

Fast training and selection of Haar features using statistics in boosting-based face detection

Minh-Tri Pham Tat-Jen Cham
 School of Computer Engineering
 Nanyang Technological University
 Singapore
 {mtpham, astjcham}@ntu.edu.sg

Abstract

Training a cascade-based face detector using boosting and Haar features is computationally expensive, often requiring weeks on single CPU machines. The bottleneck is at training and selecting Haar features for a single weak classifier, currently in minutes. Traditional techniques for training a weak classifier usually run in $O(NT \log N)$, with N examples (approximately 10,000), and T features (approximately 40,000). We present a method to train a weak classifier in time $O(Nd^2 + T)$, where d is the number of pixels of the probed image sub-window (usually from 350 to 500), by using only the statistics of the weighted input data. Experimental results revealed a significantly reduced training time of a weak classifier to the order of seconds. In particular, this method suffers very minimal immersion in training time with very large increases in members of Haar features, enjoying a significant gain in accuracy, even with reduced training time.

1. Introduction

Face detection is one of the classic problems in computer vision, with applications in many areas like surveillance, robotics, multimedia processing, and HCI. Developing a face detection system is still an open problem, but there have been important successes over the past several years. A standard image-based approach to face detection is scanning in raster order every squared sub-window of d pixels over multiple image scales, and classifying if each sub-window contains a face or not. When this strategy is used, face detection becomes a *rare event binary classification* problem, in the sense that among millions of sub-windows, only a few contain faces.

Among the most influential image-based systems is the cascade-based face detector of Viola and Jones [13]. The

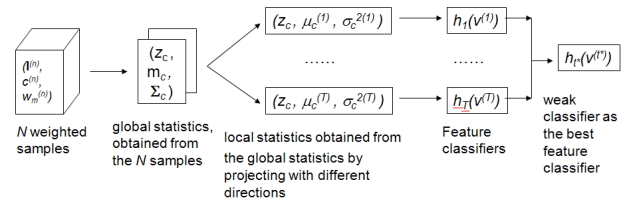


Figure 1. Diagram of this method to fast-train a weak classifier. The notations are explained in subsequent sections.

cascade is a one-branch tree, of which every node is a classifier designed to reject a large portion of the Non-Face sub-windows and pass all of the Face ones. Consequently, most Non-Face sub-windows are rejected quickly before reaching the final node, resulting in very fast face detection performance.

The node classifiers are constructed using an algorithm similar to AdaBoost [3]. The algorithm combines an ensemble of weak classifiers to produce a final boosted classifier with very high accuracy. Much of the recent work using the cascade paradigm has tended to focus on improving the underlying boosting algorithm, such as AsymBoost [11, 14], FloatBoost [7], GentleBoost [8], and RealBoost [6, 16].

The weak classifiers are typically trained from a discrete set of Haar features, which are rectangular Haar-like wavelets, as illustrated in figure 2, at different positions, widths, and heights. Each feature is associated with a feature classifier, which classifies a sub-window by first integrating the sub-window with the feature's wavelet (using integral images), and then thresholding the value with a properly chosen threshold. The weak classifier is simply selected as the feature classifier with the smallest error.

One of the greatest obstacles to a wider use of cascades is that they take a long time to train, even if all parameters

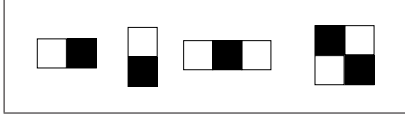


Figure 2. Four types of Haar features used in [13]

are properly chosen a priori. It took weeks of computing to produce the final cascade in [13], using multiple machines. The bottleneck is at training a weak classifier, of which the training time is $O(NT \log N)$, where N is the number of examples used and T is the size of the feature set. Training a single weak classifier has often run in minutes. In practice, one has to run many trials and choose the best configuration, resulting in even longer training time.

Since reducing N weakens the generalization, efforts have been made to reduce T by filtering the feature set and selecting only those *discriminant* for the current weak classifier, *e.g.* ranking using mutual-information [5] and Forward Feature Selection (FFS) [18]. However, ranking [5] could end up selecting only one single feature, which might *not* be the best feature, and the speed of FFS [18] is partly achieved by ignoring the weight distribution of boosting. Theoretically, FFS does not have the same hypothesis space as AdaBoost, so the features selected by FFS are not compatible for boosting.

Wu *et al.* [17] proposed to decrease the training time by using *caching*. They exploited that in different rounds of the AdaBoost algorithm, the weight distribution changes but the feature values of the training set remain constant. If one sorts all the N feature values of each feature and store all the sorting orders in a pre-processing stage, one can later compute the best threshold for each feature in time $O(N)$, regardless of the weights. Using this strategy, it is possible to reduce the training time from $O(NT \log N)$ to $O(NT)$, with a pre-calculation of time $O(NT \log N)$. The reported empirical result was rather impressive, increasing the training speed at the factor of around 15 to 60. However, this technique requires a huge amount of memory to store the sorting orders: $O(NT)$. Suppose $N = 10,000$ and $T = 40,000$ and two bytes are used to store an index, the memory required is about 800MB. Nevertheless, to our best knowledge, this method is currently the fastest implementation of training a face detector using boosting and Haar features, and is the source of comparison for our method.

In this paper, we present a method to train weak classifiers with relatively faster speed. Our approach takes a different point of view from traditional methods. Noticing that both N and T are dominant, but also important factors in constructing the weak classifiers, rather than trying to reduce either N or T , we break up the NT factor incurred in the time complexity of training a weak classifier. In our approach, the time complexity to train a weak classifier is

approximately linear to either N or T , whichever is more dominant, *but not both*. By doing so, we can train weak classifiers with more examples – giving better generalization, as well as more features – giving a wider hypothesis set for the weak classifiers.

While we respect that feature filtering methods speed up the training process, we argue that even though the Haar feature set is over-complete, it is still a small discrete subset in the feature space. Only four types of features were actually used in [13]. By exploring other types one could potentially increase the detector’s performance [6, 8]. However, increasing the number of types directly increases the size of the feature set dramatically. This in turn makes the training time longer, which is by far one of the main reasons that stops many methods from exploring other feature types.

In our approach, we used 19 Haar feature types (figure 3) on a 24×24 image sub-window, generating in total 295,920 features, six times the number of features used by Viola and Jones in [15]. Yet, it took 6 seconds to train all feature classifiers and select the best one on a conventional Intel Pentium IV PC.

The key idea to our training approach is, we treat image sub-windows as high dimensional random vectors, and keep all the necessary statistics of the data prior to training a weak classifier. We rely on classification methods which only use statistics to construct the feature classifiers, and on boosting to accelerate the performance. After estimating the statistics, we can train all T feature classifiers in time *independent* of N , which allows us to train and select much more features with little immersion in the training time. Besides, the total memory storage for all the statistics is also small, depending only on T and d .

The remaining parts of the paper are organized as follows. The framework of this method is described in section 2. In section 3, we discuss about the decisions made in the method, and use empirical results to justify them. Comparisons to other methods are presented in section 4. Conclusions are presented in section 5.

2. Training a weak classifier in face detection using boosting and Haar features

2.1. Training a node classifier

For clarity, we briefly describe how boosting is used to train a node classifier in the cascade. For further details, we refer the readers to [13, 14]. Suppose the training set has N examples, $\{(\mathbf{I}^{(n)}, c^{(n)})\}$ of the joint random vector (\mathbf{I}, c) , where \mathbf{I} is image sub-window random matrix of size \sqrt{d} -by- \sqrt{d} , of which the elements $\mathbf{I}_{y,x}$ (pixel at row y , column x) are random variables, and $c^{(n)}$ is its desired class. Let $c \in \{-1, 1\}$ with the notion that $c = 1$ being the Face class and $c = -1$ being the Non-face class.

We denote by $\vec{\mathbf{A}}$ the vectorized form of an arbitrary ma-

trix \mathbf{A} by aligning all the elements of \mathbf{A} in a column with a predefined order. Let $\mathbf{x} = \vec{\mathbf{I}}$. Obviously, $\mathbf{x} \in \mathbb{R}^d$ and can be considered as a d -dimensional random vector with an unknown distribution.

The basic idea of AdaBoost is to train an ensemble of M weak classifiers $f_m(\mathbf{x})$ in the form:

$$F_M(\mathbf{x}) = \sum_{m=1}^M a_m f_m(\mathbf{x}), \quad (1)$$

where a_m are voting coefficients. This is done greedily by training $(a_m, f_m(\mathbf{x}))$ for m from 1 to M . However, at the m -th stage, the training examples are weighted differently using a weighting function $w_m(F_{m-1}(\mathbf{x}), c)$, based on the performance of the previous stage. The idea is to increase the weights of those wrongly classified examples and decrease the weights of those correctly classified examples, so that subsequent weak classifiers focus more on the wrongly classified (harder) examples. Different weighting functions lead to different optimizing criteria (e.g. see [11, 14]), which affect the final false acceptance rate (FAR) and false rejection rate (FRR) of the ensemble. By controlling M and $w_m(\cdot)$, one can train the ensemble to have a very low FRR and a moderate FAR, suitable to reject a large portion of negative examples.

The choice of M , the choice of $w_m(\cdot)$, and how they affect the final FAR and FRR are not addressed in this paper. We only assume that at the m -th stage, the weighting function $w_m(\cdot)$ is known.

2.2. Linear relationship between integral image and feature values

Denote by $\mathbf{H}^{(t)}$ the matrix representing the t -th Haar-like wavelet w.r.t. the coordinate system of the probed sub-window. $\mathbf{H}^{(t)}$ is a matrix of size \sqrt{d} -by- \sqrt{d} , of which the elements' values are in a small discrete set. Also, denote by $v^{(t)}$ the random variable representing the feature value observed from integrating the t -th Haar feature $\mathbf{H}^{(t)}$ with a sub-window \mathbf{I} . Let $\mathbf{h}^{(t)} = \vec{\mathbf{H}}^{(t)}$, we get:

$$v^{(t)} = \mathbf{H}^{(t)} \circ \mathbf{I} = \sum_{x=1}^d \sum_{y=1}^d \mathbf{H}_{y,x}^{(t)} \mathbf{I}_{y,x} = \mathbf{h}^{(t)\top} \mathbf{x}, \quad (2)$$

which means $v^{(t)}$ can be considered as the result of *linearly projecting* the random vector \mathbf{x} down to \mathbb{R} using $\mathbf{h}^{(t)}$ as the direction of projection (along with a scaling factor).

Now consider the integral image \mathbf{J} [13] of sub-window \mathbf{I} , defined by:

$$\mathbf{J}_{y,x} = \sum_{x'=1}^x \sum_{y'=1}^y \mathbf{I}_{y',x'}. \quad (3)$$

Algorithm 1 Fast training a weak classifier

Require: $(\mathbf{y}^{(n)}, c^{(n)}, w_m^{(n)})$ for $n = 1..N$. Here, $\mathbf{y}^{(n)}$ denotes the integral image of the n -th input example, $c^{(n)}$ denotes its class, and $w^{(n)}$ denotes its current weight.

- 1: **for** each class c **do**
 - 2: Compute $\hat{z}_c = \sum_{n:c^{(n)}=c} w_m^{(n)}$.
 - 3: Compute $\hat{\mathbf{m}}_c = \hat{z}_c^{-1} \sum_{n:c^{(n)}=c} w_m^{(n)} \mathbf{y}^{(n)}$.
 - 4: Compute $\hat{\Sigma}_c = \left(\hat{z}_c^{-1} \sum_{n:c^{(n)}=c} w_m^{(n)} \mathbf{y}^{(n)} \mathbf{y}^{(n)\top} \right) - \hat{\mathbf{m}}_c \hat{\mathbf{m}}_c^\top$, see sections 2.3 and 3.2.
 - 5: **end for**
 - 6: **for** each feature t **do**
 - 7: **for** each class c **do**
 - 8: Compute $\hat{\mu}_c^{(t)} = \hat{\mathbf{m}}_c^\top \mathbf{g}^{(t)}$.
 - 9: Compute $\hat{\sigma}_c^{2(t)} = \mathbf{g}^{(t)\top} \hat{\Sigma}_c \mathbf{g}^{(t)}$, see section 2.2.
 - 10: **end for**
 - 11: Use $\{(\hat{z}_c, \hat{\mu}_c^{(t)}, \hat{\sigma}_c^{2(t)}) | c \in \{-1, +1\}\}$ to train feature classifier $h_t(v^{(t)})$, see section 2.3.
 - 12: **end for**
 - 13: Select the feature classifier $h_t(v^{(t)})$ with the smallest classification error, denote its index as t^* .
 - 14: Return $f_m(\mathbf{x}) = h_{t^*}(\mathbf{g}^{(t^*)\top} \mathbf{B} \mathbf{x})$ as the weak classifier.
-

Let $\mathbf{y} = \vec{\mathbf{J}}$. Equation (3) implies that both \mathbf{y} and \mathbf{x} are linearly related by: $\mathbf{y} = \mathbf{B} \mathbf{x}$, where \mathbf{B} is a fixed and non-singular linear transformation matrix expressing the cumulative summary. A feature value $v^{(t)}$ is also linearly related to the vectorized integral image \mathbf{y} :

$$v^{(t)} = \mathbf{h}^{(t)\top} \mathbf{x} = \mathbf{h}^{(t)\top} (\mathbf{B}^{-1} \mathbf{y}) = \mathbf{g}^{(t)\top} \mathbf{y}, \quad (4)$$

where $\mathbf{g}^{(t)} = \mathbf{B}^{-1\top} \mathbf{h}^{(t)}$ is a fixed d -dimensional vector. As pointed out in [13], the nice aspect of combining Haar features with integral images is that $\mathbf{g}^{(t)}$ are *sparse* vectors with very few non-zero elements, typically less than 10. As a result, computing a feature value can be done extremely fast, with a few hundred CPU clocks.

2.3. Training the feature classifiers using statistics

By exploring the linear relationship above further, the statistics of $v^{(t)}$ can be computed efficiently from the statistics of \mathbf{y} . Suppose $\mu^{(t)}$ and $\sigma^{2(t)}$ are the mean and the variance of $v^{(t)}$. Denote by $\langle \cdot \rangle$ the expectation of a random variable or a random vector. We have:

$$\mu^{(t)} = \langle v^{(t)} \rangle = \langle \mathbf{y}^\top \mathbf{g}^{(t)} \rangle = \mathbf{m}_y^\top \mathbf{g}^{(t)}, \quad (5)$$

where $\mathbf{m}_y = \langle \mathbf{y} \rangle$ is the mean vector of \mathbf{y} , and:

$$\begin{aligned} \sigma^{2(t)} &= \langle v^{(t)2} \rangle - \langle v^{(t)} \rangle^2 \\ &= \langle \mathbf{g}^{(t)\top} \mathbf{y} \mathbf{y}^\top \mathbf{g}^{(t)} \rangle - \mathbf{g}^{(t)\top} \mathbf{m}_y \mathbf{m}_y^\top \mathbf{g}^{(t)} \\ &= \mathbf{g}^{(t)\top} (\langle \mathbf{y} \mathbf{y}^\top \rangle - \langle \mathbf{y} \rangle \langle \mathbf{y} \rangle^\top) \mathbf{g}^{(t)} \\ &= \mathbf{g}^{(t)\top} \Sigma_y \mathbf{g}^{(t)}, \end{aligned} \quad (6)$$

where $\Sigma_{\mathbf{y}} = \langle \mathbf{y}\mathbf{y}^T \rangle - \langle \mathbf{y} \rangle \langle \mathbf{y} \rangle^T$ is the covariance matrix of \mathbf{y} . Since $\mathbf{g}^{(t)}$ is a *sparse* vector, if we know both the mean vector and the covariance matrix of \mathbf{y} , computing $\mu^{(t)}$ and $\sigma^{2(t)}$ can be done extremely fast as well.

An important observation is, while $\Sigma_{\mathbf{y}}$ itself may not be accurately estimated due to large dimensionality, the projected statistics $\mu^{(t)}$ and $\sigma^{2(t)}$ are much more accurately estimated as scalars. Additionally, higher order statistics may be obtained, but these are not explored in this paper.

We decided to train feature classifiers using only statistics of feature values and class. Several reasons were considered for this decision. On the one hand, the performance of the feature classifier in terms of accuracy is reduced, which in turns affects the weak classifier. However, when the problem becomes harder, as in the subsequent layers of the cascade, most weak classifiers tend to perform poorly, with errors typically close to 40-50 percent [15]. At those layers, the performance of an ensemble relies less on the performance of individual weak classifiers, but rather on the boosting scheme of the boosting algorithm used. Experiments in section 3.3 justify our argument.

On the other hand, if the mean vector and the covariance matrix of the current stage's random vector are known, all the statistics needed (up to the second order) to train a classifier can be computed extremely fast, and *independent* of the sample size N . Besides, estimating the mean vector and the covariance matrix only needs to be done once per weak classifier, and then substantially used for all T features, saving a large amount of computation.

We investigate this idea using a simple type of feature classifier widely known in the literature: simple thresholding with class-conditional Gaussian distribution assumptions (e.g. see chapter 2 in [2]). Here, the feature classifier $h_t(v^{(t)})$ assumes $v_m^{(t)}|c \sim \mathcal{N}(\mu_{m,c}^{(t)}, \sigma_{m,c}^{2(t)})$ for $c \in \{-1, +1\}$, where $v_m^{(t)}$ is the random variable representing the observed feature value when the image sub-window is weighted by $w_m(\cdot)$. To train the classifier, one finds a threshold $\theta_t \in \mathbb{R}$ and a parity $p_t \in \{-1, +1\}$ that minimizes the error rate $\epsilon^{(t)}(\theta_t, p_t)$ given by:

$$\epsilon^{(t)}(\theta_t, p_t) = P(v^{(t)} > \theta_t \wedge c = -p_t) + P(v^{(t)} < \theta_t \wedge c = p_t). \quad (7)$$

The minimization problem is non-convex, but has a closed form solution, as described in [2], eliminating the need to linear search for the optimal threshold. Once the optimal values (θ_t^*, p_t^*) have been obtained, the feature value is classified by:

$$\hat{c} = h_t(v^{(t)}) = p_t^* \text{sign}(v^{(t)} > \theta_t^*). \quad (8)$$

The feature classifier with the smallest error rate is chosen as the weak classifier.

We only need to estimate the statistics up to the second order to compute the probabilities in (7). Denote by \mathbf{x}_m the

weighted version of \mathbf{x} using $w_m(\cdot)$, and by $\mathbf{y}_m = \mathbf{B}\mathbf{x}_m$ its corresponding integral image random vector, an analysis similar to equations from (5) to (6) shows that:

$$\mu_{m,c}^{(t)} = \mathbf{m}_{\mathbf{y}_m|c}^T \mathbf{g}^{(t)} \quad (9)$$

$$\sigma_{m,c}^{2(t)} = \mathbf{g}^{(t)T} \Sigma_{\mathbf{y}_m|c} \mathbf{g}^{(t)}, \quad (10)$$

where $\mathbf{m}_{\mathbf{y}_m|c}$ and $\Sigma_{\mathbf{y}_m|c}$ are the mean vector and the covariance matrix of the $\mathbf{y}_m|c$ class-conditional random vector. Therefore, if $\mathbf{m}_{\mathbf{y}_m|c}$ and $\Sigma_{\mathbf{y}_m|c}$ are known, all the $\mu_{m,c}^{(t)}$ and $\sigma_{m,c}^{2(t)}$ can be computed very fast as well.

The class-conditional mean vectors $\mathbf{m}_{\mathbf{y}_m|c}$ and covariance matrices $\Sigma_{\mathbf{y}_m|c}$ can be *estimated directly* from the weighted training set, by computing the weighted class-conditional sample mean vectors and covariance matrices:

$$\hat{\mathbf{m}}_{\mathbf{y}_m|c} = z_{\mathbf{y}_m|c}^{-1} \sum_{n:c^{(n)}=c} w_m^{(n)} \mathbf{y}^{(n)} \quad (11)$$

$$\hat{\Sigma}_{\mathbf{y}_m|c} = (z_{\mathbf{y}_m|c}^{-1} \sum_{n:c^{(n)}=c} w_m^{(n)} \mathbf{y}^{(n)} \mathbf{y}^{(n)T}) - \hat{\mathbf{m}}_{\mathbf{y}_m|c} \hat{\mathbf{m}}_{\mathbf{y}_m|c}^T, \quad (12)$$

where $z_{\mathbf{y}_m|c} = \sum_{n:c^{(n)}=c} w_m^{(n)}$ is the total weight of class c , and $w_m^{(n)}$ is the weight applied to the n -th training example.

This method's pseudo-code is illustrated in algorithm 1.

2.4. Time complexity

This method can be broken into two components. The first component (from line 1 to line 5) estimates the global class-conditional statistics, which is dominated by estimating the covariance matrices. Given N examples of d dimensions, the time complexity of this component is $O(Nd^2)$.

The other component (from lines 6 to 12) computes the local class-conditional means and variances by projecting the original class-conditional mean vectors and covariance matrices in different directions. Because $\mathbf{g}^{(t)}$ are *sparse*, the projections can be computed extremely fast by taking into account only a few non-zero elements per projection. All other elements are untouched. Thus, the time to compute the projected statistics can be considered $O(1)$. Each feature classifier is trained using only six real numbers. Therefore, its time complexity is $O(1)$ as well. In total, the time complexity of the second component is $O(T)$, and the total time to train a weak classifier is $O(Nd^2 + T)$.

3. Implementation and Experimental Results

3.1. Experiment Setup

To justify the arguments we made in the previous section, we ran a few experiments. We collected 1521 face images from the BioID Face Database ¹, 508 face images from the

¹<http://www.bioid.com/downloads/facedb/index.php>

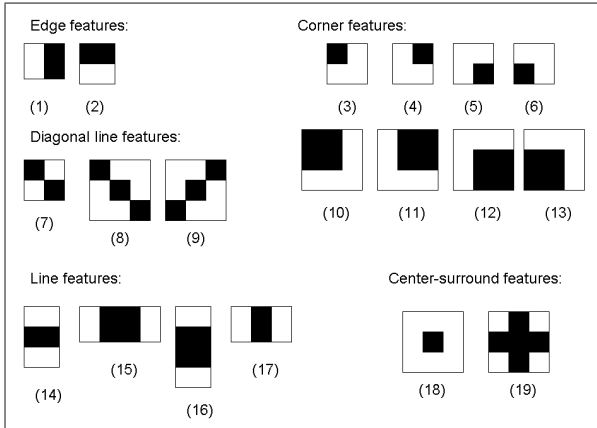


Figure 3. Nineteen types of features used in our experiment

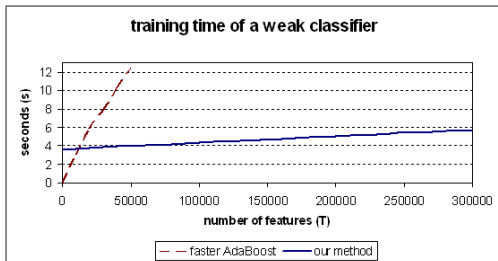


Figure 4. Comparison on the time to train a weak classifier

AR Face Database [9], and a few hundred more face images with known (manually labeled) face locations from the web. Altogether we obtained about 2500 face images. The faces are scaled and aligned to a base resolution of 24×24 pixels. They are further mirrored to form a training set of around 5000 faces. Note that our face training set contains mostly adult faces. Similar to traditional methods, we generated sufficient non-face sub-windows from a collection of a few thousand large images containing no faces.

The cascades were trained using the bootstrapping strategy used in [13, 14, 15]. We implemented two types of cascade: one using the faster AdaBoost implementation proposed by Wu *et al.* [17], the other using our proposed training method. Each node classifier was trained using AsymBoost [14] with a training set of 5000 positive sub-windows and 5000 negative sub-windows which are false positives of the previous node. Nineteen different feature types (figure 3) were used to generate a total of 295,920 Haar features.

For testing, we used the MIT+CMU test set which consists of 130 grayscale images with 507 frontal faces [12]. Post-processing was the same as in [15]. The experiments were done on a 2.8GHz Intel Pentium IV PC with 1.5 GB memory.

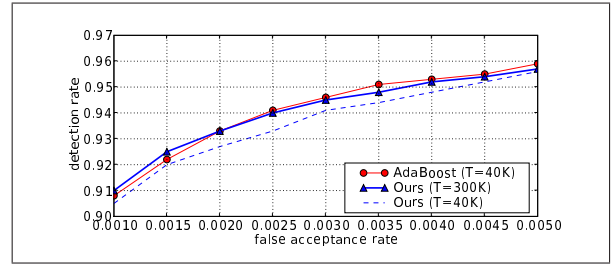


Figure 5. ROC curves of three node classifiers with $M = 15$

3.2. Training time of a weak classifier

Figure 4 shows the time to train a weak classifier on both methods. To be as fair as possible, we optimized the critical code sections in both implementations. In the faster AdaBoost implementation, the section of computing the T thresholds was written in highly optimized C code. We were able to obtain the training time of 12.4 seconds when $T = 40,000$.

The bottleneck of our method is at computing the weighted class-conditional sample covariance matrices, which is expected at time $O(Nd^2)$. However, by reordering the terms carefully, a covariance matrix can be computed by multiplying a $d \times N_c$ matrix with its transpose, where N_c is the number of examples of class c . By further relying on a highly optimized linear algebra package, GotoBLAS [4], it took 1.82 seconds to compute a matrix-matrix multiplication with $N_c = 5000$ and $d = 24^2$. The second component of this method ran in only 2.1 seconds to compute all the statistics required by $T = 295,920$ features.

3.3. Performance of the boosted classifiers

We constructed three boosted classifiers of 15 weak classifiers: faster AdaBoost with $T = 40,000$ features, our method using the same feature set with faster AdaBoost, and our method with $T = 295,920$ features. By changing the controlling parameters, we obtained the ROC curves, illustrated in figure 5. The total time to train the boosted classifiers are: 180 seconds for faster AdaBoost, 75 seconds for our method with $T = 40,000$, and 90 seconds for our method with $T = 295,920$.

An analysis of the ROC curves shows that the false acceptance rate of our method with $T = 40,000$ is about 10% higher than the other two. The results suggest that for small M - the number of weak classifiers in the ensemble, when using the same feature set, this method did not perform as well as AdaBoost. However, with more feature types the discriminant power of this method increases with minimal increase in training time.

We constructed another three boosted classifiers but this time with 200 weak classifiers. The total time to train the boosted classifiers are: 2400 seconds for faster AdaBoost,

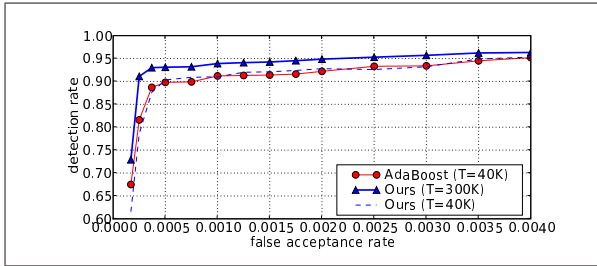


Figure 6. ROC curves of three node classifiers with $M = 200$



Figure 7. Some results on the MIT+CMU test set

1000 seconds for our method with $T = 40,000$, and 1200 seconds for our method with $T = 295,920$. Similarly, the ROC curves are illustrated in figure 6.

From the ROC curves, the performance of faster AdaBoost and our method with $T = 40,000$ are relatively similar. However, our method with $T = 295,920$ performed with about 5% reduction in false acceptance rate with minimal immersion in training time. The results indicate that it is beneficial to add more feature types, especially even when the problem becomes harder. The same observation was drawn in [8].

3.4. Final performance

We constructed three cascades: the first cascade used the faster AdaBoost implementation with $T = 40,000$ features, the second cascade used our method utilizing the same feature set of the first one, the final cascade used our method with $T = 295,920$. The three cascades had 30, 30, and 25

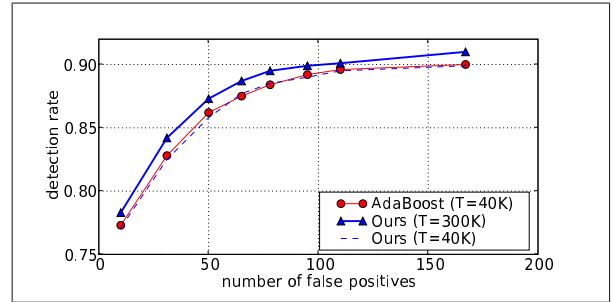


Figure 8. ROC curves comparing cascades using the faster AdaBoost implementation and this method on the MIT+CMU test set

layers respectively. We stopped training the cascades when observing that the next ensemble (of usually 200 weak classifiers) could not reduce the total false acceptance rate by less than 5% while maintaining the detection rate at above 99.9%. The ROC curves are illustrated in figure 8.

The performances of the first and the second cascade in the ROC curves show that both cascades performed relatively similar. However, the total training time of this method was significantly reduced. In our experiment, we saw a gaining factor of 2.5 in training speed.

Moreover, when training the third cascade, we saw a reduction of layers due to better performance in every node. The ROC curves show that the third cascade out-performed the faster AdaBoost implementation in accuracy, and was trained at 2.4 times faster due to fewer layers but slightly longer training time per weak classifier.

In addition, the faster AdaBoost implementation required almost 1GB of memory to store all the intermediate values, while our method only needed less than 30MB of memory for the same purpose.

4. Relation to other methods

Recently, there have been proposals to replace Discrete AdaBoost by Real AdaBoost, and use real-valued weak classifiers instead of thresholding [6, 16]. Their approach differs to ours in approximating the class-conditional distributions: they use histograms while we use Gaussian assumptions. Although we experimented with discrete-valued weak classifiers, it is possible to produce real-valued weak classifiers compatible to their approach by returning $h_t(v^{(t)}) = 0.5 \log \frac{P(v^{(t)}|c=1)}{P(v^{(t)}|c=-1)}$ instead of ± 1 . However, while Discrete AdaBoost is strongly resistant to overfitting [1], it is not known if the same argument holds for Real AdaBoost. The choice of the number of bins B raises another concern. The larger B is, the less accurate the per-bin values are estimated, and the more overfitted the weak classifier becomes.

Our method shares some analogy with Online Boosting [10, 11]: the proposed weak classifier can be used as an on-

Method	No of face examples	No of features	Time to train a weak classifier	No of weak classifiers	No of nodes	Total training time	CPU speed
Viola-Jones [15]	9,500	40,000	> 10m	4,297	32	weeks	400MHz
FloatBoost [7]	6,000	-	-	2,546	-	weeks	700MHz
Nested Cascade + LUT [16]	20,000	-	-	756	16	weeks	1.4GHz
Sparse Granular Features [6]	30,000	-	< 1m	-	-	2 days	3.0GHz
Faster AdaBoost [17]	5,000	40,000	12.4s	3,870	30	13h20m	2.8GHz
Our method	5,000	295,920	5.74s	3,502	25	5h30m	2.8GHz

Table 1. The reported training time of recent methods.

line learner. Unfortunately, using this method along with Online Boosting gains little benefit. When updating a weak classifier with one new example, one has to update all T Haar features. The time complexity of Online Boosting with or without this method is still the same: $O(T)$.

In section 2.2, we proved that Haar features are simply linear projections of the probe image sub-window. Though this paper addresses Haar features, this method is clearly applicable to any local features that are linear projections of a random vector, like the ones used in [6, 7, 8].

We summarize the reported training time of recent methods and ours in table 1.

5. Conclusions

In this paper, we present a fast method to train and select Haar features for the training of a weak classifier in face detection cascades using boosting. It trains a weak classifier in time $O(Nd^2 + T)$ (in seconds), rather than $O(NT \log N)$ (in minutes) in traditional methods. In compared with the currently fastest implementation of boosting [17], taking about 12 seconds per weak classifier using 40,000 Haar features but requiring almost 1GB of memory, this method trained a weak classifier using nearly 300,000 Haar features in 6 seconds, while requiring less than 30MB of memory and maintaining a compelling performance. Besides, it suffers very minimal immersion in training time with very large increases in members of Haar features, enjoying a significant gain in accuracy. By substantially reducing the training time, this method empowers researchers to much more quickly experiment and explore solutions to other important research issues in this area.

Acknowledgments

This work was done at Centre for Multimedia and Network Technology (CeMNet), School of Computer Engineering, Nanyang Technological University, Singapore. We would like to thank Bo Wu, Jianxin Wu, Chang Huang, Minhtuan Pham, ThanhGiang T. Vo, ThanhVu H. Nguyen, Duc-Minh Pham, Trung-Tuan Luong, M. N. Nguyen, and Xuan-Hong Dang for their support on this work.

References

- [1] P. L. Bartlett and M. Traskin. Adaboost is consistent. In *NIPS19*. MIT Press, 2006.
- [2] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.
- [3] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.
- [4] K. Goto and R. van de Geijn. High-performance implementation of the level-3 blas. Technical report, The University of Texas at Austin, Department of Computer Sciences, 2006.
- [5] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.
- [6] C. Huang, H. Ai, Y. Li, and S. Lao. High-performance rotation invariant multiview face detection. *TPAMI*, 29:671–686, 2007.
- [7] S. Z. Li, L. Zhu, Z. Zhang, A. Blake, H. Zhang, and H. Shum. Statistical learning of multi-view face detection. In *ECCV*, pages 67–81, 2002.
- [8] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM-Symposium*, pages 297–304, 2003.
- [9] A. Martinez and R. Benavente. The ar face database. Technical Report 24, CVC, U.A.B., June 1998.
- [10] N. Oza. Online bagging and boosting. In *ICSMC*, volume 3, pages 2340–2345 Vol.3, 10-12 Oct. 2005.
- [11] M.-T. Pham and T.-J. Cham. Online learning asymmetric boosted classifiers for object detection. In *CVPR*, 2007.
- [12] H. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *Proc. CVPR*, June 1998.
- [13] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. CVPR*, 2001.
- [14] P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *NIPS 14*. MIT Press, Cambridge, MA, 2002.
- [15] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004.
- [16] B. Wu, H. Ai, C. Huang, and S. Lao. Fast rotation invariant multi-view face detection based on real adaboost. In *FGR*, pages 79–84, 17-19 May 2004.
- [17] J. Wu, S. C. Brubaker, M. D. Mullin, and J. M. Rehg. Fast asymmetric learning for cascade face detection. *To Appear in TPAMI*, 2007.
- [18] J. Wu, J. M. Rehg, and M. D. Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS 16*. MIT Press, 2004.