Fast Features for Time Constrained Object Detection

Gary Overett and Lars Petersson*
NICTA
Locked Bag 8001, Canberra, Australia
gary.overett@nicta.com.au

Abstract

This paper concerns itself with the development and design of fast features suitable for time constrained object detection. Primarily we consider three aspects of feature design; the form of the precomputed datatype (e.g. the integral image), the form of the features themselves (i.e. the measurements made of an image), and the models/weak-learners used to construct weak classifiers (class, non-class statistics). The paper is laid out as a guide to feature designers, demonstrating how appropriate choices in combining the above three characteristics can prevent bottlenecks in the run-time evaluation of classifiers. This leads to reductions in the computational time of the features themselves and, by providing more discriminant features, reductions in the time taken to reach specific classification error rates.

Results are compared using variants of the well known Haar-like feature types, Rectangular Histogram of Oriented Gradient (RHOG) features and a special set of Histogram of Oriented Gradient features which are highly optimized for speed. Experimental results suggest the adoption of this set of features for time-critical applications.

Time-constrained comparisons are presented using pedestrian and road sign detection problems. Comparison results are presented on time-error plots, which are a replacement of the traditional ROC performance curves.

1. Introduction

In many computer vision problems such as detection, both *error rates* and *time to decision* determine the value of a given solution. Indeed, the stellar success of the Haar-feature, presented by Viola & Jones [16], can be primarily attributed to the fast-yet-discriminant nature of the Haar-feature (via the integral image) and its successful combination in a boosted cascade. The majority of detection feature comparisons found in literature consider mostly error-rate performance. This paper, however, compares features from a time-constrained perspective and introduces an analysis of several recent and emerging feature designs for extremely fast-yet-discriminant features.

For all detection systems adopting a Viola & Jones like framework, we see that 3 major avenues exist for improving speed and robustness. Firstly, it is possible to change the *choice of weak classifier*. Secondly, it is possible to use an alternative *choice of boosting algorithm* (or other learning algorithm) and thirdly it is possible to reform the *cascade structure*. This paper concerns itself with the design of the weak classifier, with the aim of providing improvement in the first area, that of the *choice of weak classifier*. However, for completeness, it is first worth briefly considering the other two.

The chosen boosting algorithm can have a marked effect on both the speed and classification performance of the resulting strong classifier. AdaBoost [4], and its Real-Valued counterpart RealBoost [13], are favoured options, however, other boosting algorithms may construct a faster or more discriminant classifier. For example, on some problems, researchers have found LogitBoost [5] to decrease the classification error more quickly [15].

Detection cascades are generally structured so that small and fast early stages reject a large amount of the incoming data. However, making appropriate early decisions after just a few features have 'measured' the input data is difficult. Nevertheless, it is an important factor in the speed-robustness trade off. Cascade structures are often determined manually using basic rules of thumb, or by requiring stages to meet a given performance criteria as in [15]. It is often appropriate to have the boosting algorithm itself take on the task of constructing the cascade structure so as to deal with both problems together. A good example is found in the WaldBoost algorithm [14].

The criteria for determining a suitable *choice of weak classifier* are similar to those for the overall strong classifier system. That is, quality is determined by both error rates (per feature discriminance) and time to decision

*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

(computational complexity). Within the literature, Haar-features [16], Histograms of Oriented Gradients (HOG features) [2][19], Edgelet features [18] and Region Covariance features [15] [10] are all popular choices. Each approach yields a range of performance results. Different object classes, such as pedestrians, faces, or road signs, often suggest a different choice of weak classifier. Features will be chosen for their characteristics like scale invariance, rotational invariance and perspective invariance. While all these features have been implemented using precomputed datatypes, such as the the integral image [16], they are quite different in terms of speed. The Region Covariance feature is, for example, many times slower than a basic Haar-feature implementation, but, for pedestrians at least, it is also many times more discriminant [15].

While many of these features are considered cheap to compute, some require substantial bandwidth to memory to support real-time computation in large high-resolution images. In Pettersson et al. [11], a new feature, HistFeat, is presented. HistFeat is an extremely resource efficient feature, depending on a single main memory access for feature evaluation. Like Haar features, Rectangular Histogram of Oriented Gradient (RHOG) features and Region Covariance features, HistFeat uses its own precomputed datastructure, the Histogram Image. This contains a highly packed form of Histograms of Oriented Gradients. More recently Overett and Petersson [8] presented two more features based on the histogram image, the LiteHOG and LiteHOG+ feature. These preserve the single memory access feature evaluation of HistFeat but use more of the data from within the histogram image 'pixel' and employ a more powerful weak learner based on more advanced non-parametric models [7]. The result was a feature set that was slower on a per feature basis but significantly more discriminant. The added discriminance of the LiteHOG+ feature means that it could reach a lower error rate than HistFeat in fewer features and less time.

In this paper we create several new feature designs by pairing the different features with alternate weak learners to compare the implications for evaluation speed and discriminance. Readers are most likely to be aware of per-feature speed and per-feature discriminance comparisons. E.g. one region covariance feature [15] is more discriminant than one RHOG feature. This fails to consider that it may be possible to compute *three* RHOG features in the same time as *one* region covariance feature. In this paper we perform comparisons using a time-error analysis, comparing the error performance achieved in a fixed amount of time.

At this time it is worth including a somewhat philosophical note about comparing features in the boosting context. In a climate where the primary quantitative measure of feature strength is the ROC curve (or some equivalent curve based on error-rates), there has been a growing presence of more *computationally expensive* and *memory intensive* features without considering time-to-decision. In pedestrian detection, a number of works have asserted their superiority based purely on decreased error rates. Under error rate evaluation on pedestrians, past work has found that Region Covariance [15] improves upon HOG features [2], which in turn improved upon Haar-features. However, the memory and processing requirements of the Region Covariance feature are staggering, and it is doubtful that this feature can produce real-time performance on standard PC hardware in the near future. One should remember, that the astronomical popularity of these methods in the first place is borne out of the somewhat more qualitative results shown, where tasks such as face detection were demonstrated running in real-time on normal hardware.

That said, this author denotes high value to both the Region Covariance feature and the RHOG feature, having implemented versions of both. Indeed the Histogram Image based features in this paper are effectively child and grandchild features of the RHOG feature. They represent an attempt to distill the greatest part of the HOG 'signal' into a faster variant, see Section 3.4. Further work will hopefully, succeed in doing the same for ever more powerful features, such as the Region Covariance Feature [15].

2. The Work of Feature Evaluation

Typical feature evaluation can be broken up into three parts: the *precomputed datatype* (if any), the *feature response computation* (the result of measuring the image), and the *model evaluation* (converting the feature response into a probabilistic statement about class instances in an image). See Figure 1.

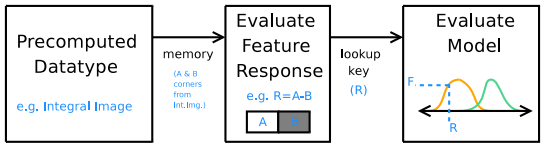

Figure 1. Memory and computational flow in evaluating a feature. The major bottlenecks for full feature evaluation are usually memory bandwidth to the precomputed datatype, computation required to evaluate the feature and/or model evaluation.

2.1. Precomputed Datatype

Pettersson et al.[11] observed that a major bottleneck limiting the speed of most visual detection algorithms is not the speed of the CPU, but rather the bandwidth to the memory. This is because precomputed datatypes such as the integral image [16] have drastically reduced the per-feature computation required to evaluate features. The task of reading the precomputed datatype is often made worse because detection algorithms typically have a semi-random way of

accessing memory. This leads to poor CPU cache performance, causing the processor to idle until the lengthy main memory access returns.

Consider the example of any of the rectangle based features; Haar, RHOG, or Region Covariance. Typical computing architectures will stream consecutive memory regions with great efficiency, however, the above features require memory access spread out over their respective precomputed datatypes. This causes a high rate of cache misses, forcing the CPU to idle.

Pettersson et al. [11], introduce a solution to this problem via a novel precomputed image which they called the *histogram image*. This is an "image" where each 32-bit word represents a Histogram of Orientations of a 4×4 pixel area in the input image. The histogram image is used in a similar way to the precomputed images for other features, but has a major advantage in terms of speed. A feature based on the histogram image requires only a single read of a 32-bit word for feature evaluation, as opposed to several semi-random reads per feature.

In this paper we will consider several features using the histogram image. Including HistFeat (as in [11]), the LiteHOG and LiteHOG+ features (as in [8]) as well as some new variants of these features using decision stump learners, see Sections 2.2.3 and 2.3.

2.1.1 Computing the Histogram Image

Most readers will not be familiar with the structure of the Histogram Image, thus we will include a description here. The major aim of the histogram image is to produce a precomputed datatype allowing single memory read feature evaluation in tandem with high per-feature discriminance. Figures 2, 3 and 4 are used with permission of Pettersson et al.[11].

An overview of the computations needed to produce the histogram image is shown in Figure 2.

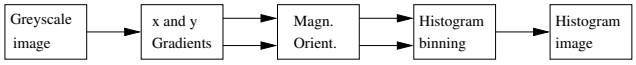

Figure 2. Given a greyscale image, first compute the x and y gradients at each pixel in the image. These gradients are then used to compute the magnitude and orientation at each pixel. Then use these magnitudes and orientations to create histograms at every possible 4×4 image patch. Each of these histograms then becomes a pixel in the histogram image.

Computing the gradients

The x and y gradients of a greyscale image are calculated using a $[-1, 0, 1]$ kernel. This kernel choice derives support from [2], which found it to be the most discriminant. Since it is also easy to implement using SSE2 technology, this was the method adopted.

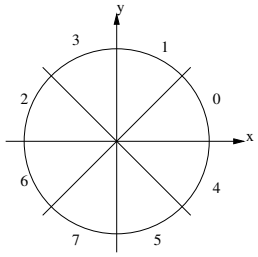

Figure 3. The orientation space is divided into 8 bins. Each pixel in the grey image is assigned to one of these orientations.

Orientation and magnitude

Following the x and y gradient computation, the gradient orientations and magnitudes at each pixel are calculated. A pixel's orientation is represented by a number between zero and seven. Hence, it can be stored as a 3 bit value.

Reducing the number of orientations to just eight means that orientation can be found via a sequence of simple comparisons. More specifically,

$$ori_{x,y} = (G_y < 0)\cdot 4 + (G_x < 0)\cdot 2 + (|G_y| > |G_x|)\cdot 1 \quad (1)$$

where $ori_{x,y}$ is the orientation at pixel (x, y), G_x is the x gradient at pixel (x, y) and G_y is the y gradient at pixel (x, y).

This results in the unconventional encoding of the different orientations shown in Figure 3. This encoding avoids any evaluation of the much slower trigonometric identities (further discussion in Section 3.4).

While calculating the orientation associated with each pixel, a simultaneous calculation of the pixel's approximate magnitude is found by taking the absolute sum of the two gradients G_x and G_y. This use of the L1 metric instead of the Euclidean distance is far more efficient and has been found to have an entirely negligible effect on discriminance. This value is immediately thresholded to give a 1 bit representation of magnitude. This is an important step in compressing the data of the histogram image, but note the discussion in Section 3.4 on discriminative loss.

Histogram binning

Next, a histogram for each 4×4 image patch is found by binning the gradients in the patch to the 8 directions shown in Figure 3. Gradients whose magnitude is less than some chosen threshold are ignored (magnitudes of 8/256 in this paper). Each of these histograms then becomes a pixel in the histogram image.

Pettersson et al. [11] chose to represent each histogram as a 32 bit word (i.e. four bits for each orientation) as shown in Figure 4. This allows a full histogram to be read in a single access to memory. The overall storage needed for the histogram image is the same as for the integral image - 32 bits per pixel in the input image.

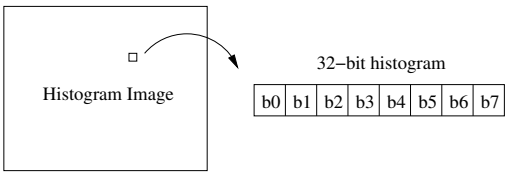

Figure 4. Each 'pixel' in the histogram image encodes a histogram of the orientations in a 4×4 neighborhood. Each orientation is represented by just 4 bits.

2.2. Computing the Feature Response

2.2.1 Haar and RHOG features

The task of computing the feature response of Haar-features and RHOG features is generally well known and can be found in [16] and [2]. The important attributes from our perspective are that their feature evaluation relies on several reads of their respective precomputed datatypes and that the evaluation itself involves only a few very fast computations. The result of this computation is the lookup key for the model.

2.2.2 HistFeat

Evaluation of the HistFeat feature can be found in [11]. It involves taking two orientation bins from a single histogram image pixel. These two values serve as the lookup key for the model with no alteration at all. An extremely brief feature evaluation indeed! Essentially this requires no computation at all, the lookup key is simply found in the 32-bit histogram image pixel.

2.2.3 LiteHOG+ and Feature Variants

Our results include two feature variants based on the histogram image; LiteHOG+, and LiteHOG [8]. Both variants involve finding a linear projection of the 8 orientation bins of the single histogram image pixel or, in the case of LiteHOG+, some subset of the 8 dimensions.

Linear projections are found according to the canonical variate of Fishers Linear Discriminant (FDA) as shown in [3],

$$w = S_w^{-1}(m_1 - m_2) \tag{2}$$

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+ $N \in [1, 8]$. Note, also that the boosting algorithm RealBoost [13] produces a weights vector, which is used in the calculation of the projection matrix w. I.e. we are using *weighted* FDA. This means that new projections are found for each RealBoost training round, uncovering new information from the same histogram image pixel according to the RealBoost weighted priorities.

Thus the evaluation of the LiteHOG+ feature involves N multiplications, and $N - 1$ additions to compute the projection, see Equation 3.

$$R = w \cdot x \tag{3}$$

where x contains the N selected bins from the histogram image. Since the LiteHOG feature space always requires an 8 dimensional projection, it is slower to compute than the LiteHOG+ feature space which may drop several dimensions from the computation. Additionally, the projections within the LiteHOG+ feature space are often more discriminant, making LiteHOG+ both faster to compute and more powerful on a per feature basis. Further details can be found in [8].

2.3. Model Learning and Evaluation

For the feature response to hold meaning about the class/non-class likelihood of an image patch we require a model of the class/non-class feature response statistics. For a feature designer, models can be divided into two aspects; the *model learning*, which allows for significant effort as it will take place in an offline context, and *model evaluation* which will form a part of the online system which is time-constrained. Thus, while we can be more extravagant in our learning processes the model must be able to be stored and referenced in such a way as to perform well in *the context of the other design choices*. For example, there is no reason to adopt a model with low processing overhead if the feature evaluation is memory bandwidth limited. In this case the CPU may be left idle - waiting on memory. This *'spare'* processing resource can be used in the evaluation of a *more detailed, more discriminant model*, if one can be found.

From an evaluation perspective our experiments consider two kinds of models; decision stumps and lookup table (LUT) based models. From a learning perspective we consider three different model learners; stump learners, histograms (LUT based) (as in [12] [17]) and smoothed response binning (SRB) learners as found in [7] and [6] which are also LUT based. SRB learners are non-parametric density estimation models which support multimodal data and where the final density estimation is produced for a fast LUT based model evaluation. Figure 5 shows a comparison of the SRB learner with the decision stump learner.

A prevalent idea in boosting theory is that the weak learners need only to be *slightly better than random* to yield a robust result in the final strong classifier. The computer vision community has largely adopted simple models such as decision stumps following the landmark paper of Viola and Jones [16]. We find that few researchers employ more advanced learners, despite the fact that they can increase

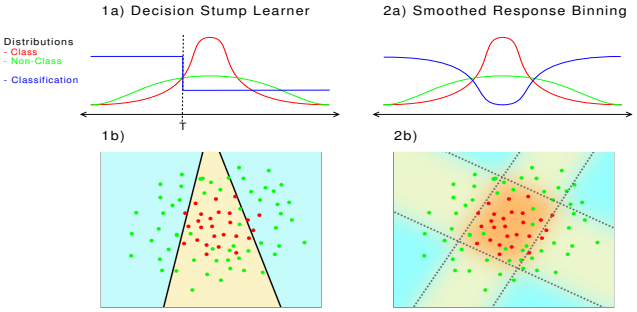

Figure 5. Comparison of the SRB model learner [6] with the decision stump learners. The top row shows how the two methods map the distribution of class data (red and green) to a learnt hypothesis (blue). The bottom row shows how SRB learners are able to discriminate more quickly on a 2D 'toy' distribution. The SRB model greatly improves the per-feature discriminance in the same evaluation time as other lookup table based methods such as histograms. Histograms suffer from severe overfitting in sparse regions leading to poorer boosting-level learning and weaker features. Per-feature error is often reduced by 75% when SRB models replace histogram learners [7].

Name	FeatSpace	Datatype	Learner
RHOG-SRB	RHOG	RHOG Integral Img.	SRB [6]
RHOG-ST	RHOG	RHOG Integral Img.	Stump
Haar-SRB	Haar	Integral Img.	SRB [6]
Haar-ST	Haar	Integral Img.	Stump
HistFeat	HistFeat	Histogram Img.	2D-Histogram
LiteHOG-SRB	LiteHOG	Histogram Img.	SRB [6]
LiteHOG+ SRB	LiteHOG	Histogram Img.	SRB [6]
LiteHOG+ ST	LiteHOG	Histogram Img.	Stump

Table 1. Features compared in our experiments.

the discriminance of individual features and thereby improve both time-to-decision performance and classification robustness. Buja & Stuetzle [1] however, reminds us that *"the base learners [at the time boosting rose to fame (circa 1998)] were anything but weak in the sense of todays orthodoxy [decision stubs etc.]"*. I.e. early boosting algorithm development considered somewhat more advanced learners.

2.4. Feature Summary

Table 1 shows a summary of the features used in our experimental comparisons. Figure 6 shows the evaluation speed of the different feature types.

2.4.1 Feature Optimization

Occasionally, when computer vision researchers compare methods they create a kind of "home advantage" for a given method by affording it a greater level of optimization than they give other methods. This is *not* the case here. We have applied significant optimization to all features compared, right down to the assembly level and memory arrangements

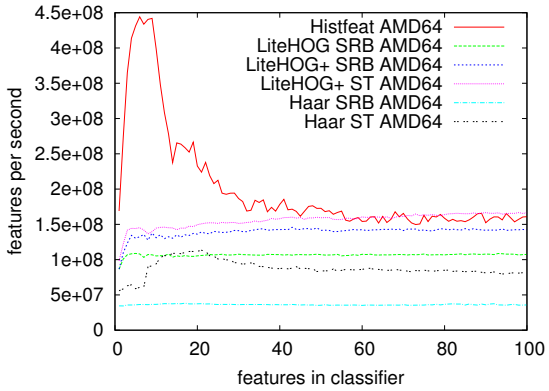

Figure 6. The number of features that can be evaluated per second, on an AMD64 2.2GHz machine. Measurements were done on a typical test video with 101 frames and a resolution of 768×576. The (x, y) step was $(1, 1)$, which results in about 365,000 patches per frame.

of each feature. For example, our Haar feature implementation requires only 40 CPU instructions. Viola & Jones [16] reported that their implementation of the Haar feature required about 60 CPU instructions.

3. Time-Error Experiments

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.

Strong classifiers can be trained in a variety of ways. Most often, training is done in a cascaded fashion with a number of stages one after the other, each rejecting a proportion of the input data. This is a good procedure when trying to achieve the best possible classifier for a particular application. Unfortunately, this procedure excludes the possibility of a fair comparison between methods. Cascade structures are highly sensitive to changes in their structure and this affects different features in markedly different ways. Boosting methods requiring each stage to meet a given false positive and false negative rate, as in [15], are suitable for comparing minimum error rates at feature starvation. However, changing the required cascade stage performance between say a 35% and a 50% reject rate may yield similar results in terms of error rates but will yield entirely different 'winners' under a time constrained analysis. Hence, to make a fair comparison of the features, only a single stage was used.

Detection cascade design is a challenging task when time-to-decision is a factor. Some supporting theory can be found in [14]. Generally however, it is a problem which would require exhaustive exploration of the cascade-space (the space of cascades with all possible between stage thresholds and stage size combinations). This is impossible.

Type	Raw Train	Raw Valid	Dist Train	Dist Valid
Stop Sign	129	32	10000	2500
Pedestrian	4926	1865	9852	1865

Table 2. Number of images used as positive training and validation data. Distortions are applied to the raw images to create a larger, more generalized dataset.

3.1. Training and Evaluation Dataset

Figure 7. A selection of 32x80 color images from the NICTA Pedestrian Dataset.

Positive input data to the RealBoost training process for the different object types are listed in Table 3.1. 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 [11]. Pedestrian images are a subset taken from the NICTA Pedestrian Dataset [9]. As such, the only distortion applied is mirroring. A total of 10,000 negative training data and 100,000 validation data were used in the stop sign experiments.

3.2. Training and evaluation parameters

For each object type listed in Table 3.1, RealBoost was used to build 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 9. To give a clearer representation of the resulting error versus time trade offs 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*. These measures provide a better description of the shape of the ROC curve than other measures such as the AdaBoost error score or the common Area Under the Curve value.

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.

$$E = \min(\sqrt{F_p^2 + F_n^2}) \tag{4}$$

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

In Figure 9, we note several interesting results. It is apparent from the results that *no single feature dominates an entire performance curve*. We believe this is a strong justification for this kind of performance versus computing time analysis.

3.3. Feature Design Guide

In this Section we will discuss the experimental results from Figures 9 and 8.

To get the most use from the time-error plots, one must first consider the time constraints of the problem at hand. We find that classifier evaluation in the range of 10-200Hz is most useful. Exhaustively searching for objects in a reasonably spaced scale-space pyramid on standard definition video [1] yields around 365,000 patches per frame. See highlighted region Figures 8 and 9.

HistFeat, the fastest feature to decrease error, would be recommended, for very easy problems where acceptable error-rates are achieved with few features. Easy to intermediate detection problems are best for the LiteHOG+ feature using SRB learners. The Haar and RHOG feature spaces provide features geared for robustness. However, they will not produce real-time results on more difficult detection problems, if high detection rates and low false positive rates are required.

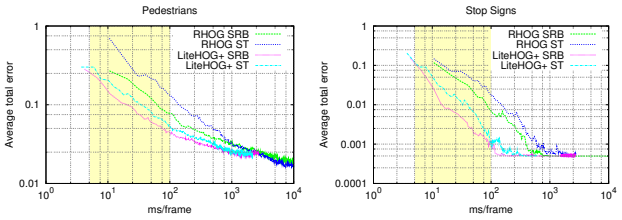

Figure 8. Comparison of Decision Stump and SRB learners. The SRB learners are superior for both LiteHOG+ and RHOG features. Note, the superiority weakens significantly for the more discriminant (on a per feature basis) RHOG feature in conjunction with the less challenging stop sign class (right hand image). We suggest two reasons for this. Highly discriminant features working on less challenging classes give rise to *strongly divergent distributions* of the class/non-class responses. When distributions are well separated, decision stumps perform similarly to more advanced models. In contrast, weaker features and harder problems yield overlapping distributions requiring more detailed models to discriminate the class/non-class statistics.

3.4. Simplification, Speed and Discriminance

Most feature simplifications involve at least some loss of discriminance in order to improve the speed of the feature. Therefore, it is worth providing some discussion of the relative merits of different design choices. As can be seen in Figure 9, RHOG - the grandparent feature of LiteHOG+, is often more discriminant given more computing time. It is worth noting where this loss of discriminance happens in an evolution from RHOG features to the various features using the histogram image.

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

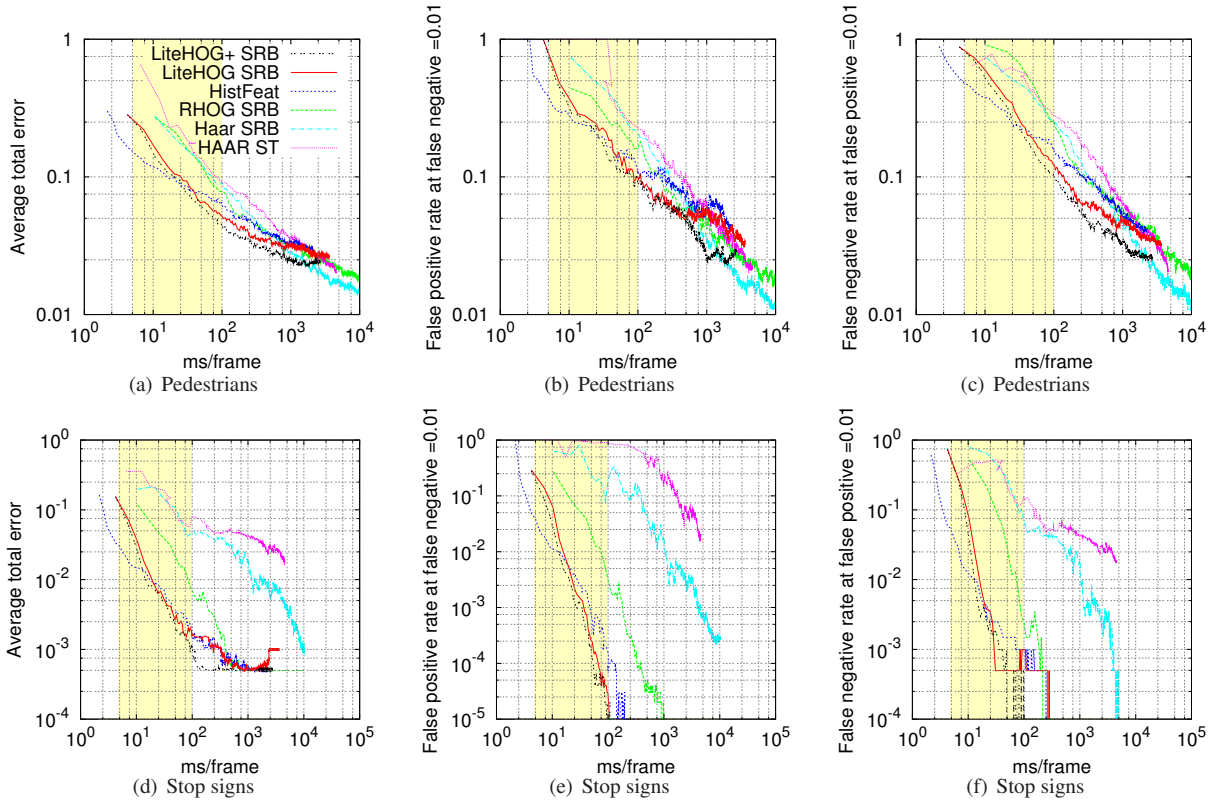

Figure 9. Performance versus computing time. The left column shows average total error versus computing time for the pedestrians and stop signs. The middle column shows the performance for a fixed negative rate of 0.01. The right hand column shows performance for a fixed false positive rate of 0.01. Note that the pedestrian column has a different scale on the y-axis. The highlighted region shows the region of classifiers with a speed performance between 10-200Hz and standard definition video. In these plots we see that the histogram image based features, LiteHOG+ and HistFeat are well suited to fast detection problems.

By far the greatest loss of discriminance can be attributed to the fact that the histogram image features act on only a *single scale*, that is, 4x4 image patches. For the first few selections of RealBoost this may not be particularly significant, however, on challenging classes like pedestrians this becomes a growing problem and these features are left behind in terms of performance. Unfortunately, it is very difficult to keep the memory efficient behaviour of the histogram image and use a rectangular and scale free datatype like the integral image.

Gradients are binned into only *8 orientations* and the *horizontal and vertical gradients are split* between bins. While 8 direction bins may seem too few, our experimentation has shown that little is to be gained by increasing the direction sensitivity through more bins. I.e. this does not significantly weaken per feature discriminance. The splitting of the horizontal and vertical gradients between bins (see Figure 3) allows a simpler calculation in Equation 1 by foregoing the trigonometric identities. For pedestrians this has been found to contribute to a small but measurable loss of discriminance. However, the discriminant loss here is not enough to justify a slower calculation of the histogram image.

Lastly, the *magnitude thresholding* step (see Section 2.1.1) lowers discriminance in two ways. Firstly, it ignores gradients which are below the threshold. Secondly it loses any magnitude information apart from its relative magnitude to the threshold. I.e. all gradient edges stronger than the threshold are considered equal. However, this parameter can be tuned for each feature class to preserve the greatest signal through to the training and selection algorithm.

As future work we are interested in implementing a "distilled" version of the Region Covariance feature, which has been so successful on pedestrians [15]. This feature relies on some similar feature measurements such as x and y gradients and their second derivative counterparts. We believe that a similar path may be taken to produce a "LiteRegionCovariance" feature which is faster and less memory intensive, but preserves as much of the parent feature's discriminance as possible.

4. Conclusion

This paper serves as a guide to feature design for time constrained detection problems. Results are demonstrated using a variety of feature implementations, with particular attention given to the designed-for-speed HistFeat and LiteHOG+ features. We have outlined the numerous sources of bottlenecks arising from different precomputed datatype designs, feature evaluation methods and learnt models. We have shown how extremely fast yet discriminant features can be produced using the histogram image datatype.

The paper shows how incorporating more advanced learners into a classifier can greatly improve the performance of a classifier, both in terms of speed and robustness. Many researchers who currently use very simple models with slower features will find they are directly able to incorporate better learners with little to no speed penalties.

The time-error plot analysis provides a useful reference to a detector designer seeking the most appropriate features for a given problem. The overall methodology used provides some direction to researchers attempting to optimize an existing slow and memory intensive feature with high discriminance into a faster child feature. This provides hope of improvement for the growing number of popular features, such as the Region Covariance feature, which are very resource intensive.

References

[1] A. Buja and W. Stuetzle. Response to Mease and Wyner, Evidence Contrary to the Statistical View of Boosting, JMLR 9: 131–156, 2008. *Journal of Machine Learning Research*, 9:165–170, 2008.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of Computer Vision and Pattern Recognition CVPR*, June 2005.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

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

[5] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

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

[7] 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.

[8] G. Overett, L. Petersson, L. Andersson, and N. Pettersson. Boosting a heterogeneous pool of fast hog features for pedestrian and sign detection. *Intelligent Vehicles Symposium*, 2009.

[9] 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.

[10] 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*, 18(8):1140–1151, August 2008.

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

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

[13] R. E. Schapire and Y. Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

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

[15] O. Tuzel, F. Porikli, and P. Meer. Human detection via classification on riemannian manifolds. In *Computer Vision and Pattern Recognition*. IEEE Computer Society, 2007.

[16] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition*, 2001.

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

[18] 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.

[19] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1491–1498, Washington, DC, USA, 2006. IEEE Computer Society.