

Vector Boosting for Rotation Invariant Multi-View Face Detection

Chang HUANG¹, Haizhou AI¹, Yuan LI¹ and Shihong LAO²

¹Computer Science and Technology Department, Tsinghua University, Beijing, 100084, China

²Sensing Technology Laboratory, Omron Corporation

E-mail: ahz@mail.tsinghua.edu.cn

Abstract

In this paper, we propose a novel tree-structured multi-view face detector (MVFD), which adopts the coarse-to-fine strategy to divide the entire face space into smaller and smaller subspaces. For this purpose, a newly extended boosting algorithm named Vector Boosting is developed to train the predictors for the branching nodes of the tree that have multi-components outputs as vectors. Our MVFD covers a large range of the face space, say, $\pm 45^\circ$ rotation in plane (RIP) and $\pm 90^\circ$ rotation off plane (ROP), and achieves high accuracy and amazing speed (about 40 ms per frame on a 320×240 video sequence) compared with previous published works. As a result, by simply rotating the detector 90° , 180° and 270° , a rotation invariant (360° RIP) MVFD is implemented that achieves real time performance (11 fps on a 320×240 video sequence) with high accuracy.

1. Introduction

Some early works on face detection, for instance, Rowley's ANN method [1] and Schneiderman's method based on Bayesian decision rule [2], have achieved high accuracy, but their applications are very limited due to the tremendous computational load. The breakthrough happened in 2001 when Viola and Jones [3] developed their cascade detector that firstly showed real-time performance for frontal face detection. There have been many related works such as Xiao et al.'s Boosting Chain [4] and Liu et al.'s Kullback-Leibler Boosting [5]. They focused on some parts of Viola's framework and adopt new methods for improvements. Moreover, to detect faces with various poses, Li proposed a pyramid-structured MVFD [6], while Jones and Viola used decision-tree method [7] to handle the RIP problem. All these works benefit from the fast Haar-like feature extraction with the integral image, and adopt the AdaBoost algorithm to guarantee the correct rate of classification.

However, some challenging requirements of many complicated practical applications still can not be met due to the ubiquitous concomitance of RIP and ROP in

real-life faces. Although this problem can be solved by rotating the image and apply MVFD several times, it significantly increases the computation complexity and false alarms. So we need to develop some special techniques to improve the efficiency of the detector.

In this paper, an efficient tree-structured MVFD is proposed to deal with both kinds of pose changes in a unified framework, which covers $\pm 45^\circ$ RIP and $\pm 90^\circ$ ROP. It is so powerful that only such four detectors are needed to handle the rotation invariant MVFD problem.

The main contributions of this paper are: 1) different from both the pyramid structure in [6] and the decision tree in [7], a WFS (Width-First-Search) tree structure is adopted to achieve higher performance in both speed and accuracy; 2) to train branching nodes of the WFS tree, the Vector Boosting algorithm is extended from the Real AdaBoost [9] [10] [11], which achieves better performance; 3) the piece-wise function, which is implemented with Look Up Table (LUT) method, is suggested to approximate complex distributions and give confidence-rated hypothesis as the weak classifier.

The rest of this paper is organized as follow: Section 2 presents the WFS tree structure; Section 3 introduces the Vector Boosting algorithm; Section 4 proposes the powerful piece-wise function; Section 5 deals with the optimization problem of the weak learner in Vector Boosting; Section 6 gives the results of a series of experiments, and Section 7 comes to the conclusions.

2. WFS Tree-Structured Detector

In recent years, Viola and Jones's framework on face detection has been proved very successful and efficient. For the MVFD, the most straightforward way of extending their framework is to train different cascades individually for each view and then use them as a whole like figure 1(a). The work in [8] has proved that even this simple method does have rather good performance in dealing with this complicated problem. Anyway there still remains a large capacity to improve through a better structure.

One approach is the pyramid structure [6] that adopts coarse-to-fine strategy to handle pose variance

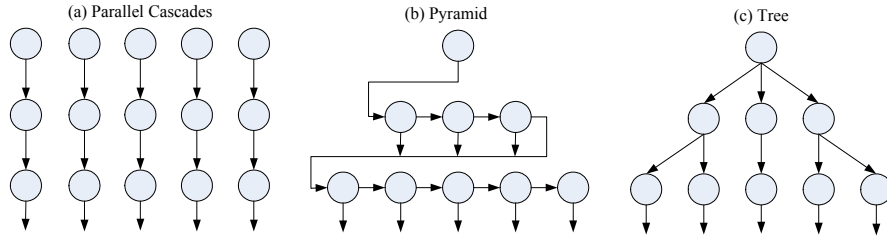


Figure 1. Different structures of MVFD

of ROP (Figure 1(b)). Due to the similarities that exist in different poses of faces, the pyramid method treats them as one ensemble positive class so as to improve the efficiency of extracted features. However, the neglect of their intrinsic diversities makes the pyramid method have no discrimination in different poses. As a result, a sample that has passed the parent node has to be sent to all its child nodes (See figure 1(b)), which considerably slows down the decision-making process.

On the contrary, another approach, the decision tree method [7], puts emphasis upon the diversities between different poses and the tree works as a pose estimator. With the imperative judgments made by the decision tree, it significantly reduces the time spent on pose estimation. However the results are somewhat unstable that makes its generalization ability not so well.

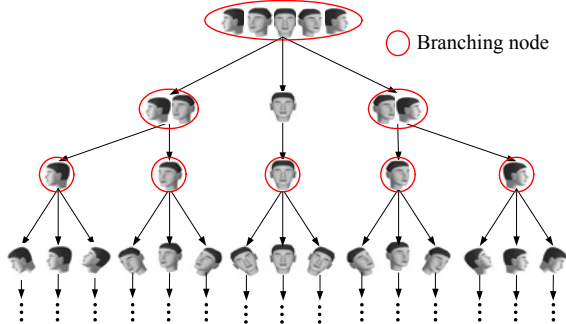


Figure 2. The Structure of the WFS Tree

As a matter of fact, there are two main tasks for MVFD problem: one is to distinguish between faces and non-faces; the other is to identify the pose of a face. The first task needs to reject non-faces as quickly as possible, so it is inclined to find the similarities of faces of different poses so as to separate them from non faces, while the latter task focuses on the diversities between different poses. The conflict between the two tasks really leads to the dilemma that either treating all faces as a single class (as in the pyramid method) or different individually separated classes (as in the decision tree method) is unsatisfactory for MVFD problem. Hence we propose a WFS tree structure to balance these two aspects so as to enhance the detector in both accuracy and speed.

Covering the face space with $\pm 45^\circ$ RIP and $\pm 90^\circ$ ROP, the proposed detector tree adopts the coarse to

fine strategy to divide the entire face space into smaller and smaller subspaces as shown in figure 2. The root node that covers the largest space is partitioned into left profile, frontal and right profile views in the second layer; to describe the ROP more accurately, full-profile and half-profile views are defined in the coming layer; finally in the fourth layer, each view is split into three categories according to their different RIP.

Our tree-structured detector will not make exclusive selection of the path for a sample like the decision tree method. Instead, each branching node computes a determinative vector $\mathbf{G}(\mathbf{x})$ for the decision which child nodes the sample should be sent to. For example, in the root branching node of figure 2, a sample making $\mathbf{G}(\mathbf{x}) = (1, 1, 0)$ means it may be a left profile face or a frontal one but can not be a right profile one, so in the second layer, it will be sent to the left node and the middle node but not the right one. Another sample that makes $\mathbf{G}(\mathbf{x}) = (0, 0, 0)$ is thought of outlying from any child node and will be rejected immediately. To browse all active nodes in the tree, the Width-First-Search strategy is adopted, and its pseudo code is given below.

-
0. (Input) Given a sample \mathbf{x} and the constructed tree detector T .
 1. (Initialization) Set the node list L empty; push the root node of T into L ; empty the output list O .
 2. (WFS procedure)
 - While L is not empty, do
 - Pop the first node d from L .
 - Calculate the determinative vector $\mathbf{G}^{(d)}(\mathbf{x})$, where $\mathbf{G}^{(d)}(\mathbf{x}) = [g_1^{(d)}(\mathbf{x}), \dots, g_n^{(d)}(\mathbf{x})]$
 - For $t=1, \dots, n$:
 - If $g_t^{(d)}(\mathbf{x}) = 1$
 - Get the t -th child node s_t of d
 - If s_t is a leaf node
 - Push l_t , the label of s_t , into the list O .
 - Else
 - Push s_t into the list L .
 - End if
 - End if
 - End for
 - End do
 - 3. (Output) Output all labels in the list O for sample \mathbf{x} .

Figure 3. Width-First-Search in the detector tree

In fact, the WFS tree method does not try to give quick pose estimation like [7] which amounts to loss in accuracy, nor simply merges different poses without consideration of their in-class differences like [6] which amounts to loss in speed. Hence, the WFS tree can outperform them by means of paying attention to both diversities and similarities between various poses that guarantees both high accuracy and faster speed.

From the discussion above, it is obvious that the branching node plays an important role in the WFS tree. To support this new technique, in the next section, a newly extended version of the Real AdaBoost in [9] that is named Vector Boosting will be introduced. With its help, the branching nodes are trained for the required determinative vector $\mathbf{G}(\mathbf{x})$ fast and accurately.

3. Vector Boosting

The Vector Boosting is proposed as an extended version of the Real AdaBoost in which both its weak learner and its final output are vectors rather than scalars. The original inspiration of Vector Boosting is drawn from the multi-class multi-label (MCML) version of the Real AdaBoost [9], which assigns a set of labels for each sample and decomposes the original problem into k orthogonal binary ones. The major problem of this algorithm is that for each binary classification problem, a sample is regarded as either positive or negative. However in many complicated cases, it is not tenable since some samples are neither positive nor negative for certain binary classification problems of which they are independent, which makes the MCML version of Real AdaBoost inapplicable. Take the root node of the WFS tree in figure 2 for example. A frontal face sample that makes the determinative vector $\mathbf{G}(\mathbf{x}) = (*, +1, *)$ be acceptable ($*$ can be either $+1$ or -1), i.e., the first (i.e. for left profile face) and the third (i.e. for right profile face) binary classifications are independent of frontal face samples.

Once a complicated classification problem is decomposed into a set of binary ones, Vector Boosting will deal with them in a unified framework by means of a shared output space of multi-components vector. Each binary problem has its own “interested” direction in this output space that is denoted as its projection vector. In this way, different binary problems are not necessarily independent (with orthogonal projection vectors); they could also be correlated (with non-orthogonal projection vectors) in general.

Let $S = \{(\mathbf{x}_1, \mathbf{v}_1, y_1), \dots, (\mathbf{x}_m, \mathbf{v}_m, y_m)\}$ be a sequence of training samples where \mathbf{x}_i belongs to an instance space \mathcal{X} , \mathbf{v}_i belongs to a finite k -dimensional projection vector

set Ω and the label $y_i = \pm 1$ (i.e. positive or negative). We present the pseudo code of a generalized version of k -dimensional Vector Boosting in figure 4, which deals with n binary classification problems synchronously.

For a classification problem that has been decomposed into n binary ones, given:

(1) Projection vector set $\Omega = \{\omega_1, \dots, \omega_n\}, \omega_i \in \mathbb{R}^k$

(2) Sample set $S = \{(\mathbf{x}_1, \mathbf{v}_1, y_1), \dots, (\mathbf{x}_m, \mathbf{v}_m, y_m)\}$,
where $\mathbf{x} \in \mathcal{X}$, $\mathbf{v} \in \Omega$ and its label $y = \pm 1$.

- Initialize the sample distribution $D_1(i)=1/m$,
- For $t = 1, \dots, T$
 - ◇ Under current distribution, train a weak classifier

$$\mathbf{h}_t(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}^k. \quad (\text{weak learner})$$

- ◇ Update the sample distribution

$$D_{t+1}(i) = \frac{D_t(i) \exp[-y_i (\mathbf{v}_i \cdot \mathbf{h}_t(\mathbf{x}_i))]}{Z_t}, \quad (1)$$

where Z_t is the normalization factor so as to keep D_{t+1} as a probability distribution.

- The final output space is $\mathbf{H}(\mathbf{x}) = \sum_{t=1}^T \mathbf{h}_t(\mathbf{x})$ (2)

- The confidence space is $\mathbf{F}(\mathbf{x}) = \mathbf{A}\mathbf{H}(\mathbf{x})$. (3)

where the transformation matrix $\mathbf{A} = [\omega_1, \dots, \omega_n]^T$.

- The final strong classifier is

$$\mathbf{G}(\mathbf{x}) = \text{sign}(\mathbf{F}(\mathbf{x}) - \mathbf{B}). \quad (4)$$

\mathbf{B} is the threshold vector whose default value is zero.

Figure 4. A generalized version of Vector Boosting

It is configured in figure 4 to handle a complicated problem in a k -dimensional output space, which has been decomposed into n binary ones. In the same way of Real AdaBoost, the weak learner is called repeatedly under the updated distribution to form a highly accurate classifier. It should be mentioned that in the kernel update rule (equation 1), the margin of a sample \mathbf{x}_i with its label y and projection vector \mathbf{v}_i is defined as $y_i(\mathbf{v}_i \cdot \mathbf{h}_t(\mathbf{x}_i))$ due to the vectorization of the output. Thus, the orthogonal component of a weak classifier’s output makes no contribution to the updating of the sample’s weight. In this way, Vector Boosting increases the weights of samples that have been wrongly classified according to the projection vector (in its “interested” direction) and decrease those with correct predictions.

The final output is the linear combination of all trained weak classifiers (equation 2). To calculate the confidences for each binary problem, a $n \times k$ matrix \mathbf{A} is employed to transform the k -dimensional output space into n -dimensional confidence space (equation 3), which is made up of all n projection vectors in set Ω . Each dimension of the confidence space corresponds to a certain binary problem. Finally, the strong classifier

with Boolean outputs is got with the threshold vector \mathbf{B} (equation 4).

Practically in our experiments, we still decompose the complicated classification problem into several orthogonal binary ones for simplification but maintain the independences between various face poses. For example, when training the root branching node in figure 2, we collect three face sets, which are left profile, frontal and right profile, assign their projection vectors as $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ respectively, and label them positive (i.e. $y_i = +1$). After that, prepare another three non-face sets with the same projection vectors but label them negative (i.e. $y_i = -1$). These three binary problems appear to be trained independently, but share the selected features so as to outperform the individually training methods such as the work in [8]. Figure 5 draws the results of three classes in the output space $\mathbf{H}(\mathbf{x})$, where $H(\mathbf{x}, 1)$ is the dimension for left profile view and $H(\mathbf{x}, 3)$ is for right profile view. (The frontal view and its corresponding projection vector are omitted here for clarity). It is true that both left profile and right profile faces can be well separated from the non faces in this 2-D space with their own projection vectors ω_1 and ω_3 .

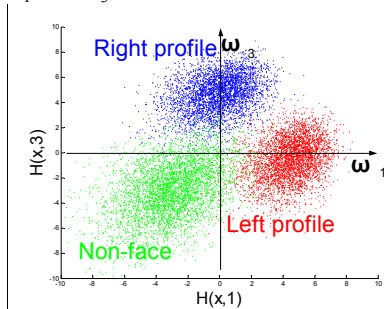


Figure 5. Distributions of 3 classes in the output space

It is easy to see that when $n = k = 1$ and ω_1 (the only projection vector in Ω) is a unit vector, Vector Boosting is exactly the same as Real AdaBoost. In fact, due to the consistency in updating rule (equation 1), Vector Boosting keeps the same training error bound as Real AdaBoost, that is,

$$\text{training error } P_{\text{error}} \leq \Pi Z_n. \quad (5)$$

Its proof can be found in Appendix A.

In the next two sections, the design of the weak classifier and its optimization method for Vector Boosting are described.

4. Piece-wise Functions

In Viola and Jones' cascade detector [3], they use integral image method to fast calculate the Haar-like feature $f(\mathbf{x})$, on which a weak classifier $h(\mathbf{x})$ is defined as a threshold-type function with Boolean output as

shown in figure 7(a). It can be formally expressed as $h(\mathbf{x}) = \text{sign}[f(\mathbf{x}) - b]$, where b is a threshold. Although it is simple and easy to train, it is not able to take full advantage of information from the extracted feature. For example, as shown in the left of figure 6, the positive and negative samples can be well classified by a proper threshold. However, it cannot describe their divergences accurately enough due to its coarse granularity. In addition, during the updating process of the AdaBoost algorithm, their divergences diminishes continually (e.g. the right diagram of figure 6 shows the distributions on a Haar-like feature selected in the fifth round), which largely weakens the discrimination power of the threshold-type function, and sometimes even hampers the convergence of the algorithm (e.g. in the latter layer of the cascade when the face and non-face patterns are very similar). In conclusion, the coarse granularity of the threshold-type weak classifier greatly impedes the improvement of the speed and accuracy of the detector.

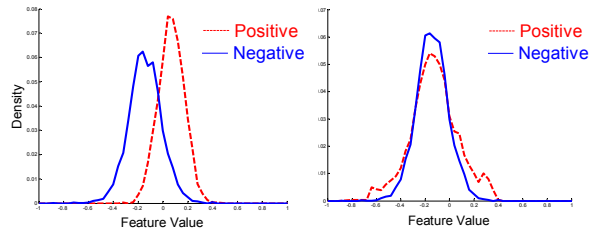


Figure 6. Distributions on Haar-like features

A more efficient design for weak classifiers, the piece-wise function illustrated in figure 7(b), divides the feature space into many bins with equal width and output a constant value for each bin. The piece-wise function is able to approximate various distributions more accurately without the constraint of Boolean output, which is essentially a symmetrical equidistant sampling process. It also meets the very requirements of the weak learner in Real AdaBoost since it is really a natural way of domain partition referred in [9]. Besides, as the piece-wise function can be efficiently implemented with Look Up Table (LUT), it does not bring in more computational load compared with the threshold-type function (only a multiplication is needed to convert feature value to the index of LUT), but it significantly enhances the capability of the weak classifier by means of real-valued outputs.

According to "There is no free lunch" theorem, it is very important to choose suitable granularity for a piece-wise function. The finer the granularity is, the more accurately the function can estimate (lower training error), but the more sensitive it will be to the noise and the size of training set (higher structural risk). Empirically, it is strongly suggested that in the first several layers, the granularity can be a little finer so as

to make the training converge as fast as possible, while in the latter layers, it should be coarser to make the classification results robust.

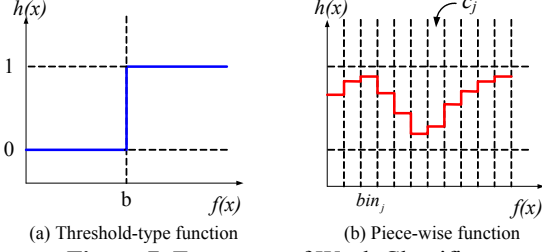


Figure 7. Two types of Weak Classifier

5. The Optimization of Weak Learner

Reviewing equation 3, on the basis of that the training error in Vector Boosting is bounded by the product of all normalization factors, the objective of weak learner is just to minimize this factor of current round if adopting greedy strategy. Suppose a weak classifier $\mathbf{h}(\mathbf{x}; \theta, \mu)$ is characterized by two parameters: θ specifies its Haar-like feature and μ specifies its piece-wise function. Following Viola's exhaustive search method in [3], we enumerate a finite redundant Haar-like feature set and optimize a piece-wise function for each feature so as to obtain the most discriminating one. Then the only unsolved problem in weak learner is how to get the optimal piece-wise function μ .

A piece-wise function is configured by two parts: one is the division of feature space, the other is the constant for each division (i.e. bin). For simplification, the first one is fixed for each feature empirically, and the last one, the output constant for each bin, can be optimized as follows.

Suppose samples $S = \{(\mathbf{x}_1, \mathbf{v}_1, y_1), \dots, (\mathbf{x}_m, \mathbf{v}_m, y_m)\}$ is under the distribution of $D_t(i)$. On a certain feature $f(\mathbf{x})$, they can be grouped into many bins with the predefined division of piece-wise function. Denote them as:

$$S_j = \{(\mathbf{x}_i, \mathbf{v}_i, y_i) \mid \mathbf{x}_i \in \text{bin}_j\}, \text{ where } j \text{ is the bin index.}$$

Let c_j be the constant output for the j -th bin, and then we can get the normalization factor

$$Z_t = \sum_j \sum_{S_j} D_t(i) \exp(-y_i (\mathbf{v}_i \cdot \mathbf{c}_j)) \quad (6)$$

For group k (bin k), the training loss is

$$\text{loss}_k(\mathbf{c}_k) = \sum_{S_k} D_t(i) \exp(-y_i (\mathbf{v}_i \cdot \mathbf{c}_k)), \quad (7)$$

It is not difficult to verify that this loss function is convex with its independent variable \mathbf{c}_k . Hence, each \mathbf{c}_j can be easily optimized with a proper optimization algorithm such as Newton-Step method.

The weak learner procedure can be summarized in the following pseudo code.

Given: Sample set $S = \{(\mathbf{x}_1, \mathbf{v}_1, y_1), \dots, (\mathbf{x}_m, \mathbf{v}_m, y_m)\}$,

finite feature set $P = \{f_k(\mathbf{x}; \theta_k)\}$, $1 \leq k \leq n$,

and the current distribution $D_t(i)$.

- For $k = 1, \dots, n$ (each feature)
 - ✧ Group all samples into different bins
$$S_j = \{(\mathbf{x}_i, \mathbf{v}_i, y_i) \mid \mathbf{x}_i \in \text{bin}_j\}, 1 \leq j \leq p$$
 - ✧ For $j=1, \dots, p$ (each bin)
 - ◆ Using Newton-step method to compute
$$\mathbf{c}_k^* = \arg \min_{\mathbf{c}_k} \left(\sum_{S_k} D_t(i) \exp(-y_i (\mathbf{v}_i \cdot \mathbf{c}_k)) \right) \quad (8)$$
 - ✧ Create weak classifier $\mathbf{h}(\mathbf{x}; \theta_k, \mu_k)$ based on $\{\mathbf{c}_k^*\}$, and use equation 5 to calculate its corresponding normalization factor $Z_t(\theta_k, \mu_k)$.
- Return the optimal weak classifier

$$\mathbf{h}_t^*(\mathbf{x}; \theta, \mu) = \arg \min_{\mathbf{h}(\mathbf{x}; \theta_k, \mu_k)} (Z_t(\theta_k, \mu_k)) \quad (9)$$

Figure 8. Weak learner with piece-wise function

In summary, the novel contributions of our MVFD include the WFS tree, the Vector Boosting algorithm and the weak classifiers based on piece-wise function. And our MVFD still benefits from Viola et al's cascade framework [3] and adopts some other techniques such as the nested weak classifiers in [8].

6. Experiments

To construct a WFS tree shown in figure 2, we first divide all faces into 5 categories according to left-right ROP, and then continue to split each category into 3 views, each of which takes charge of 30° RIP. Besides, each view covers $[-30^\circ, +30^\circ]$ up-down ROP for robustness. Consequently, our WFS tree covers 180° left-right ROP, 60° up-right ROP and 90° RIP.

The training process of the WFS tree is as follow

- Given: the false positive rate f and the detection rate d for each node; the expected overall false positive rate F for all views; the positive sample set P and the negative sample set N .
- Let current node E = the root node of the tree
- (Training procedure of the node E)
 - ✧ From P and N , collect samples that have passed all E 's parent nodes, forming two training set p and n with proper size. (Bootstrap)
 - ✧ With p and n , use the Vector Boosting to train a strong classifier $\mathbf{G}(\mathbf{x})$ until the required detection rate d and the false positive rate f are achieved.
 - ✧ Evaluate current overall false positive rate F_{cur} .
 - ✧ If $F_{\text{cur}} > F$

- ◆ For each of E 's child nodes E_{child} , let $E=E_{\text{child}}$ and recursively call the training procedure of the node E .
- ◇ Else
- ◆ Return current node E .

Figure 9. The training procedure of the WFS tree

We collected and labeled approximately 75000 face samples, including 30000 frontal ones, 25000 half profile ones and 20000 profile ones. Finally the WFS tree MVFD is composed of 204 nodes in 16 layers, totally including 7081 weak classifiers.



(a) frontal (b) half profile (c) full profile
Figure 10. Some training samples

6.1 Multi-View Face Detection Test

To give an overall evaluation, we test our MVFD on the CMU profile set, which consists of 208 images with 441 faces. Some of the detection results are shown in figure 12. In table 1 and figure 11, the results are compared with some previous published works, that is, Schneiderman's Bayesian decision rule method [2] and the Wu's individual cascade method [8]. Other related works such as Viola's decision tree [7] and Stan Z Li's pyramid method [6] are not compared due to they did not give full testing results on this set (Viola gave a ROC curve on the non-upright test set but not the CMU profile set).

In order to trade off between detection rate and false alarm, we use a control parameter that correspondingly adjusts the threshold vector \mathbf{B} of each node to give the whole ROC curve. According to table 1 and figure 11, our WFS tree achieves the best performance among these three methods.

On a Pentium 4, 2.8GHz PC, scaling the scanning window from 24×24 to 256×256 with scale ratio 1.25, our MVFD takes only about 40 ms on the detection of a 320×240 image. Compared with Wu's individual cascade method [8] that reported 80ms, the

consumed time is reduced by a half. The significant improvement is due to the WFS tree structure and the efficient training algorithm: Vector Boosting.

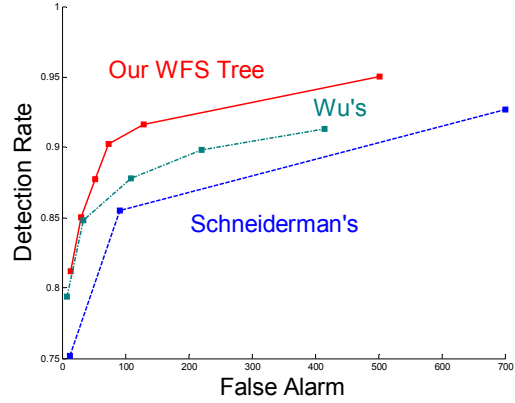


Figure 11. ROC curves on CMU profile set

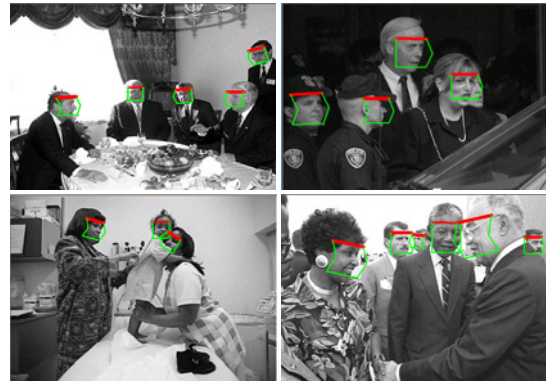


Figure 12. MVFD results on CMU profile set

6.2 Rotation Invariant MVFD Test

Since our MVFD covers $\pm 45^\circ$ RIP, we simply rotate it by 90° , 180° and 270° to construct three detectors so as to fully cover 360° RIP, and they work together to deal with the rotation invariant problem. Also on a Pentium 4, 2.8GHz PC, its speed is about 11 fps on a 320×240 video sequence, which is about 1.5 times faster than the method [8] (250ms per frame). It is not strange that the time consumed by 4 detectors is less than 4 times of one detector, because some time is spent on the integral image that needs only be

Table 1. MVFD results on CMU profile face test set. (Totally 208 images, 441 faces)

False Alarm Method	8	12	16	34	48	72	91	109	181	221	415	470	700
WFS tree			83.0%		88.0%	89.6%			92.3%			95.7%	
[8]	79.4%			84.8%				87.8%		89.8%	91.3%		
Schneiderman		75.2%			-		85.5%		-			-	92.7%

computed once.

Since so far there are no standard testing sets for this extremely challenging problem, we just show some detection results in figure 13.



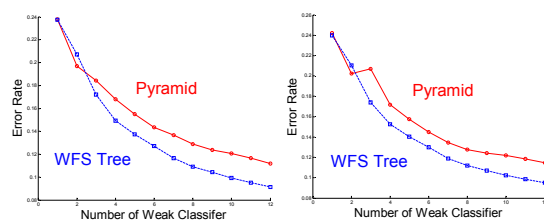
Figure 13. Rotation invariant MVFD results

6.3 Comparison with the Pyramid Structure

We have argued in section 2 that the WFS tree structure outperforms the pyramid structure for MVFD. Here we give a comparison experiment under the same condition to verify this argument and give some further analysis. In this experiment, with the same training samples, the same Haar-like feature set, the same type of weak classifiers (piece-wise functions), we train the root node shown in figure 2, which divides the coarsest view into three finer ones.

As explained in section 2 and 3, the WFS tree method considers these three categories independently and outputs a 3-dimensional vector as the final prediction. This contrasts with the pyramid method in which faces from different views are merged into an ensemble positive class and only scalar result is output. We recorded the convergences of both methods, which are shown in figure 14, where x-axis denotes the number of adopted weak classifiers, while y-axis denotes the error rate with the default threshold $\mathbf{B} = 0$.

According to this experiment, the WFS tree method converges faster than the pyramid method. Exactly, to achieve the same error rate, the WFS method needs only about two thirds of weak classifiers that required by the pyramid method, which means the consumed time in the root node can be reduced by one third if adopting the WFS tree method.



(a) Error rates on training set (b) Error rates on test set

Figure 14. WFS Tree vs. Pyramid

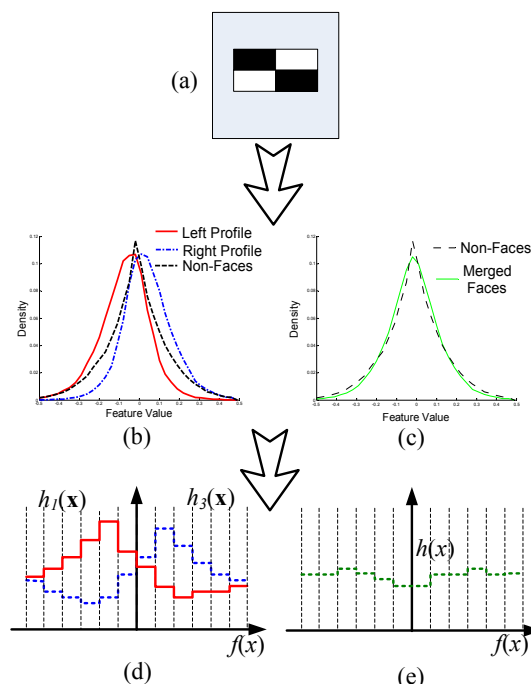


Figure 15. In-class differences have different affects on the Vector Boosting and Real AdaBoost

As a matter of fact, it is the Vector Boosting that results in such encouraging improvement. See figure 15 for example. Part (a) is a Haar-like feature selected in the second round by the Vector Boosting. For the WFS tree method, part (b) draws distributions of three classes on this feature: left profile, right profile and non faces (frontal faces are omitted again for clarity). The distribution of left profile faces is opposite to that of right profile faces since the feature in (a) is left-right symmetric. Both face categories can be well separated from non faces with the optimal vector weak classifier $\mathbf{h}(\mathbf{x})$ shown in part (d), in which $h(x,1)$ and $h(x,3)$ are its two corresponding components. On the contrary, if adopting the pyramid method, left and right profile faces are merged into a single positive class, whose distribution on the same feature is very similar with that of non-faces (See part (c)). Consequently, the optimal weak classifier $h(x)$ with Real AdaBoost in

part (e) is rather weak, which hardly provides evidences for the classification.

To sum up, the in-class differences between various poses can have significant bad effects on the decision made by scalar hypothesis that is trained by Real AdaBoost, but they have less effect on the vectorization one trained by Vector Boosting since the Vector Boosting takes into consideration of both between-class and in-class differences.

7. Conclusion

In this paper, we propose a WFS tree structure detector for the MVFD problem. In order to construct the WFS tree detector efficiently, two techniques are developed: 1) the Vector Boosting is derived from the Real AdaBoost to train branching nodes of WFS tree; 2) a piece-wise function is used to approximate complicated distributions. As far as we know, compared with previous published methods for MVFD problem, our WFS tree structure method achieves significant improvements in both speed and accuracy, and four such WFS tree-structured detectors worked together as a whole have made up of a rotation invariant MVFD that has achieved real-time performance for this extremely challenging problem the first time in the world.

Furthermore, the Vector Boosting algorithm configured in this paper can be regarded as a unified framework for the AdaBoost family, in which by means of properly predefined projection vectors it works exactly as the classical AdaBoost algorithm. Although it is developed for the MVFD problem, it could be applied to other complicated classification problems too.

8. Acknowledgements

This work is supported mainly by a grant from OMRON Corporation. It is also supported in part by National Science Foundation of China under grant No.60332010.

9. Reference

- [1] H. A. Rowley, "Neural Network-based Human Face Detection", Ph.D. thesis, Carnegie Mellon University, May 1999.
- [2] H. Schneiderman and T. Kanade, "A Statistical Method for 3D Object Detection Applied to Faces and Cars". CVPR 2000.
- [3] P. Viola, M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.
- [4] Rong Xiao, Long Zhu, Hongjiang Zhang, Boosting Chain Learning for Object Detection, ICCV 2003.
- [5] C.Liu and H. Y. Shum, "Kullback-Leibler Boosting", CVPR 2003.
- [6] S. Z. Li, L. Zhu, Z. Q. Zhang, et al., "Statistical Learning of Multi-View Face Detection". ECCV 2002.
- [7] Jones and Viola, "Fast Multi-view Face Detection". MERL-TR2003-96, July 2003.
- [8] Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao, Fast rotation invariant multi-view face detection based on real adaboost, FG 2004.
- [9] R. E. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-rated Predictions", Machine Learning, 37, 1999, 297-336.
- [10] Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm". In Proc. of the 13-th Conf. on Machine Learning, Morgan Kaufmann, 1996, 148-156.
- [11] R.E. Schapire, Y. Freund, P. Bartlett and W.S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods", The Annals of Statistics, 26(5), 1998, 1651-1686.

Appendix A: Training Error Bound of the Vector Boosting

Reviewing Section 4, for a sample \mathbf{x}_i , we only concern about its corresponding binary output, so the training error is:

$$P_{error} = \frac{1}{m} \sum_{i=1}^m \llbracket y_i \neq \mathbf{G}(\mathbf{x}_i, j) \rrbracket, \quad (10)$$

where j is the index of \mathbf{x}_i 's corresponding projection vector, and $\mathbf{G}(\mathbf{x}_i, j)$ is the concerned binary output.

Assume the threshold vector \mathbf{B} is zero as the default value, and then $G(\mathbf{x}_i, j) = \text{sign}(\mathbf{v}_i \cdot \mathbf{F}(\mathbf{x}_i))$.

Since $\llbracket y_i \neq G(\mathbf{x}_i, j) \rrbracket \leq \exp(-y_i(\mathbf{v}_i \cdot \mathbf{F}(\mathbf{x}_i)))$, plug it into equation (8), we get:

$$P_{error} \leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i(\mathbf{v}_i \cdot \mathbf{F}(\mathbf{x}_i))) \quad (11)$$

With the retrospection of the update rule in the Vector Boosting (equation 1), since $\mathbf{F}(\mathbf{x}_i) = \sum_{j=1}^l \mathbf{h}_j(\mathbf{x}_i)$,

a sample's training loss can be rewritten as:

$$\begin{aligned} \frac{1}{m} \exp(-y_i(\mathbf{v}_i \cdot \mathbf{F}(\mathbf{x}_i))) &= \frac{1}{m} \prod_{j=1}^l \exp(-y_i(\mathbf{v}_i \cdot \mathbf{h}_j(\mathbf{x}_i))) \\ &= D_{t+1}(i) \prod_{j=1}^l Z_j \end{aligned} \quad (12)$$

Now use (10) to substitute the right hand side of (9) and give the required results that

$$P_{error} \leq \sum_{i=1}^m D_{t+1}(i) \prod_{j=1}^l Z_j = \prod_{j=1}^l Z_j, \quad (13)$$

The last equation is satisfied as $\sum_{i=1}^m D_{t+1}(i) = 1$, which is the final probability distribution.