MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A New Weak Learning Algorithm for Real Hyperplane Features Applied to Face Detection

Raphael Pelossof, Michael Jones

TR2007-074 June 2008

Abstract

This paper explores the use of thresholded hyperplanes as the building blocks of a classifier for face detection. We are motivated by the work of Viola and Jones [10] who used Haar-like wavelet features as their weak classifiers in the AdaBoost learning algorithm. These weak classifiers were chosen for their speed. We explore how much may be gained by using more powerful but less computationally efficient weak classifiers. The generalized haar wavelets used in Viola and Jones can be viewed as a constrained subset of linear hyperplanes. Can a more powerful detector be constructed if we use unconstrained linear hyperplanes in place of the generalized Haar wavelets. In addition to being of theoretical interest, this question has practical importance for hardware implementations of a face detector in which dot products may be very fast to compute. The difficulty with using thresholded hyperplanes as weak classifiers is that the brute force search over all possible hyperplanes which was used in Viola-Jones is no longer practical. We propose a new gradient descent based algorithm which finds separating hyperplanes by directly minimizing the AdaBoost Z score. We also provide a baseline comparison to other search algorithms for unconstrained hyperplanes.

A new weak learning algorithm for real hyperplane features applied to face detection

Raphael Pelossof
Columbia University
1214 Amsterdam Ave, New York, NY 10027
pelossof@cs.columbia.edu

Michael J. Jones
MERL
201 Broadway, Cambridge, MA 02144
mjones@merl.com

Abstract

This paper explores the use of thresholded hyperplanes as the building blocks of a classifier for face detection. We are motivated by the work of Viola and Jones [10] who used Haar-like wavelet features as their weak classifiers in the AdaBoost learning algorithm. These weak classifiers were chosen for their speed. We explore how much may be gained by using more powerful but less computationally efficient weak classifiers. The generalized Haar wavelets used in Viola and Jones can be viewed as a constrained subset of linear hyperplanes. Can a more powerful detector be constructed if we use unconstrained linear hyperplanes in place of the generalized Haar wavelets? In addition to being of theoretical interest, this question has practical importance for hardware implementations of a face detector in which dot products may be very fast to compute.

The difficulty with using thresholded hyperplanes as weak classifiers is that the brute force search over all possible hyperplanes which was used in Viola-Jones is no longer practical. We propose a new gradient descent based algorithm which finds separating hyperplanes by directly minimizing the AdaBoost Z score. We also provide a baseline comparison to other search algorithms for unconstrained hyperplanes.

1. Introduction

Automatic face detection in photographs has undergone rapid development in recent years. One important development was the rapid detection framework of Viola and Jones [10]. Part of the success of their framework is the use of fast-to-compute weak classifiers that are combined to yield a very accurate face detector. These weak classifiers are thresholded Haar-like wavelets also called rectangle features. One interesting question that arises is how much better could more powerful weak classifiers do? Haar-like wavelets can be represented as hyperplanes with particular constraints. Applying the Haar-like wavelet to the input image patch is then simply a dot product. In practice, Haar-like wavelets are computed more efficiently by using an integral image representation, but they are equivalent to a dot product (followed by thresholding). Thus, one natural choice for a more powerful weak classifier is an unrestricted hyperplane followed by thresholding. This paper answers the question of how much better such thresholded hyperplanes are than rectangle features for face detection.

This question has practical importance as well as theoretical interest. Various companies are putting face detectors into products (digital cameras, cell phones, digital video recorders, etc). In some of these products the face detector is implemented on either a specialized ASIC or on an embedded processor or DSP chip. Some such hardware can compute a general dot product just as fast as a rectangle feature. In such cases, a significant advantage may be gained by using a face detector based on unrestricted hyperplanes instead of Haar-like wavelets.

2. Related Work

An extensive survey of multiple different face detection algorithms was conducted by Lienhart and Maydt [5]. We however, constrain the discussion to research related to hyperplane based weak classifiers. There have been a few other papers that have explored the idea of using more powerful hyperplane based weak classifiers within AdaBoost to build a face detector. In their paper on Kullback-Leibler boosting, Liu and Shum [6] also used a hyperplane as their weak classifier. They provide a very detailed framework to find the most informative classifying hyperplanes. However, they mention that the stochastic ascent that is used as the optimization algorithm in lower dimensional problems becomes inefficient in high dimensions because of the size of the search space. They propose a 1D optimization to find optimal features which seems to work very well. Their algorithm also uses multiple thresholds for each hyperplane to form a sophisticated decision boundary. In comparison,

our optimization algorithm works on classification tasks independent of the dimensionality of the data, treating low and high dimensional problems similarly. Additionally, our weak classifier uses only a single threshold which results in a much simpler decision boundary, and requires less overhead and less time to evaluate.

Another paper that used more powerful weak classifiers is Huang *et al.* [3]. They introduced the idea of granular filters which are linear combinations of sums within squares. These filters can also be computed as a dot product. The granular filters can thus also be viewed as a constrained set of hyperplanes although less constrained than the Haar-like filters. Their goal was to use a more powerful set of weak classifiers but ones that are still computationally efficient in software. Huang *et al.* also had to struggle with the problem of a huge search space and they developed a heuristic based weak learning algorithm to solve this problem.

Using thresholded hyperplanes as classifiers, both Li and Zhang [4], Viola and Jones [10], and Lienhart and Maydt [5] select at each AdaBoost iteration a single filter that minimizes a cost function from a fixed overcomplete set of Haar-like filters. Due to the size of the Haar-like feature space, the methods are forced to sample only a subset of the Haar-like filters. This subset, although overcomplete, includes only filters with a small number of boxes (up to 4 boxes per filter). It is very likely that by either enlarging the sampled set by adding more boxes, or removing the box restrictions on the hyperplane, the accuracy of the detector will be increased.

3. Overview of Viola-Jones detection framework

The Viola-Jones [10] detection framework is based on learning a strong classifier that distinguishes face patches from non-face patches using the AdaBoost learning algorithm. The core of AdaBoost is the weak learning algorithm that chooses a weak classifier which has better than chance error on the weighted training data. The weak classifier can be any classifier. In the case of Viola-Jones, it is a thresholded rectangle feature as illustrated in figure 1. The strong classifier, $C(\mathbf{x})$ resulting from confidence-rated AdaBoost training has the following form (which is slightly different from that used in [10]):

$$C(\mathbf{x}) = sign(\sum_{k=1}^{K} h_k(\mathbf{x}) - T) \quad (1)$$

where

$$h_k(\mathbf{x}) = \begin{cases} \alpha_k & \text{if } f_k(\mathbf{x}) > \theta_k \\ \beta_k & \text{otherwise} \end{cases} \quad (2)$$

$\alpha_k, \beta_k \in \mathcal{R}$, $f_k(\mathbf{x})$ is a rectangle filter and $\mathbf{x}$ is an input image patch. We call $h_k(\mathbf{x})$, which is a classifier, a rectangle *feature* and $f_k(\mathbf{x})$, which is a linear function of the image patch, a rectangle *filter*.

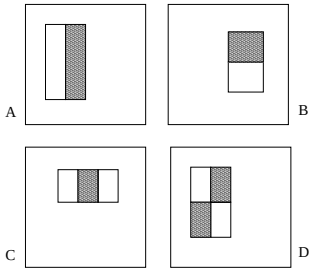

Figure 1. Example rectangle filters shown relative to the image patch. The sum of the pixels in the gray rectangles are subtracted from the sum of pixels in the white rectangles.

The basic AdaBoost algorithm using confidence-rated predictions [1, 8] slightly specialized for face detection learning using linear filters is given in figure 2.

In the case of 24x24 pixel example images, the number of possible rectangle filters is not too large (on the order of 100,000) and so can be searched over in a brute force manner to find the one with lowest error on the weighted data. As noted previously, a rectangle filter can be represented as a hyperplane and evaluated on an image patch by a dot product. In this sense, the set of rectangle filters comprise a very restricted set of hyperplanes and the interesting question arises as to how much better the face detector could be if it used unrestricted hyperplanes instead. This question has practical importance when considering hardware implementations in which dot products can be computed very fast.

Another key component of the Viola-Jones framework is the use of a cascade of classifiers to dramatically increase the speed of the detector. Viola and Jones show that using a cascade, the average number of weak classifiers computed per patch on a typical image is only 8. For this reason, we concern ourselves in this paper with only the first 10 weak classifiers since this is where almost all of the computation is done.

There are two ways for unrestricted hyperplanes to do "better" than rectangle filters. One is for the accuracy (measured as a ROC curve plotting false positives versus false negatives) to be improved. The other is for the average number of weak classifiers evaluated per patch to be reduced. The speed of the face detector is directly proportional to the average number of weak classifiers computed per patch. In a software implementation in which dot products are much slower than computing rectangle filters using the integral image, reducing the average number of weak classifiers computed is overwhelmed by the slowness of the dot product. But in a hardware implementation in which dot products are fast, reducing the average number of weak

- Given example images and labels $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i$ is an image represented as a vector of dimension P and $y_i \in \{-1, 1\}$ for negative and positive examples, respectively.
- Initialize weights $D_1(i) = \frac{1}{n}$.
- For $k = 1, \ldots, K$:

  1. Call the weak learner using distribution D_k on the examples to yield a weak classifier

  $$h_k(\mathbf{x}) = \begin{cases} \alpha_k & \text{if } f_k(\mathbf{x}) > \theta_k \\ \beta_k & \text{otherwise} \end{cases}$$

  where $\alpha_k, \beta_k \in \mathcal{R}$, $f_k(\mathbf{x}) = \mathbf{f}_k^T \mathbf{x}$, $\mathbf{f}_k$ is a vector with the same dimensionality as $\mathbf{x}$ and $\mathbf{f}_k^T$ represents the transpose of $\mathbf{f}_k$.

  2. Update the weights:

  $$D_{k+1}(i) = \frac{D_k(i) exp(-y_i h_k(\mathbf{x}_i))}{Z_k}$$

  where Z_k is a normalization factor chosen so that D_{k+1} will be a distribution.

- The final strong classifier is:

$$C(\mathbf{x}) = sign(\sum_{k=1}^{K} h_k(\mathbf{x}) - T)$$

where T is a threshold which can be adjusted to trade off false positives with false negatives.

Figure 2. The AdaBoost algorithm used to train a strong classifier in the Viola-Jones framework. Adapted from Schapire and Singer [8].

classifiers computed can have a large effect on speed.

The difficulty with using thresholded hyperplanes as weak classifiers is that the simple brute force weak learner used in Viola-Jones no longer works. The search space of 576 (= 24x24) dimensional hyperplanes is astronomical so a manageable subsample of hyperplanes is not sufficient to cover the space adequately.

The main focus of the remainder of the paper is presenting two different weak learner algorithms for thresholded hyperplanes. We also present results using the same weak learner as Viola-Jones and 20,000 randomly chosen hyperplanes to demonstrate that it is indeed inadequate for building a good detector.

4. Selection algorithms for unrestricted hyperplanes

Each of the weak learner search algorithms (step 1 of the AdaBoost algorithm in figure 2) has the following form:

- Construct weak learner $h_k(\mathbf{x})$

  1. Select a hyperplane, $\mathbf{f}$ (somehow)
  2. Given the hyperplane, find the optimal θ, α and β that minimize the Z score.

In some of the weak learners, these two steps are iterated. For example, in the brute force search weak learner these two steps are iterated for every possible hyperplane, and the one that minimizes the Z score is chosen. The Z score is the normalization factor in the weight update step of the AdaBoost algorithm.

$$Z_k = \sum_{i=1}^{n} D_k(i) e^{-y_i h_k(\mathbf{x}_i)} \tag{3}$$

The product of the Z_k's was shown by Schapire and Singer [8] to be a bound on the error and so is a good criteria for the weak learner to minimize.

The main difference among the weak learning algorithms is how the hyperplane is chosen in step 1. Step 2 is exactly the same in each algorithm. This step is computed as follows. First the responses, r_i are computed. These are defined as $r_i = \mathbf{f}^T \mathbf{x}_i$ for each example image patch, $\mathbf{x}_i$. The responses are simply scalars. The responses are then sorted. Next, every possible threshold that falls midway between two sorted responses is tested by computing the associated Z score. To compute the Z score, the optimal α and β are required. Schapire and Singer show that the optimal α and β depend only on the weights of the true positives (W_+^+), false positives (W_+^-), true negatives (W_-^-) and false negatives (W_-^+) for that threshold and so can be computed directly from these values. The equations for them are given below:

$$W_+^+ = \sum_{i: y_i = +1 \wedge \mathbf{f}^T \mathbf{x_i} > \theta} D(i) \tag{4}$$

$$W_+^- = \sum_{i: y_i = +1 \wedge \mathbf{f}^T \mathbf{x_i} \leq \theta} D(i) \tag{5}$$

$$W_-^- = \sum_{i: y_i = -1 \wedge \mathbf{f}^T \mathbf{x_i} \leq \theta} D(i) \tag{6}$$

$$W_-^+ = \sum_{i: y_i = -1 \wedge \mathbf{f}^T \mathbf{x_i} > \theta} D(i) \tag{7}$$

$$\alpha = \frac{1}{2} \log \frac{W_+^+}{W_-^+} \tag{8}$$

$$\beta = \frac{1}{2} \log \frac{W_+^-}{W_-^-} \tag{9}$$

These values can be computed easily since the responses are sorted. Note that each response also has associated with it the weight and label of its example. After each possible θ (along with the corresponding optimal α and β) is tested, the parameters with lowest Z score are returned.

We will now focus our discussion on different algorithms we used for hyperplane selection.

4.1. Random hyperplanes

Our first weak learner selection algorithm is the same brute force algorithm used in Viola-Jones. We expect this weak learner to perform poorly but try it anyway for completeness. A random subsample of all possible hyperplanes is generated to yield a set of 20,000 hyperplanes. Each one is then tested by finding the optimal θ, α and β as described above and its Z score is computed. The hyperplane, θ, α and β with minimum Z score are selected as the weak classifier, $h(\mathbf{x})$.

4.2. Weighted least squares

The next weak learner selection algorithm we try is weighted least squares (WLS). The problem is formulated in the traditional least squares regression framework, where the data labels are the desired outputs $\mathbf{y} = [y_1, y_2, ..., y_n]^T$, of the projections of the images $X = [x_1 x_2 ... x_n]^T$ on the hyperplane $\mathbf{f}$, weighted by example importance. The weighting is incorporated into the least squares formulation by introduction of the the diagonal weight matrix $W = diag(D(1), D(2), ..., D(n))$, which weights each example by its importance.

$$WX\mathbf{f} = W\mathbf{y} \quad (10)$$

$$(WX)^T(WX)\mathbf{f} = (WX)^T W\mathbf{y} \quad (11)$$

$$\mathbf{f} = (X^T W^2 X)^{-1} X^T W^2 \mathbf{y} \quad (12)$$

The shortcoming of WLS is that correctly classified patches that yield a dot product larger than one, are penalized with square cost with respect to their distance from the target label. This motivates us to use margin based methods which take advantage of correctly classified examples with a dot product larger than it's label.

4.3. Gradient descent

We have also developed a new weak learning algorithm that chooses a hyperplane that directly minimizes Z_k. Since this is the same cost function that AdaBoost minimizes, it would seem to be the best choice. Friedman [2] proposes a similar gradient based approach. However, the gradient boosting algorithm proposed in his paper minimizes a different loss function. In the following explanation, we drop the subscript k since it is clear that we are minimizing Z at a particular boosting iteration, k. This minimization is equivalent to maximization of the weighted exponentiated negative margin.

To minimize Z we use gradient descent. Since Z is a linear combination of exponentiated threshold functions $h(\mathbf{x})$, we cannot directly take derivatives of it. We therefore approximate it using a linear combination of exponentiated sigmoid functions (eq 19).

To approximate $h(\mathbf{x})$ we will scale the sigmoid function assuming $\alpha > \beta$. Notice that if $\beta > \alpha$ we flip the hyperplane (by setting $\mathbf{f} \leftarrow -\mathbf{f}$) and get the desired contraint.

Using a smoothness constant $c \in \mathcal{R}$, we define:

$$S(x) = \frac{1}{1+e^{-x}} \quad (13)$$

$$s_i = S(c(\mathbf{f}^T\mathbf{x_i} - \theta)) \quad (14)$$

We then approximate $h(\mathbf{x})$ as $\tilde{h}(\mathbf{x})$ by rescaling and shifting the sigmoid response function from the range $(0, 1)$ to the range (β, α):

$$\tilde{h}(\mathbf{x}_i) = \beta + (\alpha - \beta)s_i \quad (15)$$

$$= \beta + \frac{\alpha - \beta}{1 + e^{-c(\mathbf{f}^T\mathbf{x_i} - \theta)}} \quad (16)$$

The function $\tilde{h}(\mathbf{x})$ is a smooth approximation to the step function $h(\mathbf{x})$.

For convinience we define $\tilde{z}_i$ as:

$$\tilde{z}_i = D(i)e^{-y_i(\beta + (\alpha-\beta)s_i)} \quad (17)$$

Now that we have a smooth approximation of the step function, we are able to take derivatives of the estimated $\tilde{Z}$ with respect to the hyperplane $\mathbf{f}$. We would like to find $\mathbf{f}, \theta, \alpha, \beta$ that minimize $\tilde{Z}$.

$$\tilde{Z} = \sum_{i=1}^{n} D(i)e^{-y_i(\beta + (\alpha-\beta)s_i)} \quad (18)$$

$$= \sum_{i=1}^{n} \tilde{z}_i \quad (19)$$

We take derivatives of $\tilde{Z}$ with respect to $\mathbf{f}$ at each of the data points. Since the derivative in 22 cannot be analytically solved when setting to zero, we have to take a gradient descent approach to minimize $\tilde{Z}$. We will iteratively take a steps of size η against the gradient to minimize $\tilde{Z}$:

1. Take derivative with respect to the hyperplane

$$\frac{\partial \tilde{z}_i}{\partial \mathbf{f}} = z_i \frac{\partial}{\partial \mathbf{f}} \left\{ -y_i \left(\beta + \frac{\alpha - \beta}{1 + e^{-c(\mathbf{f}^T\mathbf{x_i} - \theta)}} \right) \right\} \quad (20)$$

$$= z_i \frac{\partial}{\partial \mathbf{f}} \{ -y_i(\beta + (\alpha - \beta)s_i) \} \quad (21)$$

$$= -c(\alpha - \beta) y_i \tilde{z}_i s_i (1 - s_i)\mathbf{x}_i \quad (22)$$

2. Update the current hyperplane

$$\mathbf{f} \leftarrow \mathbf{f} - \eta \sum_{i=1}^{n} \frac{\partial \tilde{z}_i}{\partial \mathbf{f}} \quad (23)$$

3. Normalize the updated hyperplane

$$\mathbf{f} \leftarrow \frac{\mathbf{f}}{||\mathbf{f}||} \tag{24}$$

4. Update parameters α, β, θ to minimize Z given the new hyperplane $\mathbf{f}$

During the gradient descent, the hyperplane has a tendency to grow. Its $L2$ norm grows to 10 in 100 iterations. The growth in the hyperplane's norm only affects the threshold in terms of classification. However, when the hyperplane's norm grows, it effectively causes the algorithm to take larger and larger steps thereby causing oscillations which prohibit convergence. Normalizing the hyperplane ensures that the algorithm does not oscillate given a small enough step size η. The sigmoid smoothing parameter c also has a dual effect. According to equation 16 it controls the smoothness of the approximation surface $\tilde{Z}$, and according to 22 it controls the size of the gradient steps. Choosing higher values for c would yield better approximations of Z however, the resulting surface would become less smooth and the optimization would be more likely to get stuck in a local optimum.

Given the new hyperplane $\mathbf{f}$ we would like to find the new triplet $\{\theta, \alpha, \beta\}$ that minimize the real Z, since these parameters are no longer correct for the new hyperplane. We use the method described at the beginning of this section following equations 8, 9 to set alpha and beta for each possible threshold. We then select the triplet that minimizes Z.

This gradient descent process is iterated until there is no more improvement in Z in successive gradient descent steps. The resulting hyperplane $\mathbf{f}$ and parameters θ, α, β are passed back to Adaboost as the weak learner. The dataset gets reweighted and a new starting hyperplane is selected.

Because gradient descent may converge to a bad local minima, we run gradient descent using 10 different random starting hyperplanes. The final hyperplane with lowest Z score from each of these runs is returned.

5. Experiments

Our training set contains 3000 frontal face images and 10000 non-face images collected from the World Wide Web. Each face image is scaled and cropped to 24x24 pixels. Each non-face is a randomly selected patch from a larger image and is also scaled to 24x24 pixels. Each example image x_i is regarded as a 576 dimensional vector $x_i \in \mathcal{R}^{576}$. The corresponding labels $y_i \in \{-1, +1\}$ are equal to -1 for non face patches and $+1$ for face patches.

A classifier with 50 features was trained using each of the weak learners described above. We found good settings for c and η by trial and error.

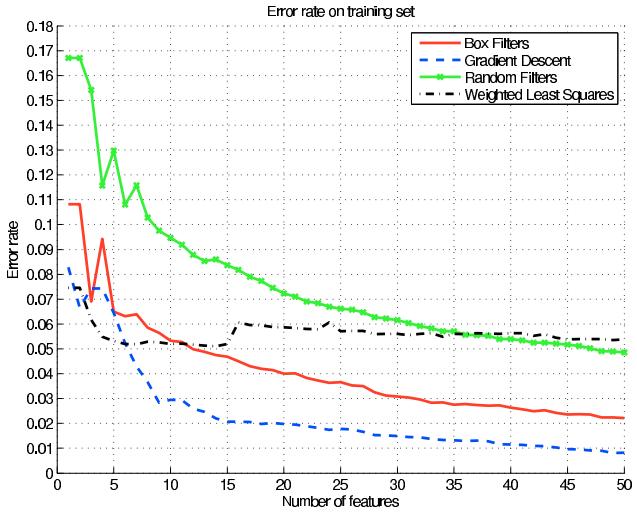

Figure 3. (a)Training error for different search algorithms

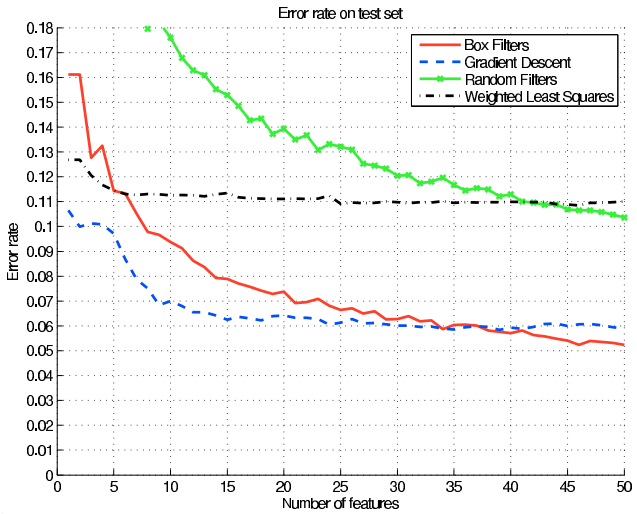

Figure 4. (a)Test error for different search algorithms

A 50 rectangle feature classifier was also trained using the brute force weak learner as in [10]. The pool of rectangle filters that the weak learner could choose from consisted of 26,365 rectangle filters of the types shown in figure 1.

5.1. Results

Each classifier was tested on a test set of 9832 faces and 50,000 non-faces. Like the training set, each example image was of size 24x24 pixels.

We define the error rate as the weight of misclassified examples divided by the total weight of all examples where the total weight of positives examples equals the total weight of negative examples.

The error rate on the training set versus the number of features is shown in figure 3. The error rate on the test set versus the number of features is shown in figure 4. A ROC

curve plotting false positive rate versus false negative rate on the test set is shown in figure 5. For the ROC curves, only 10 features were used in each classifier. This is because on a typical face detector cascade, the average number of features computed per patch is less than 10 so the first 10 features really determine the speed of the classifier.

As expected, the brute force weak learner that uses a random sample of hyperplanes, has low accuracy and is significantly worse than the rectangle feature classifier.

WLS has fairly low error after the first feature, however, it quickly falls behind the rectangle feature classifier as more features are selected. We suspect that the main reason for this behavior is the squared penalty term that penalizes correct detections with high margin.

The gradient descent weak learner does achieve a significant improvement over the rectangle feature classifier at least until about the 30th feature. In terms of their ROC curves, the thresholded hyperplanes from the gradient descent weak learner achieve an equal error rate (where the false positive rate equals the false negative rate) of about 6.8 % while the rectangle features have an equal error rate of about 9.4 %. The main effect of this improved accuracy in the beginning stages of a cascade are on the average number of features computed per image patch. For example the gradient descent based classifier after one weak classifier has about the same error on the test set as the rectangle feature classifier after about 7 weak classifiers. The rectangle feature classifier takes about 23 weak classifiers to acheive the same error rate as the gradient descent based classifier after only 10 hyperplanes. In practice, this means that many fewer thresholded hyperplanes are needed to achieve the same error rate as the rectangle features. This leads to a significant reduction in the average number of features computed per patch in a cascaded detector.

It is also interesting to visualize the first few hyperplanes selected by each of the weak learners and the first few rectangle filters selected. The first few hyperplanes chosen by the gradient descent weak learner look distinctly face-like as shown in figure 6. The hyperplane chosen by weighted least squares has very little face-like structure which also helps explain why it generalizes more poorly.

5.2. Hardware oriented hybrid detectors

In practice most of the computation of a cascaded face detector is spent on the first 10 or so features. Therefore, using on average less features at the beginning of the cascade while achieving the same error rate will greatly increase the speed of the detector. This is what is exactly what we get by using the gradient descent based detector presented here when concatenated to an existing rectangle filter based detector. Because general dot products can be very fast in hardware, these results have greatest importance for hardware implementations of face detectors which are becoming

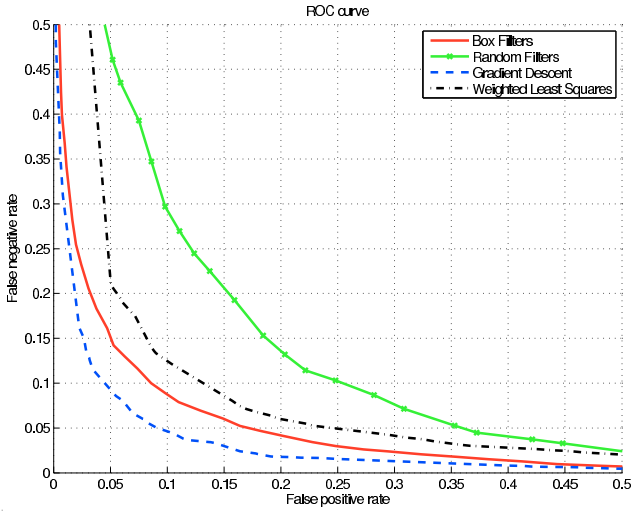

Figure 5. (a)ROC curve for different search algorithms

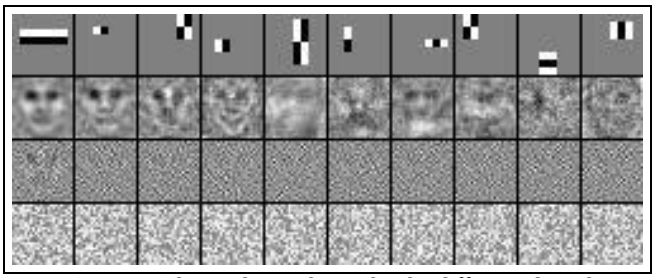

Figure 6. First 10 hyperplanes chosen by the different algorithms. Each row represents a different selection algorithm. The features are sorted by selection order from left to right. (row 1)Viola and Jones box-filters (row 2) Gradient descent (row 3) Weighted least squares (row 4) Random hyperplanes

quite common.

To build a full face detector, many more weak classifiers would have to be learned than just the first 50. The learning process also requires some form of resampling to generate more difficult non-face patches. Instead of training a full gradient descent based detector, we created a hybrid detector. This was done by adding the first 10 thresholded hyperplanes learned using the gradient descent to the beginning of an existing rectangle filter based detector. The original box filter based detector has 1520 rectangle features. Thus, the hybrid detector has 1530 features. We tested the average number of weak classifiers evaluated per candidate patch for such a hybrid detector. On the MIT+CMU test set [9, 7], which has 130 images and 507 faces, the box filter based detector at a 88% detection rate and a 1/1,014,170 false positive rate evaluates on average 9.9 features per patch. The hybrid detector at a 88% detection rate and a 1/1,029,310 false positive rate evaluates on average 8.0 features per patch. Therefore, for approximately the same point on the ROC curve the hybrid detector yields almost a 20% speed up in terms of the number of features evaluated per patch. Some

Figure 7. Example of detections on a low contrast image

detection results for the hybrid detector on difficult images from the MIT+CMU test set are shown in figures 7 and 8. These show detections on a low contrast image an image with a variety of different types of faces and slightly different poses.

6. Conclusions

Haar-like features are very difficult to improve on for general purpose computers in terms of their ratio of classification accuracy over computational cost. However, for specialized hardware in which dot products are very fast, the costs change, and more powerful features such as real hyperplanes (dot products) can be just as cheap to compute. For such cases, we have shown that real hyperplane features can lead to significant improvements over Haar-like features. Thus, the main contributions of this paper are a new gradient descent weak learning algorithm for unrestricted thresholded hyperplanes and the demonstration that such weak classifiers can lead to a significant speed-up in a cascaded face detector in terms of number of features computed per patch.

Figure 8. Example of detections on a wide variety of faces

References

[1] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23–37. Springer-Verlag, 1995. 2

[2] J. Friedman. Greedy function approximation: a gradient boosting machine, 1999. 4

[3] C. Huang, H. Ai, Y. Li, and S. Lao. Learning sparse features in granular space for multi-view face detection. In *Proc. of IEEE International conference on Automatic Face and Gesture Recognition*, Southampton, UK, April 2006. 2

[4] S. Li and Z. Zhang. Floatboost learning and statistical face detection, 2004. 2

[5] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *International Conference on Image Processing*, pages I: 900–903, 2002. 1, 2

[6] C. Liu and H. Shum. Kullback-leibler boosting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 587–594, 2003. 1

[7] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In *IEEE Patt. Anal. Mach. Intell.*, volume 20, pages 22–38, 1998. 6

[8] R. Schapire and Y. Singer. Improving boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3), 1999. 2, 3

[9] K. Sung and T. Poggio. Example-based learning for view-based face detection. In *IEEE Patt. Anal. Mach. Intell.*, volume 20, pages 39–51, 1998. 6

[10] P. Viola and M. Jones. Robust real-time face detection. *Int. J. Computer Vision*, 57:137–154, 2004. 1, 2, 5