Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection

Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky
Microprocessor Research Lab, Intel Labs
Intel Corporation, Santa Clara, CA 95052, USA
Rainer.Lienhart@intel.com

MRL Technical Report, May 2002, revised December 2002

ABSTRACT

Recently Viola et al. have introduced a rapid object detection scheme based on a boosted cascade of simple feature classifiers. In this paper we introduce and empirically analysis two extensions to their approach: Firstly, a novel set of rotated haar-like features is introduced. These novel features significantly enrich the simple features of [6] and can also be calculated efficiently. With these new rotated features our sample face detector shows off on average a 10% lower false alarm rate at a given hit rate. Secondly, we present a through analysis of different boosting algorithms (namely Discrete, Real and Gentle Adaboost) and weak classifiers on the detection performance and computational complexity. We will see that Gentle Adaboost with small CART trees as base classifiers outperform Discrete Adaboost and stumps. The complete object detection training and detection system as well as a trained face detector are available in the Open Computer Vision Library at sourceforge.net [8].

1 Introduction

Recently Viola et al. have proposed a multi-stage classification procedure that reduces the processing time substantially while achieving almost the same accuracy as compared to a much slower and more complex single stage classifier [6]. This paper extends their rapid object detection framework in two important ways:

Firstly, their basic and over-complete set of haar-like features is extended by an efficient set of 45° rotated features, which add additional domain-knowledge to the learning framework and which is otherwise hard to learn. These novel features can be computed rapidly at all scales in constant time.

Secondly, we empirically show that Gentle Adaboost outperforms (with respect to detection accuracy) Discrete and Real Adaboost for object detection tasks, while having a lower computational complexity, i.e., requiring a lower number of features for the same performance. Also, the usage of small decision trees instead of stumps as weak classifiers further improves the detection performance at a comparable detection speed.

The complete training and detection system as well as a trained face detector are available in the Open Computer Vision Library at http:/sourceforge.net/projects/opencvlibrary/ [8].

2 Features

The main purpose of using features instead of raw pixel values as the input to a learning algorithm is to reduce/increase the in-class/out-of-class variability compared to the raw input data, and thus making classification easier. Features usually encode knowledge about the domain, which is difficult to learn from a raw and finite set of input data.

The complexity of feature evaluation is also a very important aspect since almost all object detection algorithms slide a fixed-size window at all scales over the input image. As we will see, our features can be computed at any position and any scale in the same constant time. At most 8 table lookups are needed per feature.

2.1 Feature Pool

Our feature pool was inspired by the over-complete haar-like features used by Papageorgiou *et al.* in [5,4] and their very fast computation scheme proposed by Viola *et al.* in [6], and is a generalization of their work.

Let us assume that the basic unit for testing for the presence of an object is a window of $W \times H$ pixels. Also assume that we have a very fast way of computing the sum of pixels of any upright and 45° rotated rectangle inside the window. A rectangle is specified by the tuple $r = (x, y, w, h, \alpha)$ with $0 \le x, x + w \le W$, $0 \le y, y + h \le H$, $x, y \ge 0$, $w, h > 0$, $\alpha \in \{0°, 45°\}$, and its pixel sum is denoted by $RecSum(r)$. Two examples of such rectangles are given in Figure 1.

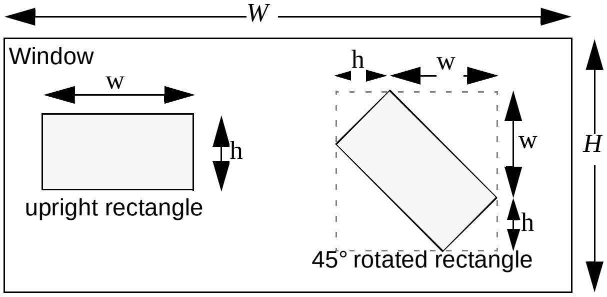

Figure 1: Example of an upright and 45° rotated rectangle.

Our raw feature set is then the set of all possible features of the form

$$feature_I = \sum_{i \in I = \{1, \ldots, N\}} \omega_i \cdot RecSum(r_i) ,$$

where the weights $\omega_i \in \Re$, the rectangles r_i, and N are arbitrarily chosen.

This raw feature set is (almost) infinitely large. For practical reasons, it is reduced as follows:

1. Only weighted combinations of pixel sums of two rectangles are considered (i.e., $N = 2$).
2. The weights have opposite signs, and are used to compensate for the difference in area size between the two rectangles. Thus, for non-overlapping rectangles we have

$-w_0 \cdot Area(r_0) = w_1 \cdot Area(r_1)$. Without restrictions we can set $w_0 = -1$ and get $w_1 = Area(r_0)/Area(r_1)$.

3. The features mimic haar-like features and early features of the human visual pathway such as center-surround and directional responses.

These restrictions lead us to the 14 feature prototypes shown in Figure 2:

- Four edge features,
- Eight line features, and
- Two center-surround features.

These prototypes are scaled independently in vertical and horizontal direction in order to generate a rich, over-complete set of features. Note that the line features can be calculated by two rectangles only. Hereto it is assumed that the first rectangle r_0 encompasses the black and white rectangle and the second rectangle r_1 represents the black area. For instance, line feature (2a) with total height of 2 and width of 6 at the top left corner (5,3) can be written as

$$feature_I = -1 \cdot RecSum(5,3,6,2,0°) + 3 \cdot RecSum(7,3,2,2,0°) .$$

Only features (1a), (1b), (2a), (2c) and (4a) of Figure 2 have been used by [4,5,6]. In our experiments the additional features significantly enhanced the expressional power of the learning system and consequently improved the performance of the object detection system. This is especially true if the object under detection exhibit diagonal structures such as it is the case for many brand logos.

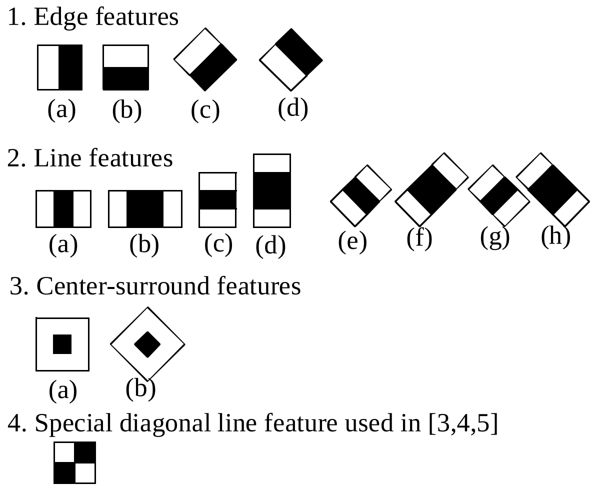
1. Edge features
(a)
(b)
(c)
(d)
2. Line features
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
3. Center-surround features
(a)
(b)
4. Special diagonal line feature used in [3,4,5]

Figure 2: Feature prototypes of simple haar-like and center-surround features. Black areas have negative and white areas positive weights.

NUMBER OF FEATURES. The number of features derived from each prototype is quite large and differs from prototype to prototype and can be calculated as follows. Let $X = \lfloor W/w \rfloor$ and $Y = \lfloor H/h \rfloor$ be the maximum scaling factors in *x* and *y* direction. An upright feature of size *w*x*h* then generates

$$XY \cdot \left(W+1-w\frac{X+1}{2}\right) \cdot \left(H+1-h\frac{Y+1}{2}\right)$$

features for an image of size *W*x*H*, while a 45° rotated feature generates

$$XY \cdot \left(W+1-z\frac{X+1}{2}\right) \cdot \left(H+1-z\frac{Y+1}{2}\right) \text{ with } z = w+h.$$

Table 1 lists the number of features for a window size of 24x24.

Feature Type	w/h	X/Y	#
1a ; 1b	2/1 ; 1/2	12/24 ; 24/12	43,200
1c ; 1d	2/1 ; 1/2	8/8	8,464
2a ; 2c	3/1 ; 1/3	8/24 ; 24/8	27,600
2b ; 2d	4/1 ; 1/4	6/24 ; 24/6	20,736
2e ; 2g	3/1 ; 1/3	6/6	4,356
2f ; 2h	4/1 ; 1/4	4/4	3,600
3a	3/3	8/8	8,464
3b	3/3	3/3	1,521
Sum			117,941

Table 1: Number of features inside of a 24x24 window for each prototype.

2.2 Fast Feature Computation

All our features can be computed very fast in constant time for any size by means of two auxiliary images. For upright rectangles the auxiliary image is the *Summed Area Table* $SAT(x,y)$. $SAT(x,y)$ is defined as the sum of the pixels of the upright rectangle ranging from the top left corner at (0,0) to the bottom right corner at (*x*,*y*) (see Figure 3a) [6]:

$$SAT(x,y) = \sum_{x' \le x, y' \le y} I(x',y') .$$

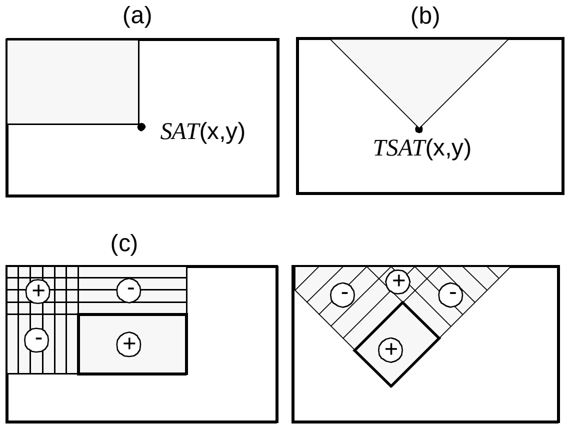

Figure 3: (a) Upright *Summed Area Table* (*SAT*) and (b) *Rotated Summed Area Table* (*RSAT*); calculation scheme of the pixel sum of upright (c) and rotated (d) rectangles.

It can be calculated with one pass over all pixels from left to right and top to bottom by means of

$$SAT(x,y) = SAT(x,y-1) + SAT(x-1,y) + I(x,y) - SAT(x-1,y-1)$$

with

$$SAT(-1,y) = SAT(x,-1) = SAT(-1,-1) = 0$$

From this the pixel sum of any upright rectangle $r = (x,y,w,h,0)$ can be determined by four table lookups (see also Figure 3(c):

$$\begin{aligned} RecSum(r) = {} & SAT(x-1,y-1)+SAT(x+w-1,y+h-1) \\ & -SAT(x-1,y+h-1)-SAT(x+w-1,y-1) \end{aligned}$$

This insight was first published in [6].

For 45° rotated rectangles the auxiliary image is the *Rotated Summed Area Table* $RSAT(x,y)$. It is defined as the sum of the pixels of a 45° rotated rectangle with the bottom most corner at (x,y) and extending upwards till the boundaries of the image (see Figure 3b):

$$RSAT(x,y) = \sum_{y' \le y,\, y' \le y-|x-x'|} I(x',y') .$$

It can be calculated also in one pass from left to right and top to bottom over all pixels by

$$\begin{aligned} RSAT(x,y) = {} & RSAT(x-1,y-1)+RSAT(x+1,y-1)-RSAT(x,y-2) \\ & +I(x,y)+I(x,y-1) \end{aligned}$$

with

$$RSAT(-1,y) = RSAT(x,-1) = RSAT(x,-2) = 0$$

$$RSAT(-1,-1) = RSAT(-1,-2) = 0$$

as shown in Figure 4. From this the pixel sum of any rotated rectangle $r = (x,y,w,h,45°)$ can be determined by four table lookups (see Figure 5):

$$\begin{aligned} RecSum(r) = {} & RSAT(x-h+w,y+w+h-1)+RSAT(x,y-1) \\ & -RSAT(x-h,y+h-1)-RSAT(x+w,y+w-1) \end{aligned} .$$

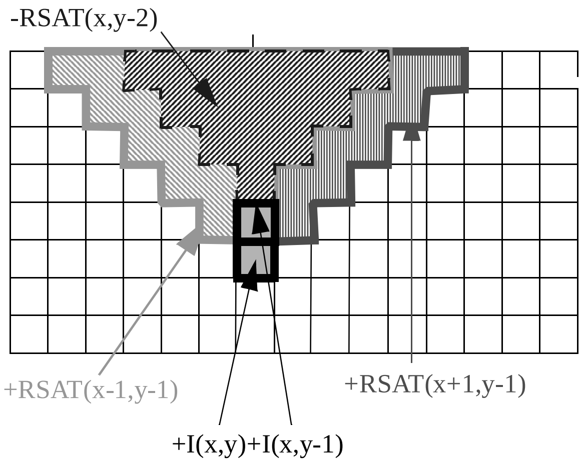

Figure 4: Calculation scheme for Rotated Summed Area Tables (*RSAT*).

2.3 Fast Lighting Correction

The special properties of the haar-like features also enable fast contrast stretching of the form

$$\bar{I}(x,y) = \frac{I(x,y)-\mu}{c\sigma} , \; c \in R^{+} .$$

μ can easily be determined by means of *SAT*(x,y). Computing σ, however, involves the sum of squared pixels. It can easily be derived by calculating a second set of *SAT* and *RSAT* auxiliary images for $I^2(x,y)$. Then, calculating σ for any window requires only 4 additional table lookups. In our experiments c was set to 2.

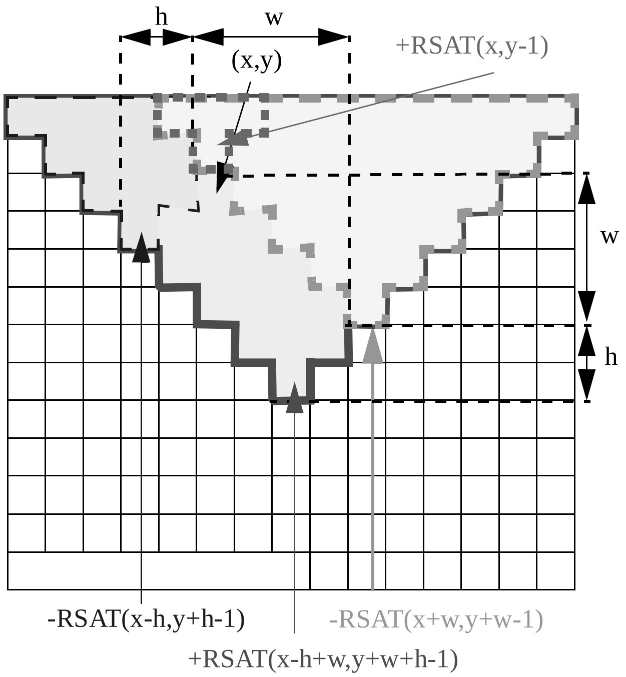

Figure 5: Calculation scheme for rotated areas.

3 (Stage) Classifier

We use boosting as our basic classifier. Boosting is a powerful learning concept. It combines the performance of many "weak" classifiers to produce a powerful 'committee' [1]. A weak classifier is only required to be better than chance, and thus can be very simple and computationally inexpensive. Many of them smartly combined, however, result in a strong classifier, which often outperforms most 'monolithic' strong classifiers such as SVMs and Neural Networks.

Different variants of boosting are known such as Discrete Adaboost (see Figure 6), Real AdaBoost, and Gentle AdaBoost (see Figure 7)[1]. All of them are identical with respect to computational complexity from a classification perspective, but differ in their learning algorithm. All three are investigated in our experimental results.

Learning is based on N training examples $(x_1,y_1),\ldots,(x_N,y_N)$ with $x \in \Re^k$ and $y_i \in \{-1,1\}$. x_i is a K-component vector. Each component encodes a feature relevant for the learning task at hand. The desired two-class output is encoded as –1 and +1. In the case of object detection, the input component x_i is one haar-like feature. An output of +1 and -1 indicates whether the input pattern does contain a complete instance of the object class of interest.

4 Cascade of Classifiers

A cascade of classifiers is a degenerated decision tree where at each stage a classifier is trained to detect almost all objects of interest (frontal faces in our example) while rejecting a certain fraction of the non-object patterns [6] (see Figure 8). For instance, in our case each stage was trained to eliminated 50% of the non-face patterns while falsely eliminating only 0.1% of the frontal face patterns; 20 stages were trained. Assuming that our test set is representative for the learning task, we can expect a false alarm rate about $0.5^{20} \approx 9.6e-07$ and a hit rate about $0.999^{20} \approx 0.98$.

Discrete AdaBoost (Freund & Schapire [1])

```
1. Given $N$ examples $(x_1, y_1), \ldots, (x_N, y_N)$ with $x \in \Re^k, y_i \in \{-1, 1\}$
2. Start with weights $w_i = 1/N$, $i = 1, \ldots, N$.
3. Repeat for $m = 1, \ldots, M$
    (a) Fit the classifier $f_m(x) \in \{-1, 1\}$ using weights $w_i$ on the training data $(x_1, y_1), \ldots, (x_N, y_N)$ .
    (b) Compute $err_m = E_w[1_{(y \neq f_m(x))}]$ , $c_m = \log((1 - err_m)/err_m)$ .
    (c) Set $w_i \leftarrow w_i \cdot \exp(c_m \cdot 1_{(y_i \neq f_m(x_i))})$ , $i = 1, \ldots, N$, and renormalize weights so that $\sum_i w_i = 1$ .
4. Output the classifier $sign\left[\sum_{m=1}^{M} c_m \cdot f_m(x)\right]$
```

Figure 6:Discrete AdaBoost training algorithm [1].

Gentle AdaBoost

```
1. Given $N$ examples $(x_1, y_1), \ldots, (x_N, y_N)$ with $x \in \Re^k, y_i \in \{-1, 1\}$
2. Start with weights $w_i = 1/N$, $i = 1, \ldots, N$.
3. Repeat for $m = 1, \ldots, M$
    (a) Fit the regression function $f_m(x)$ by weighted least-squares of $y_i$ to $x_i$ with weights $w_i$
    (c) Set $w_i \leftarrow w_i \cdot \exp(-y_i \cdot f_m(x_i))$ , $i = 1, \ldots, N$, and renormalize weights so that $\sum_i w_i = 1$ .
4. Output the classifier $sign\left[\sum_{m=1}^{M} f_m(x)\right]$
```

Figure 7:Gentle AdaBoost training algorithm [1]

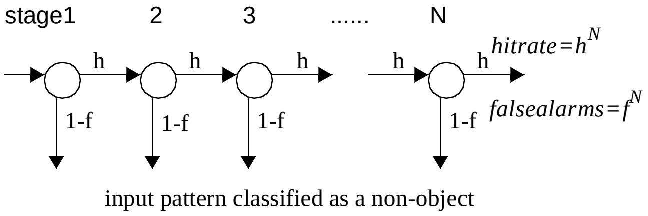

Figure 8: Cascade of classifiers with *N* stages. At each stage a classifier is trained to achieve a hit rate of *h* and a false alarm rate of *f*.

Each stage was trained using one out of the three Boosting variants. Boosting can learn a strong classifier based on a (large) set of weak classifiers by re-weighting the training samples. Weak classifiers are only required to be slightly better than chance. Our set of weak classifiers are all classifiers which use one feature from our feature pool in combination with a simple binary thresholding decision. At each round of boosting, the feature-based classifier is added that best classifies the weighted training samples. With increasing stage number the number of weak classifiers, which are needed to achieve the desired false alarm rate at the given hit rate, increases (for more detail see [6]).

5 Experimental Results

All experiments were performanced on the complete CMU Frontal Face Test Set of 130 grayscale pictures with 510 frontal faces [7]. A hit was declared if and only if

- the Euclidian distance between the center of a detected and actual face was less than 30% of the width of the actual face as well as
- the width (i.e., size) of the detected face was within ±50% of the actual face width.

Every detected face, which was not a hit, was counted as a false alarm. Hit rates are reported in percent, while the false alarms are specified by their absolute numbers in order to make the results comparable with related work on the CMU Frontal Face Test set.

Except otherwise noted 5000 positive frontal face patterns and 3000 negative patterns filtered by stage 0 to *n*-1 were used to train stage *n* of the cascade classifer. The 5000 positive frontal face patterns were derived from 1000 original face patterns by random rotation about ±10 degree, random scaling about ±10%, random mirroring and random shifting up to ±1 pixel. Each stage was trained to reject about half of the negative patterns, while correctly accepting 99.9% of the face patterns. A fully trained cascade consisted of 20 stages.

During detection, a sliding window was moved pixel by pixel over the picture at each scale. Starting with the original scale, the features were enlarged by 10% and 20%, respectively (i.e., representing a rescale factor of 1.1 and 1.2, respectively) until exceeding the size of the picture in at least one dimension.

Often multiple faces are detect at near by location and scale at an actual face location. Therefore, multiple nearby detection results were merged. Receiver Operating Curves (ROCs) were constructed by varing the required number of detected faces per actual face before merging into a single detection result.

During experimentation only one parameter was changed at a time. The best mode of a parameter found in an experiment was used for the subsequent experiments.

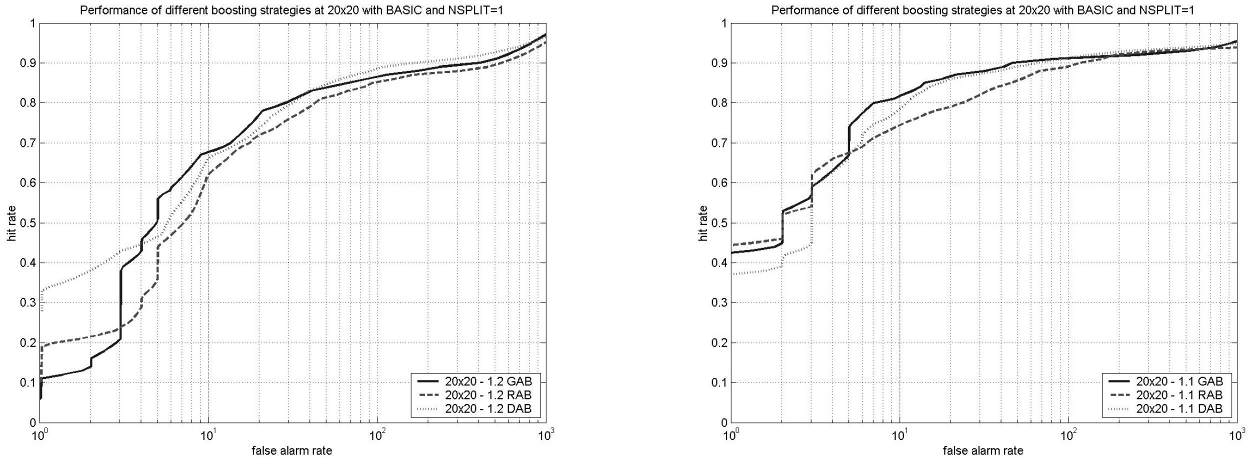

Figure 10: Performance comparison between identically trained cascades with three different boosting algorithms. Only the basic feature set and stumps as weak classifiers (nsplit=1) were used.

5.1 Feature Scaling

Any multi-scale image search requires either rescaling of the picture or the features. One of the advantage of the Haar-like features is that they can easily be rescaled. Independent of the scale each feature requires only a fixed number of look-ups in the sum and squared sum auxilary images. These look-ups are performed relative to the top left corner and must be at integral positions. Obviously, by fractional rescaling the new correct positions become fractional. A plain vanilla solution is to round all relative look-up positions to the nearest integer position. However, performance may degrade significantly, since the ratio between the two areas of a feature may have changed significantly compared to the area ratio at training due to rounding. One solution is to correct the weights of the different rectangle sums so that the original area ratio between them for a given haar-like feature is the same as it was at the original size. The impact of this weight adapation on the performance is amazing as can be seen in Figure 9.”*-Rounding” show the ROCs for simple rounding, while “*-AreaRatio” shows the impact if also the weight of the different rectangles is adjusted to reflect the weights in the feature at the original scale.

5.2 Comparision Between Different Boosting Algorithms

We compared three different boosting algorithms:

- Discrete Adaboost,
- Real Adaboost, and
- Gentle Adaboost.

Three 20-stage cascade classifiers were trained with the respective boosting algorithm using the basic feature set (i.e., features 1a, 1b, 2a, 2c, and 4a of Figure 2) and stumps as the weak classifiers. As can be seen from Figure 10, Gentle Adaboost outperformed the other two boosting algorithm, despite the fact that it needed on average fewer features (see Table 2, second column). For instance, at a an absolute false alarm rate of 10 on the CMU test set, RAB deteted only 75.4% and DAB only 79.5% of all frontal faces, while GAB achieved 82.7% at a rescale factor of 1.1. Also, the smaller rescaling factor of 1.1 was very beneficial if a very low false alarm rate at high detection performance had to be achieved. At 10 false alarms on the CMU test set, GAB improved from 68.8% detection rate with rescaling factor of 1.2 to 82.7% at a rescaling factor of 1.1.

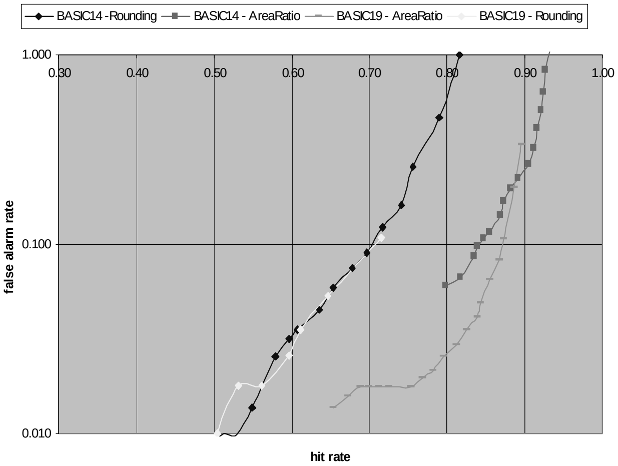

Figure 9: Performance comparision between different feature scaling approaches. “*-Rounding” rounds the fractional position to the nearest integer position, while “*-AreaRatio” also restores the ratio between the different rectangles to its original value used during training.

Table 2 shows in the second column (nsplit =1) the average number of features needed to be evaluted for background patterns by the different classifiers. As can be seen GAB is not only the best, but also the fastest classifier. Therefore, we only investigate a rescale scaling factor 1.1 and GAB in the subsequent experiments.

NSPLIT	1	2	3	4
DAB	45.09	44.43	31.86	44.86
GAB	30.99	36.03	28.58	35.40
RAB	26.28	33.16	26.73	35.71

Table 2: Average number of features evaluated per background pattern at a pattern size of 20x20.

5.3 Input Pattern Size

Many different input pattern sizes have been reported in related work on face detection ranging from 16x16 up to 32x32. However, none of them have systematically investigated the effect of the input pattern size on detection performance. As our experiments show for faces an input pattern size of 20x20 achieves the highest hit rate at an absolute false alarms between 5 and 100 on the CMU Frontal Face Test Set (see Figure 11). Only for less than 5 false alarms, an input pattern size of 24x24 worked better. A similar observation has been made by [2].

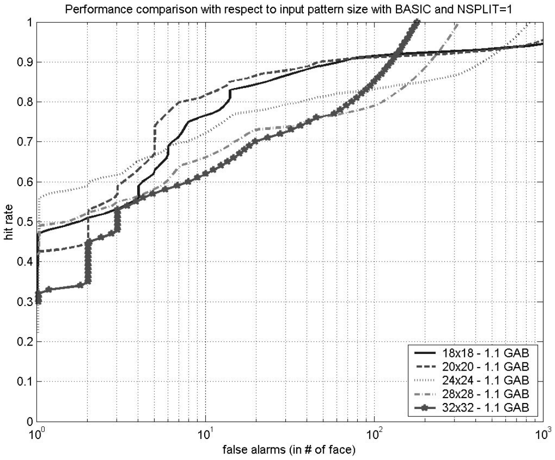

Figure 11: Performance comparison between identically trained cascades, but with different input pattern sizes. GAB was used together with the basic feature set and stumps as weak classifiers (nsplit=1).

5.4 Tree vs. Stumps

Stumps as weak classifer do not allow learning dependencies between features. In general, N split nodes are needed to model dependency between N-1 variables. Therefore, we allow our weak classifier to be a CART tree with NSPLIT split nodes. Then, NSPLIT=1 represents the stump case.

As can be seen from Figure 12 and Figure 13 stumps are outperformed by weak tree classifiers with 2, 3 or 4 split nodes. For 18x18 four split nodes performed best, while for 20x20 two nodes were slighlty better. The difference between weak tree classifiers with 2, 3 or 4 split nodes is smaller than their superiority with respect to stumps.The order of the computational complexity of the resulting detection classifier was unaffected by the choise of the value of NSPLIT (see Table 1). The more powerful CARTs proportionally needed less weak classifiers to achieve the same performance at each stage.

5.5 Basic vs. Extended Haar-like Features

Two face detection systems were trained: One with the basic and one with the extended haar-like feature set. On average the false alarm rate was about 10% lower for the extended haar-like feature set at comparable hit rates. Figure 14 shows the ROC for both classifiers using 12 stages. At the same time the computational complexity was comparable. The average number of features evaluation per patch was about 31 (see [3] for more details).

These results suggest that although the larger haar-like feature set usually complicates learning, it was more than paid of by the added domain knowledge. In principle, the center surround feature would have been sufficient to approximate all other features, however, it is in general hard for any machine learning algorithm to learn joint behavior in a reliable way.

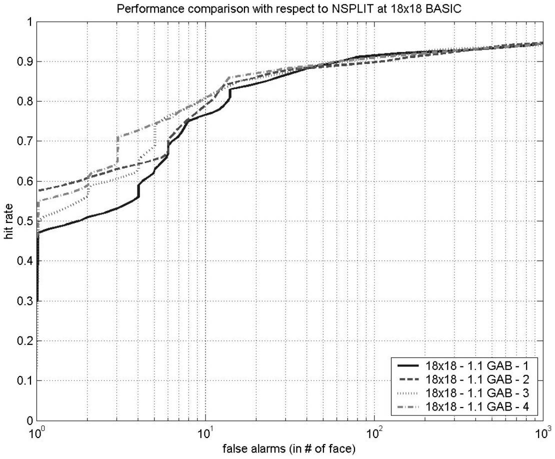

Figure 12: Performance comparison with respect to the order of the weak CART classifiers. GAB was used together with the basic feature set and a pattern size of 18x18.

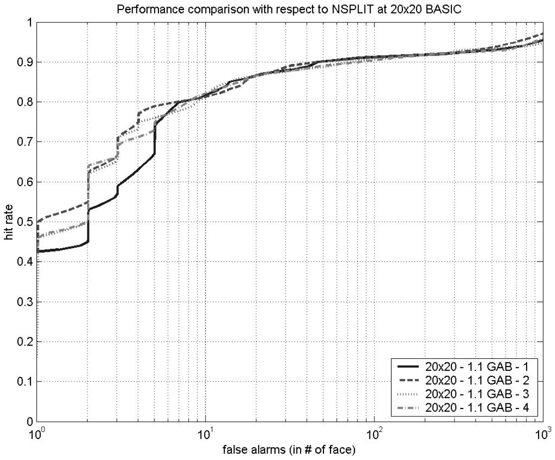

Figure 13: Performance comparison with respect to the order of the weak CART classifiers. GAB was used together with the basic feature set and a pattern size of 20x20.

5.6 Training Set Size

So far, all trained cascades used 5000 positive and 3000 negative examples per stage to limit the computational complexity during training. We also trained one 18x18 classifiers with all positive face examples, 10795 in total and 5000 negative training examples. As can be seen from Figure 15, there is little difference in the training results. Large training sets only slightly improve performance indicating that the cascade trained with 5000/3000 examples already came close to its representation power.

Conclusion

Our experimental results suggest, that 20x20 is the optimal input pattern size for frontal face detection. In addition, they show that

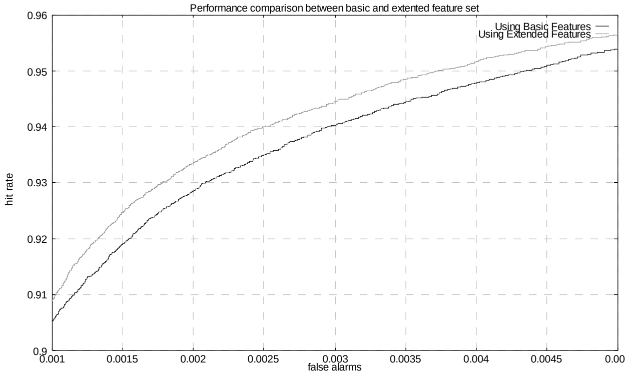

Figure 14: Basic versus extended feature set: On average the false alarm rate of the face detector exploiting the extended feature set was about 10% better at the same hit rate (taken from [3]).

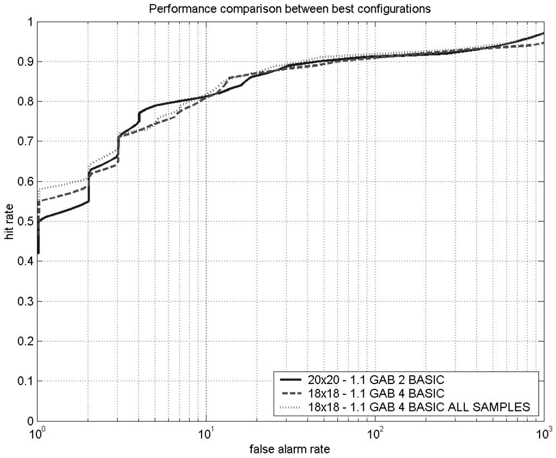

Figure 15: Performance comparison with respect to the training set size. One 18x18 classifier was trained with 10795 face and 5000 non-face examples using GAB and the basic feature set.

Gentle Adaboost outperforms Discrete and Real Adaboost. Logitboot could not be used due to convergence problem on later stages in the cascade training. It is also beneficial not just to use the simplest of all tree classifiers, i.e., stumps, as the basis for the weak classifiers, but representationally more powerful classifiers such as small CART trees, which can model second and/or third order dependencies.

We also introduced an extended set of haar-like features. Although frontal faces exhibit little diagonal structures, the 45 degree rotated features increased the accuracy. In practice, the have observed that the rotated features can boost detection performance if the object under detection exhibit some diagonal structures such as many brand logos.

The complete training and detection system as well as a trained face detector are available in the Open Computer Vision Library at http:/sourceforge.net/projects/opencvlibrary/ [8].

6 REFERENCES

[1] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In Machine Learning: Proceedings of the Thirteenth International Conference, Morgan Kauman, San Francisco, pp. 148-156, 1996.

[2] Stan Z. Li, Long Zhu, ZhenQiu Zhang, Andrew Blake, HongJiang Zhang, and Harry Shum. Statistical Learning of Multi-View Face Detection. In Proceedings of The 7th European Conference on Computer Vision. Copenhagen, Denmark. May, 2002.

[3] Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002.

[4] A. Mohan, C. Papageorgiou, T. Poggio. Example-based object detection in images by components. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, No. 4, pp. 349 -361, April 2001.

[5] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for Object Detection. In *International Conference on Computer Vision*, 1998.

[6] Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001.

[7] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In IEEE Patt. Anal. Mach. Intell., Vol. 20, pp. 22-38, 1998.

[8] Open Computer Vision Library. http:/sourceforge.net/projects/opencvlibrary/