trigonometric identities in calculating orientations.

Some HOG feature implementations add more weight to an orientation bin for stronger gradients. Within the histogram image no such weighting is performed. Rather all gradients with magnitudes larger than a given threshold are summed in the relevant orientation bin.

2) The histogram image pixel: A normal RHOG implementation [2] creates a histogram over an arbitrary rectangular region [2]. In order to concentrate all the information required to evaluate a feature into a single histogram image pixel, binning is only performed over each 4×4 image patch. Thus the maximum value for any histogram bin is just 16. By reducing the occasional value of 16 down to 15 we can capture the histogram response in just 4 bits. This reduces the total memory requirement for a single histogram image pixel to just 4×8 bits $= 32$ bits. See Figure 3.

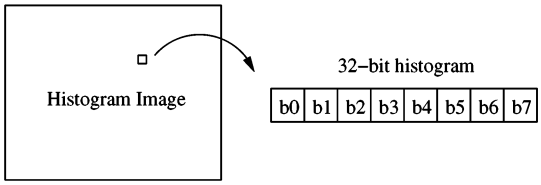

Fig. 3. Each 'pixel' in the histogram image encodes a histogram of the orientations in a 4×4 neighborhood. Each orientation is represented by just 4 bits.

B. Classifier Evaluation

The histogram image facilitates feature evaluation dependant on only a single memory access. How that feature is constructed is not predetermined by the histogram image. In the case of HistFeat [5], just 2 orientations are taken from the histogram image pixel and used to index a 2 dimensional histogram model. For the LiteHOG and LiteHOG+ features we use up to 8 orientations from the histogram image pixel but

project the N dimensional space into single scalar response. This response is used to index a model which is constructed similar to [14].

1) Finding a 1D Projection: Feature modelling and hypothesis construction is simpler and faster in 1 dimensional space than in higher dimensions. Thus we require a suitable means of dimensionality reduction. Examining several options, we have selected Fishers Linear Discriminant (FDA) [15] because it is *fast to compute* during training while producing *meaningful projections*. FDA finds an optimal projection for separating 2 Gaussian class means in a high dimensional space. While our distributions are not Gaussian we find that discriminant projections are still achieved. Other methods such as Local Fisher Discriminant Analysis [16] would probably supply an even better projection. Unfortunately, this method is too slow to compute for a reasonable portion of the feature space.

A downside of the FDA algorithm is that it produces only a single projection which is not necessarily optimal for the training data. Therefore, to create the larger LiteHOG+ feature space we find 256 possible projections using FDA on arbitrary subsets of the 8 dimensional histogram image pixel.

Apart from finding better projections, the LiteHOG+ approach gives a significant reduction in the computation required to evaluate some features. Consider the example where the most discriminant feature is found using a projection of just 3 dimensions. In this case only 3 multiplications and additions, rather than 8, are required in Equation 5. Thus this feature is both faster and more discriminant.

Linear projections are found according to the canonical variate of Fishers Linear Discriminant as shown in [15],

$$w = S_w^{-1}(m_1 - m_2) \tag{1}$$

where w is the N-dimensional projection matrix, S_w is the within class scatter matrix and m_1, m_2 are the means of the positive and negative classes respectively. For LiteHOG $N = 8$, in the case of LiteHOG+ $N \in [1, 8]$.

At this point we must deal with a 'special problem' which arises from the histogram image. A combination of the gradient magnitude thresholding (Section III-A2) and the low level of edges found in typical negative data (due to sky, road, walls etc.) means that a common bin value in the histogram image is zero for each selected dimension. That is, all N bins counted no gradients over the given threshold. Over a typical video sequence up to 40% of the histogram image 'pixels' may contain straight zeros across all 8 dimensions. This may seem an indication that the histogram image discards too much information. However, experimentation shows that gradients are still captured around objects such as signs and pedestrians, and therefore only information in regions such as sky or road are discarded. The issue for Fishers Linear Discriminant is that it is only optimal for a Gaussian distribution. The actual distribution at hand is 8-dimensional, concentrated at the origin and strictly positive. Thus, we apply Fishers Linear Discriminant only to those points which are not at the origin to make the distribution appear more Gaussian. No projection is needed for these points at the origin and we want the projection to 'focus' on the remaining data.

Let,

$$\hat{C}_i \subseteq C_i \quad \text{such that} \quad \forall x_j \in \hat{C}_i, x_j \neq \mathbf{0} \tag{2}$$

so that $\hat{C}_i$ is the subset of the class training data C_i without the points at the origin. Let the within class scatter matrix $\hat{S}_w$ be defined using the positive and negative class subsets,

$$\hat{S}_w = \sum_{n \in \hat{C}_1} (x_n - \hat{m}_1)(x_n - \hat{m}_1)^T + \sum_{n \in \hat{C}_2} (x_n - \hat{m}_2)(x_n - \hat{m}_2)^T \tag{3}$$

using the subset means $\hat{x}_1$ and $\hat{x}_2$.

This gives the final projection matrix,

$$\hat{w} = \hat{S}_w^{-1}(\hat{m}_1 - \hat{m}_2). \tag{4}$$

Once the projection is applied, the LiteHOG+ feature response can be dealt with in a manner similar to other scalar feature responses, such as Haar-features. Final weak classifier evaluation is applied as in Equation 5.

2) Creating a Model: A popular model for scalar feature responses is the simple lookup table *a posteriori* maps based on histogram statistics. These are both fast and discriminative [17] [18]. However, the benefit of improved modelling while keeping the fast lookup table approach has been shown in [19] and [14]. For Haar features this modelling approach yielded a 75% average error reduction [19]. Therefore we apply a similar Smoothed Response Binning Method to our scalar LiteHOG and LiteHOG+ feature responses.

The final weak classifier response is defined as,

$$f(x) = g(\hat{w} \cdot x) \tag{5}$$

where x contains the N selected orientations from the histogram image and $g()$ is the RealBoost weak learner classification response using the Smoothed Response Binning approach found in [19].

C. Computational Complexity of a Feature

Unlike the LiteHOG feature, LiteHOG+ does not always require 8 multiplications in Equation 5, fewer multiplications results in faster feature evaluation. In order to optimize our feature selection, we apply a simple extension to RealBoost as defined in Dollár et al. [13]. When optimizing for speed it is natural that if two features give the same error reduction that we should favor the faster one. Dollár et al. introduce the concept of a partial feature f_t' which is defined for every feature f_t with an error bound Z_t and complexity c_t. The partial feature is then defined as having an error bound of $Z_t' = Z_t^{1/c_t}$ and complexity $c_t' = 1$. They observe that selecting c_t copies of f_t' reduces the upper bound by $\prod_{t=1}^{c_t} Z_t' = Z_t$, i.e. selecting c_t copies of f_t' is the same as selecting one copy of f_t, both in terms of computational cost and the effect on the upper bound.

The final update rule selects the feature f_t with computation cost c_t and error ϵ_t which minimizes

$$Z'_t = Z_t^{1/c_t} = (2\sqrt{\epsilon_t(1-\epsilon_t)})^{1/c_t} \quad (6)$$

While it is possible to estimate the computational cost of a feature quite accurately on its own, it is not always possible to estimate the cost of adding that feature in conjunction with others. For example, if a feature f_1 has such a low computational cost that it leaves the CPU idle while waiting for memory, it is possible to add a more complex feature f_0 to be evaluated before it in order to use the 'spare' processing resources. Thus the cost of adding f_0 to the stage is not necessarily as high as initially thought. It is likely that exhaustive evaluation is the only means of determining the true cost of a feature. Since this is not possible we take the approach of merely managing the ratios of high and low complexity features using the selection rule in Equation 6. The modified RealBoost algorithm is shown in Algorithm 1.

$trainClassifiers(X, Y, F)$:
$X = \{x_1, x_2, ..., x_N\}$, the set of example windows
$Y = \{y_1, y_2, ..., y_N\}, y_i \in -1, 1$, the corresponding labels
$F = \{f_1, f_2, ..., f_M\}$, the set of features
$D_1(i) = 1/N$, the set of training weights
For $t = 1, ..., T$ (or until the desired rate is met)

1) Train classifiers h_j using distribution D_t. The classifier takes on two possible values: $h_+ = \frac{1}{2} ln\left(\frac{W_{++}}{W_{+-}}\right)$ and $h_- = \frac{1}{2} ln\left(\frac{W_{-+}}{W_{--}}\right)$ for positive and negative examples respectively. W_{pq} is the weight of the examples given the label p which have true label q.
2) Select the classifier h_t which minimizes

$$Z'_t = Z_t^{1/c_t} = \left(\sum_{i=1}^{N} D_t(i) exp(-y_i h_t(x_i))\right)^{1/c_t} \quad (7)$$

3) Update distribution $D_{t+1}(i) = \frac{D_t(i) exp(-y_i h_t(x_i))}{Z_t}$

The final strong classifier (cascade stage) is

$$H(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right) \quad (8)$$

Alg. 1: The modified RealBoost Algorithm

IV. Experiments

In this section, we perform a set of experiments to evaluate the speed and classification tradeoffs of different features, particularly the LiteHOG and LiteHOG+ features.

These experiments provide a per feature comparison of:

- Strong classifier performance given a range of CPU time budgets. This allows us to select the fastest feature for a desired classification performance.
- The number of features that can be evaluated per second in video data.

Comparisons are performed on five different feature types. Our new LiteHOG and LiteHOG+ features, its predecessor HistFeat, as well as the Haar and RHOG features. HistFeat is implemented as in [5]. Haar features are implemented in the classical fashion but we use the improved smoothed learning as in [14]. RHOG features are implemented in a similar fashion to the Rectangular HOG features in [2]. Our native version of this feature differs slightly in its orientation binning scheme, descriptor block arrangement, and discards the block normalization. These differences are generally either required to fit within our boosting implementation or give a greater emphasis on speed.

Significant effort has been put into the speed optimization of all features compared, down to the assembly level. I.e. we are not comparing our speed optimized features with weaker/slower implementations of the other features. Features are optimized by producing two implementations of each feature. One for training purposes and another highly optimized version for run-time evaluation. This second version allows the rearrangement of features so as to collocate features which will reference similar (x, y) coordinates in their precomputed datatype.

A. Time-Constrained Strong Classifier Performance

We employ the same experimental evaluation technique as in [5]. That is, we train *single stage classifiers* instead of *cascade structured classifiers*. Cascade classifiers are highly sensitive to small changes in rejection thresholds making reasonable comparisons between methods difficult. Of course, single stage classifiers do not achieve the best possible classification or speed performance but we believe that the resulting comparisons are more reliable.

Fig. 4. Example images of different types of road signs in our dataset.

Fig. 5. A selection of 32x80 color images from the NICTA Pedestrian Dataset.

1) Dataset: Positive input data to the RealBoost training process for the different object types are listed in Table I. The numbers reflect the original raw hand-labelled images taken by a digital camera, and the number of images after distortions have been applied as in [5]. Pedestrian images are a subset taken from the NICTA Pedestrian Dataset [20]. As such, the only distortion applied is mirroring. A total of 10,000 negative training data and 100,000 validation data were used in all road sign experiments.

2) Training and evaluation: For each object type listed in Table I, RealBoost was used to train 1000 different strong classifiers consisting of 1 to 1000 features. For each of the strong classifiers we calculated the *ROC curves* and the *scan time* on a typical video sequence.

By pairing the ROC and scan time data we produced the time-error curves shown in Figure 6. To give a clearer

TABLE I
POSITIVE INPUT DATA, RAW IMAGES FROM A DIGITAL CAMERA AND DISTORTED IMAGES GENERATED FROM THE ORIGINAL RAW IMAGES.

Type	Raw Train	Raw Valid	Dist Train	Dist Valid
Speed Sign	1110	222	10000	2500
Stop Sign	129	32	10000	2500
Give Way Sign	154	38	10000	2500
Pedestrian	4926	1865	9852	1865

representation of the resulting error versus time tradeoffs we sample the ROC curve at 3 operation points; a) *the 'knee'*, see below b) *false negative rate = 0.01*; and c) *false positive rate = 0.01*.

The minimum total error E on the ROC curve is taken to be at the 'knee' of the curve, where the ROC curve is closest to perfect classification.

$$E = \min(\sqrt{F_p^2 + F_n^2}) \tag{9}$$

where E is the total error, F_p is the false positive rate and F_n is the false negative rate.

In Figure 6, we note several interesting results.

No single feature dominates an entire performance curve: This is apparent in all three time-error plots in Figure 6. The LiteHOG and LiteHOG+ features are no exception. We believe this is a strong justification for this kind of performance versus computing time analysis.

Haar features and RHOG features dominate the curves for slower classification speeds: This is particularly apparent in Figures 6(a) and 6(b) for pedestrians and speed signs. So for applications where detection accuracy is preferred over speed Haar and RHOG features are probably more suitable choices than the new LiteHOG and LiteHOG+ features. Additionally, one might consider the Region Covariance feature [7] which is known to give particularly impressive detection results for pedestrians.

HistFeat is the best performer for very fast classifiers: HistFeat is the fastest of all the features on a per feature basis and therefore can be expected to dominate at the fastest end of the curves. Each individual application will have its own desired speed requirements. We note that many useful applications such as in-vehicle sign and pedestrian detection become plausible with a low latency processing of standard definition video [1] frames at a frequency of around 10-200Hz (5-100ms/frame). This is the region highlighted in yellow in Figure 6. Evaluation slower than 100ms introduces a latency at which signs and pedestrians can not be detected at normal driving speeds. Evaluation speeds of up to 200Hz can be useful if multiple detectors are to run on a single architecture.

LiteHOG+ pedestrian classifier dominates at speeds of 20-1000ms: See Figure 6(a). For much of the highlighted speed range LiteHOG+ is 2-4 times faster than a Haar or RHOG classifier of equivalent error-rate performance. Therefore, even if higher accuracy is required it may be desirable to use LiteHOG+ in the early stages of the cascade to avoid the evaluation of slower Haar or RHOG features on every incoming image patch.

HistFeat, LiteHOG and LiteHOG+ features overfit on speed sign data: Results on the speed sign classifiers, Figure 6(b), show significant overfitting for some features. We see that these features do suffer some overtraining effects at which point learning stalls and classification performance actually gets worse. However, LiteHOG+ is still an outstanding performer for fast classifiers. This suggests using it in a prefilter stage or perhaps attempting to prevent the overtraining through the use of a larger and more diverse training set. We note, however, that the low number of validation samples at such high accuracies means that validation is somewhat noisy. Therefore this portion of the road sign results is not entirely conclusive.

HistFeat, LiteHOG and LiteHOG+ features build significantly faster classifiers on stop sign data: Figure 6(c), shows huge speed gains on stop sign data. These features are almost an order of magnitude faster than RHOG features and sometimes two orders of magnitude faster than Haar features! This suggests than gradients were a more useful low level feature on the somewhat homogeneous stop sign dataset (unlike speed signs which may have different speed values printed on the face). Haar features do not capture such gradients as effectively.

B. Features Per Second

Equation 10 defines the total time spent per frame of video. This is dependant on the preprocessing time for a single frame of video T_p, the average number of features computed in each image window f_{ave}, the total number of inspected windows per frame N, and the evaluation time per feature T_e.

$$T = T_p + f_{ave} \cdot N \cdot T_e \tag{10}$$

Figure 7 shows the number of features which can be calculated per second for different classifier stage lengths. LiteHOG+ is clearly faster to evaluate than the LiteHOG feature. Since the LiteHOG+ feature also provides significantly more discriminant features for pedestrians this speed gain is particularly pleasing. I.e. the LiteHOG+ features learns a better classifier in fewer features and the features themselves are faster to compute! For very short classifiers (less than 10 features) the processor is able to cache all the memory required to evaluate the classifier, this means that features with very low computational demands, such as HistFeat, evaluate many times faster. Conversely, larger classifiers justify the use of a feature such as LiteHOG+ which balances the wait for memory with extended processor use.

C. A Heterogeneous Classifier

In order to produce a set of classifiers with performance comparable to the best of either HistFeat or LiteHOG+ in Figure 6(a), we trained a set of combination classifiers, with

[1]**Standard Definition Video;** No such single standard exists! Generally with resolutions of around 704×480 to 768×576. This paper uses Standard Definition PAL at 768×576.

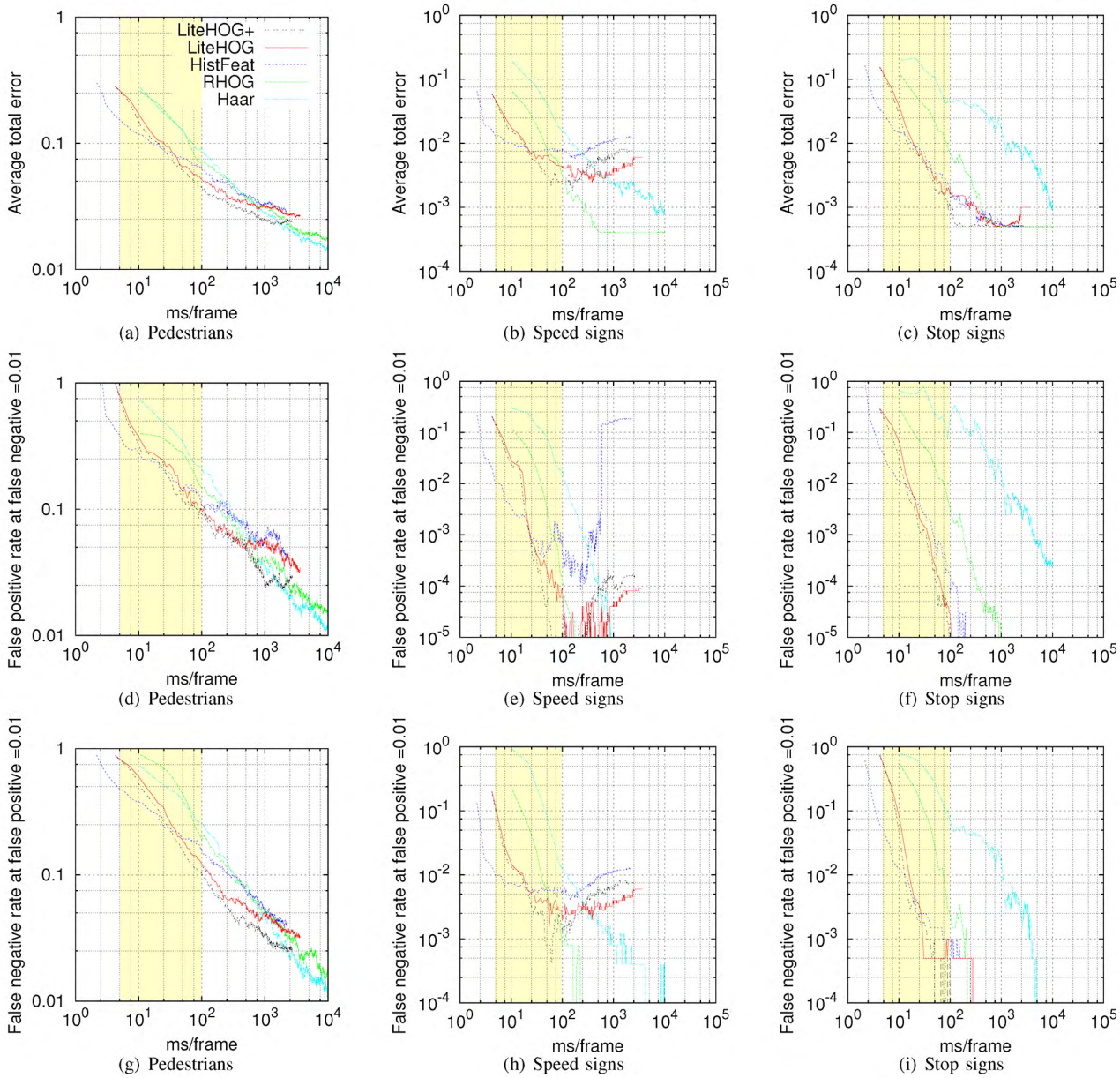

Fig. 6. Performance versus computing time. The top row shows average total error versus computing time for the different object classes. The middle row shows the performance for a fixed negative rate of 0.01. The last row shows performance for a fixed false positive rate of 0.01. Note that the pedestrian column has a different scale on the y-axis. In these plots we see that LiteHOG+ is well suited to fast pedestrian detection problems. While it does not dominate all of the time-error plots it is often the best feature for much of the target range (yellow region), 5-100ms/frame.

up to 50 features, using both HistFeat and LiteHOG+ features. Figure 7 includes the features per second performance of these classifiers. The resulting classifiers clearly evaluate at only slightly slower speeds than the pure HistFeat classifiers. Figure 8 shows the performance of the combined classifier on the pedestrian data. For slower computing times (more than 11ms) the classifiers are similar but slightly worse than pure LiteHOG+. This is probably due to selection bias toward the overfitting HistFeat features, a problem discussed in [21]. For faster classifier speeds the results are comparable to the best of either HistFeat or LiteHOG+.

V. Conclusion

In this paper, two novel features LiteHOG and LiteHOG+ were presented. Both have been shown to be extremely fast and discriminant features, particularly suited to low latency pedestrian detection problems. The expanded feature set LiteHOG+ is generally the dominant feature on most detection tasks. The features are suitable for use in a cascade of boosted weak classifiers, but may also be useful within other machine learning frameworks.

Experiments on real world video data provides a comparison of the LiteHOG and LiteHOG+ features to HistFeat, a

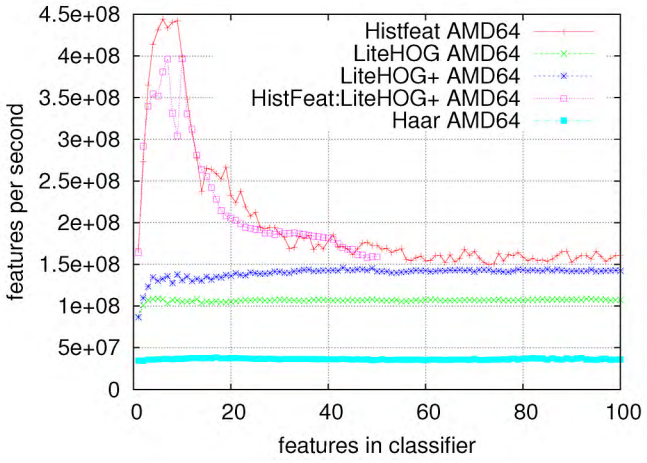

Fig. 7. The number of features that can be evaluated per second, on an AMD64 2.2GHz machine, for LiteHOG+, LiteHOG, HistFeat, and the Haar features. Measurements were done on a video with 101 frames and a resolution of 768×576. The (x, y) step was $(1, 1)$, which results in about 365,000 patches per frame. For larger classifiers LiteHOG+ is only slightly slower per feature than HistFeat. This indicates that LiteHOG+ is only weakly processor limited. The LiteHOG feature is significantly slower than HistFeat the still much faster than Haar features. This shows that LiteHOG is a processor limited feature.

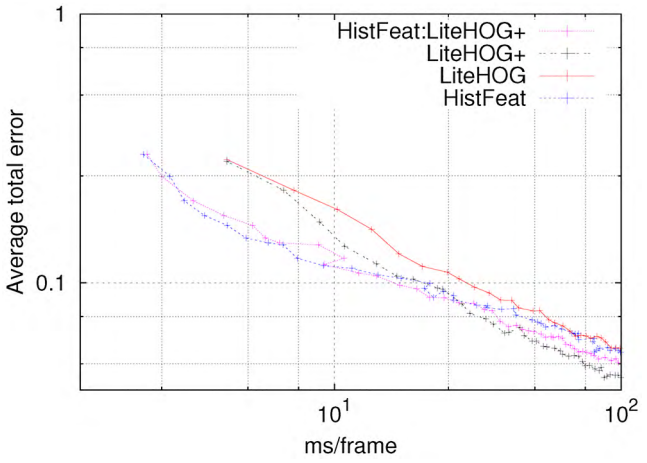

Fig. 8. Pedestrians: Performance vs computing time. The combined classifiers using HistFeat and LiteHOG+ has a competitive performance over a range of speeds though it does not dominate at any point.

Rectangular Histogram of Oriented Gradients feature (RHOG) and Haar features. The experiments shown provide guidance to researches seeking a suitable feature for various time constrained detection tasks.

Detailed analysis has been given, outlining the behavior of various feature types on typical PC architectures. A method for balancing processing and memory bandwidth has been given. This makes use of a simple change to the AdaBoost or RealBoost feature selection algorithm. This change means that selections consider the computational complexity of prospective feature selections. Its usefulness was demonstrated in a classifier consisting of LiteHOG+ and HistFeat features.

REFERENCES

[1] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *IEEE Computer Society Conference On Computer Vision And Pattern Recognition*, vol. 1. IEEE Computer Society; 1999, 2001.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Computer Vision and Pattern Recognition*, vol. 01, pp. 886–893, 2005.

[3] Q. Zhu, S. Avidan, M. Yeh, and K. Cheng, "Fast human detection using a cascade of histograms of oriented gradients," in *IEEE Computer Society Conference on Computer vision and Pattern Recognition*, vol. 2, 2006, pp. 1491–1498.

[4] B. Alefs, G. Eschemann, H. Ramoser, and C. Beleznai, "Road sign detection from edge orientation histograms," in *IEEE Intelligent Vehicles Symposium (IV2007)*, June 2007.

[5] N. Pettersson, L. Petersson, and L. Andersson, "The histogram feature – a resource-efficient weak classifier," in *IEEE Intelligent Vehicles Symposium (IV2008)*, June 2008.

[6] B. Wu and R. Nevatia, "Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors," *International Journal of Computer Vision*, 2007.

[7] O. Tuzel, F. Porikli, and P. Meer, "Human detection via classification on riemannian manifolds." in *CVPR*. IEEE Computer Society, 2007.

[8] S. Paisitkriangkrai, C. Shen, and J. Zhang, "Fast pedestrian detection using a cascade of boosted covariance features," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 8, pp. 1140–1151, August 2008.

[9] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, pp. 771–780, sept 1999.

[10] R. E. Schapire and Y. Singer, "Improved boosting using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999. [Online]. Available: citeseer.ist.psu.edu/article/singer99improved.html

[11] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *The Annals of Statistics*, vol. 38, no. 2, pp. 337–374, 2000.

[12] J. Sochman and J. Matas, "Waldboost - learning for time constrained sequential detection," *Computer Vision and Pattern Recognition*, vol. 2, pp. 150– 156, 2005.

[13] P. Dollár, Z. Tu, H. Tao, and S. Belongie, "Feature Mining for Image Classification," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, 2007, pp. 1–8.

[14] G. Overett and L. Petersson, "Improved response modelling on weak classifiers for boosting," *IEEE International Conference on Robotics and Automation*, 2007.

[15] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0471056693

[16] M. Sugiyama, "Local Fisher discriminant analysis for supervised dimensionality reduction," in *Proceedings of the 23rd international conference on Machine learning*. ACM New York, NY, USA, 2006, pp. 905–912.

[17] B. Rasolzadeh, L. Petersson, and N. Pettersson, "Response binning: Improved weak classifiers for boosting," in *IEEE Intelligent Vehicles Symposium (IV2006)*, June 2006.

[18] B. Wu, H. Ai, C. Huang, and S. Lao, "Fast rotation invariant multi-view face detection based on real adaboost," *IEEE International Conference on Automatic Face and Gesture Recognition*, 2004.

[19] G. Overett and L. Petersson, "On the importance of accurate weak classifier learning for boosted weak classifiers," in *IEEE Intelligent Vehicles Symposium (IV2008)*, June 2008.

[20] G. Overett, L. Petersson, N. Brewer, L. Andersson, and N. Pettersson, "A new pedestrian dataset for supervised learning," in *IEEE Intelligent Vehicles Symposium (IV2008)*, June 2008.

[21] G. Overett and L. Petersson, "Boosting with multiple classifier families," *IEEE Intelligent Vehicle Symposium*, 2007.