International Journal of Computer Vision 57(2), 137–154, 2004

Robust Real-Time Face Detection

PAUL VIOLA
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
viola@microsoft.com

MICHAEL J. JONES
Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, USA
mjones@merl.com

Received September 10, 2001; Revised July 10, 2003; Accepted July 11, 2003

Abstract. This paper describes a face detection framework that is capable of processing images extremely rapidly while achieving high detection rates. There are three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly. The second is a simple and efficient classifier which is built using the AdaBoost learning algorithm (Freund and Schapire, 1995) to select a small number of critical visual features from a very large set of potential features. The third contribution is a method for combining classifiers in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising face-like regions. A set of experiments in the domain of face detection is presented. The system yields face detection performance comparable to the best previous systems (Sung and Poggio, 1998; Rowley et al., 1998; Schneiderman and Kanade, 2000; Roth et al., 2000). Implemented on a conventional desktop, face detection proceeds at 15 frames per second.

Keywords: face detection, boosting, human sensing

1. Introduction

This paper brings together new algorithms and insights to construct a framework for robust and extremely rapid visual detection. Toward this end we have constructed a frontal face detection system which achieves detection and false positive rates which are equivalent to the best published results (Sung and Poggio, 1998; Rowley et al., 1998; Osuna et al., 1997a; Schneiderman and Kanade, 2000; Roth et al., 2000). This face detection system is most clearly distinguished from previous approaches in its ability to detect faces extremely rapidly. Operating on 384 by 288 pixel images, faces are detected at 15 frames per second on a conventional 700 MHz Intel Pentium III. In other face detection systems, auxiliary information, such as image differences in video sequences, or pixel color in color images, have been used to achieve high frame rates. Our system achieves high frame rates working only with the information present in a single grey scale image. These alternative sources of information can also be integrated with our system to achieve even higher frame rates.

There are three main contributions of our face detection framework. We will introduce each of these ideas briefly below and then describe them in detail in subsequent sections.

The first contribution of this paper is a new image representation called an *integral image* that allows for very fast feature evaluation. Motivated in part by the work of Papageorgiou et al. (1998) our detection system does not work directly with image intensities. Like

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

these authors we use a set of features which are reminiscent of Haar Basis functions (though we will also use related filters which are more complex than Haar filters). In order to compute these features very rapidly at many scales we introduce the integral image representation for images (the integral image is very similar to the summed area table used in computer graphics (Crow, 1984) for texture mapping). The integral image can be computed from an image using a few operations per pixel. Once computed, any one of these Haar-like features can be computed at any scale or location in *constant* time.

The second contribution of this paper is a simple and efficient classifier that is built by selecting a small number of important features from a huge library of potential features using AdaBoost (Freund and Schapire, 1995). Within any image sub-window the total number of Haar-like features is very large, far larger than the number of pixels. In order to ensure fast classification, the learning process must exclude a large majority of the available features, and focus on a small set of critical features. Motivated by the work of Tieu and Viola (2000) feature selection is achieved using the AdaBoost learning algorithm by constraining each weak classifier to depend on only a single feature. As a result each stage of the boosting process, which selects a new weak classifier, can be viewed as a feature selection process. AdaBoost provides an effective learning algorithm and strong bounds on generalization performance (Schapire et al., 1998).

The third major contribution of this paper is a method for combining successively more complex classifiers in a cascade structure which dramatically increases the speed of the detector by focusing attention on promising regions of the image. The notion behind focus of attention approaches is that it is often possible to rapidly determine where in an image a face might occur (Tsotsos et al., 1995; Itti et al., 1998; Amit and Geman, 1999; Fleuret and Geman, 2001). More complex processing is reserved only for these promising regions. The key measure of such an approach is the "false negative" rate of the attentional process. It must be the case that all, or almost all, face instances are selected by the attentional filter.

We will describe a process for training an extremely simple and efficient classifier which can be used as a "supervised" focus of attention operator.[1] A face detection attentional operator can be learned which will filter out over 50% of the image while preserving 99% of the faces (as evaluated over a large dataset). This filter is exceedingly efficient; it can be evaluated in 20 simple operations per location/scale (approximately 60 microprocessor instructions).

Those sub-windows which are not rejected by the initial classifier are processed by a sequence of classifiers, each slightly more complex than the last. If any classifier rejects the sub-window, no further processing is performed. The structure of the cascaded detection process is essentially that of a degenerate decision tree, and as such is related to the work of Fleuret and Geman (2001) and Amit and Geman (1999).

The complete face detection cascade has 38 classifiers, which total over 80,000 operations. Nevertheless the cascade structure results in extremely rapid average detection times. On a difficult dataset, containing 507 faces and 75 million sub-windows, faces are detected using an average of 270 microprocessor instructions per sub-window. In comparison, this system is about 15 times faster than an implementation of the detection system constructed by Rowley et al. (1998).[2]

An extremely fast face detector will have broad practical applications. These include user interfaces, image databases, and teleconferencing. This increase in speed will enable real-time face detection applications on systems where they were previously infeasible. In applications where rapid frame-rates are not necessary, our system will allow for significant additional post-processing and analysis. In addition our system can be implemented on a wide range of small low power devices, including hand-helds and embedded processors. In our lab we have implemented this face detector on a low power 200 mips *Strong Arm* processor which lacks floating point hardware and have achieved detection at two frames per second.

1.1. Overview

The remaining sections of the paper will discuss the implementation of the detector, related theory, and experiments. Section 2 will detail the form of the features as well as a new scheme for computing them rapidly. Section 3 will discuss the method in which these features are combined to form a classifier. The machine learning method used, a application of AdaBoost, also acts as a feature selection mechanism. While the classifiers that are constructed in this way have good computational and classification performance, they are far too slow for a real-time classifier. Section 4 will describe a method for constructing a cascade of classifiers which together yield an extremely reliable and efficient face detector. Section 5 will describe a number of experimental results, including a detailed description of our experimental methodology. Finally Section 6 contains a discussion of this system and its relationship to related systems.

2. Features

Our face detection procedure classifies images based on the value of simple features. There are many motivations for using features rather than the pixels directly. The most common reason is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data. For this system there is also a second critical motivation for features: the feature-based system operates much faster than a pixel-based system.

The simple features used are reminiscent of Haar basis functions which have been used by Papageorgiou et al. (1998). More specifically, we use three kinds of features. The value of a *two-rectangle feature* is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent (see Fig. 1). A *three-rectangle feature* computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally a *four-rectangle feature* computes the difference between diagonal pairs of rectangles.

Given that the base resolution of the detector is 24×24, the exhaustive set of rectangle features is quite large, 160,000. Note that unlike the Haar basis, the set of rectangle features is overcomplete.[3]

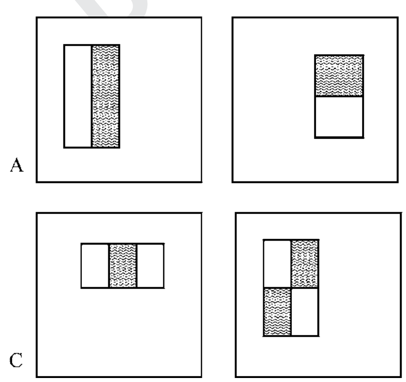

Figure 1. Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

2.1. Integral Image

Rectangle features can be computed very rapidly using an intermediate representation for the image which we call the integral image.[4] The integral image at location x, y contains the sum of the pixels above and to the left of x, y, inclusive:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image (see Fig. 2). Using the following pair of recurrences:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{1}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{2}$$

(where $s(x, y)$ is the cumulative row sum, $s(x, -1) = 0$, and $ii(-1, y) = 0$) the integral image can be computed in one pass over the original image.

Using the integral image any rectangular sum can be computed in four array references (see Fig. 3). Clearly the difference between two rectangular sums can be computed in eight references. Since the two-rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features.

One alternative motivation for the integral image comes from the "boxlets" work of Simard et al.

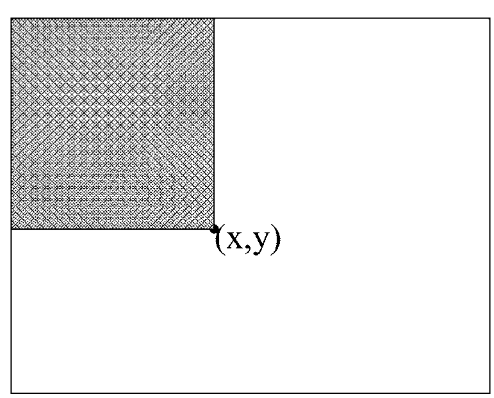

Figure 2. The value of the integral image at point (x, y) is the sum of all the pixels above and to the left.

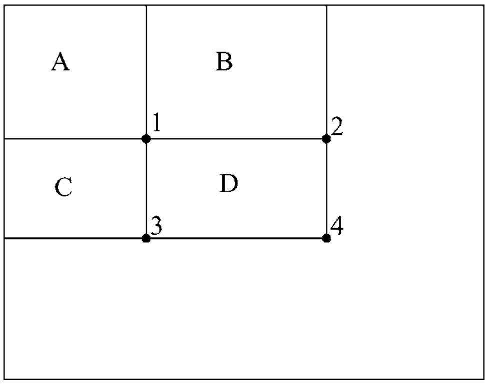

Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

(1999). The authors point out that in the case of linear operations (e.g. $f \cdot g$), any invertible linear operation can be applied to f or g if its inverse is applied to the result. For example in the case of convolution, if the derivative operator is applied both to the image and the kernel the result must then be double integrated:

$$f * g = \int\int (f' * g').$$

The authors go on to show that convolution can be significantly accelerated if the derivatives of f and g are sparse (or can be made so). A similar insight is that an invertible linear operation can be applied to f if its inverse is applied to g:

$$(f'') * \left(\int\int g\right) = f * g.$$

Viewed in this framework computation of the rectangle sum can be expressed as a dot product, $i \cdot r$, where i is the image and r is the box car image (with value 1 within the rectangle of interest and 0 outside). This operation can be rewritten

$$i \cdot r = \left(\int\int i\right) \cdot r''.$$

The integral image is in fact the double integral of the image (first along rows and then along columns). The second derivative of the rectangle (first in row and then in column) yields four delta functions at the corners of the rectangle. Evaluation of the second dot product is accomplished with four array accesses.

2.2. *Feature Discussion*

Rectangle features are somewhat primitive when compared with alternatives such as steerable filters (Freeman and Adelson, 1991; Greenspan et al., 1994). Steerable filters, and their relatives, are excellent for the detailed analysis of boundaries, image compression, and texture analysis. While rectangle features are also sensitive to the presence of edges, bars, and other simple image structure, they are quite coarse. Unlike steerable filters, the only orientations available are vertical, horizontal and diagonal. Since orthogonality is not central to this feature set, we choose to generate a very large and varied set of rectangle features. Typically the representation is about 400 times overcomplete. This overcomplete set provides features of arbitrary aspect ratio and of finely sampled location. Empirically it appears as though the set of rectangle features provide a rich image representation which supports effective learning. The extreme computational efficiency of rectangle features provides ample compensation for their limitations.

In order to appreciate the computational advantage of the integral image technique, consider a more conventional approach in which a pyramid of images is computed. Like most face detection systems, our detector scans the input at many scales; starting at the base scale in which faces are detected at a size of 24×24 pixels, a 384 by 288 pixel image is scanned at 12 scales each a factor of 1.25 larger than the last. The conventional approach is to compute a pyramid of 12 images, each 1.25 times smaller than the previous image. A fixed scale detector is then scanned across each of these images. Computation of the pyramid, while straightforward, requires significant time. Implemented efficiently on conventional hardware (using bilinear interpolation to scale each level of the pyramid) it takes around .05 seconds to compute a 12 level pyramid of this size (on an Intel PIII 700 MHz processor).[5]

In contrast we have defined a meaningful set of rectangle features, which have the property that a single feature can be evaluated at any scale and location in a few operations. We will show in Section 4 that effective face detectors can be constructed with as few as *two* rectangle features. Given the computational efficiency of these features, the face detection process can be completed for an entire image at every scale at 15 frames per second, about the same time required to evaluate the 12 level image pyramid alone. Any procedure which requires a pyramid of this type will necessarily run slower than our detector.

3. Learning Classification Functions

Given a feature set and a training set of positive and negative images, any number of machine learning approaches could be used to learn a classification function. Sung and Poggio use a mixture of Gaussian model (Sung and Poggio, 1998). Rowley et al. (1998) use a small set of simple image features and a neural network. Osuna et al. (1997b) used a support vector machine. More recently Roth et al. (2000) have proposed a new and unusual image representation and have used the Winnow learning procedure.

Recall that there are 160,000 rectangle features associated with each image sub-window, a number far larger than the number of pixels. Even though each feature can be computed very efficiently, computing the complete set is prohibitively expensive. Our hypothesis, which is borne out by experiment, is that a very small number of these features can be combined to form an effective classifier. The main challenge is to find these features.

In our system a variant of AdaBoost is used both to select the features and to train the classifier (Freund and Schapire, 1995). In its original form, the AdaBoost learning algorithm is used to boost the classification performance of a simple learning algorithm (e.g., it might be used to boost the performance of a simple perceptron). It does this by combining a collection of weak classification functions to form a stronger classifier. In the language of boosting the simple learning algorithm is called a weak learner. So, for example the perceptron learning algorithm searches over the set of possible perceptrons and returns the perceptron with the lowest classification error. The learner is called weak because we do not expect even the best classification function to classify the training data well (i.e. for a given problem the best perceptron may only classify the training data correctly 51% of the time). In order for the weak learner to be boosted, it is called upon to solve a sequence of learning problems. After the first round of learning, the examples are re-weighted in order to emphasize those which were incorrectly classified by the previous weak classifier. The final strong classifier takes the form of a perceptron, a weighted combination of weak classifiers followed by a threshold.[6]

The formal guarantees provided by the AdaBoost learning procedure are quite strong. Freund and Schapire proved that the training error of the strong classifier approaches zero exponentially in the number of rounds. More importantly a number of results were later proved about generalization performance (Schapire et al., 1997). The key insight is that generalization performance is related to the margin of the examples, and that AdaBoost achieves large margins rapidly.

The conventional AdaBoost procedure can be easily interpreted as a greedy feature selection process. Consider the general problem of boosting, in which a large set of classification functions are combined using a weighted majority vote. The challenge is to associate a large weight with each good classification function and a smaller weight with poor functions. AdaBoost is an aggressive mechanism for selecting a small set of good classification functions which nevertheless have significant variety. Drawing an analogy between weak classifiers and features, AdaBoost is an effective procedure for searching out a small number of good "features" which nevertheless have significant variety.

One practical method for completing this analogy is to restrict the weak learner to the set of classification functions each of which depend on a single feature. In support of this goal, the weak learning algorithm is designed to select the single rectangle feature which best separates the positive and negative examples (this is similar to the approach of Tieu and Viola (2000) in the domain of image database retrieval). For each feature, the weak learner determines the optimal threshold classification function, such that the minimum number of examples are misclassified. A weak classifier ($h(x, f, p, \theta)$) thus consists of a feature (f), a threshold (θ) and a polarity (p) indicating the direction of the inequality:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}$$

Here x is a 24×24 pixel sub-window of an image.

In practice no single feature can perform the classification task with low error. Features which are selected early in the process yield error rates between 0.1 and 0.3. Features selected in later rounds, as the task becomes more difficult, yield error rates between 0.4 and 0.5. Table 1 shows the learning algorithm.

The weak classifiers that we use (thresholded single features) can be viewed as single node decision trees.

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

Table 1. The boosting algorithm for learning a query online. T hypotheses are constructed each using a single feature. The final hypothesis is a weighted linear combination of the T hypotheses where the weights are inversely proportional to the training errors.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Select the best weak classifier with respect to the weighted error
     $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \,|\, h(x_i, f, p, \theta) - y_i \,| \,.$$
     See Section 3.1 for a discussion of an efficient implementation.
  3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.
  4. Update the weights:
     $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
  where $\alpha_t = \log \frac{1}{\beta_t}$

Such structures have been called decision stumps in the machine learning literature. The original work of Freund and Schapire (1995) also experimented with boosting decision stumps.

3.1. *Learning Discussion*

The algorithm described in Table 1 is used to select key weak classifiers from the set of possible weak classifiers. While the AdaBoost process is quite efficient, the set of weak classifier is extraodinarily large. Since there is one weak classifier for each distinct feature/threshold combination, there are effectively *KN* weak classifiers, where K is the number of features and N is the number of examples. In order to appreciate the dependency on N, suppose that the examples are sorted by a given feature value. With respect to the training process any two thresholds that lie between the same pair of sorted examples is equivalent. Therefore the total number of distinct thresholds is N. Given a task with $N = 20000$ and $K = 160000$ there are 3.2 billion distinct binary weak classifiers.

The wrapper method can also be used to learn a perceptron which utilizes M weak classifiers (John et al., 1994) The wrapper method also proceeds incrementally by adding one weak classifier to the perceptron in each round. The weak classifier added is the one which when added to the current set yields a perceptron with lowest error. Each round takes at least $O(NKN)$ (or 60 Trillion operations); the time to enumerate all binary features and evaluate each example using that feature. This neglects the time to learn the perceptron weights. Even so, the final work to learn a 200 feature classifier would be something like $O(MNKN)$ which is 10^{16} operations.

The key advantage of AdaBoost as a feature selection mechanism, over competitors such as the wrapper method, is the speed of learning. Using AdaBoost a 200 feature classifier can be learned in $O(MNK)$ or about 10^{11} operations. One key advantage is that in each round the entire dependence on previously selected features is efficiently and compactly encoded using the example weights. These weights can then be used to evaluate a given weak classifier in constant time.

The weak classifier selection algorithm proceeds as follows. For each feature, the examples are sorted based on feature value. The AdaBoost optimal threshold for that feature can then be computed in a single pass over this sorted list. For each element in the sorted list, four sums are maintained and evaluated: the total sum of positive example weights T^+, the total sum of negative example weights T^-, the sum of positive weights below the current example S^+ and the sum of negative weights below the current example S^-. The error for a threshold which splits the range between the current and previous example in the sorted list is:

$$e = \min\left(S^+ + (T^- - S^-), S^- + (T^+ - S^+)\right),$$

or the minimum of the error of labeling all examples below the current example negative and labeling the examples above positive versus the error of the converse. These sums are easily updated as the search proceeds.

Many general feature selection procedures have been proposed (see chapter 8 of Webb (1999) for a review). Our final application demanded a very aggressive process which would discard the vast majority of features. For a similar recognition problem Papageorgiou et al. (1998) proposed a scheme for feature selection based on feature variance. They demonstrated good results selecting 37 features out of a total 1734 features. While this is a significant reduction, the number of features evaluated for every image sub-window is still reasonably large.

Roth et al. (2000) propose a feature selection process based on the Winnow exponential perceptron learning rule. These authors use a very large and unusual feature set, where *each pixel* is mapped into a binary vector of d dimensions (when a particular pixel takes on the value x, in the range $[0, d-1]$, the x-th dimension is set to 1 and the other dimensions to 0). The binary vectors for each pixel are concatenated to form a single binary vector with nd dimensions (n is the number of pixels). The classification rule is a perceptron, which assigns one weight to each dimension of the input vector. The Winnow learning process converges to a solution where many of these weights are zero. Nevertheless a very large number of features are retained (perhaps a few hundred or thousand).

3.2. *Learning Results*

While details on the training and performance of the final system are presented in Section 5, several simple results merit discussion. Initial experiments demonstrated that a classifier constructed from 200 features would yield reasonable results (see Fig. 4). Given a detection rate of 95% the classifier yielded a false positive rate of 1 in 14084 on a testing dataset. This is promising, but for a face detector to be practical for real applications, the false positive rate must be closer to 1 in 1,000,000.

For the task of face detection, the initial rectangle features selected by AdaBoost are meaningful and easily interpreted. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks (see Fig. 5). This feature is relatively large in comparison with the detection sub-window, and should be somewhat insensitive to size and location of the face. The second feature selected relies on the property that the eyes are darker than the bridge of the nose.

In summary the 200-feature classifier provides initial evidence that a boosted classifier constructed from rectangle features is an effective technique for face detection. In terms of detection, these results are compelling but not sufficient for many real-world tasks. In terms of computation, this classifier is very fast, requiring 0.7 seconds to scan an 384 by 288 pixel image. Unfortunately, the most straightforward technique for improving detection performance, adding

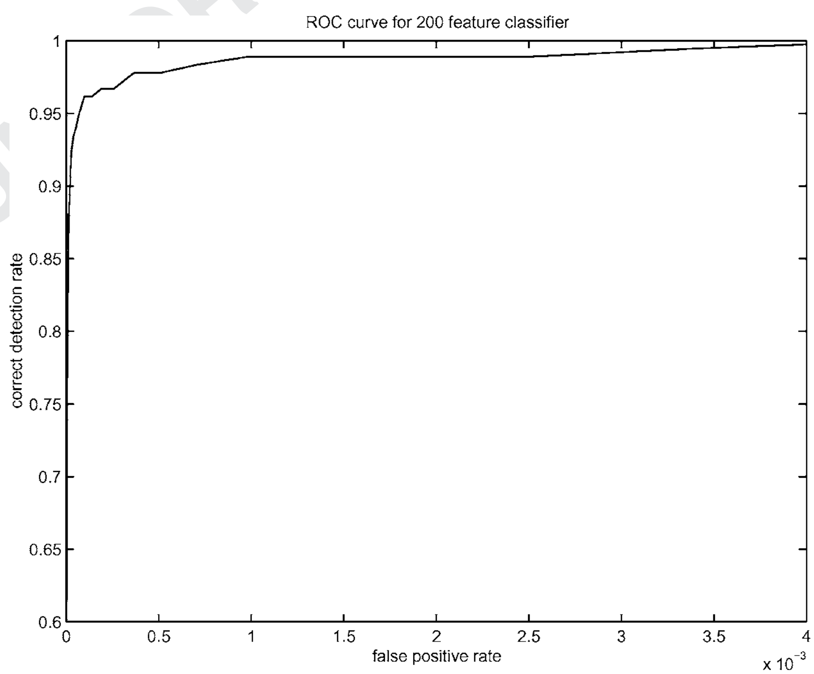

Figure 4. Receiver operating characteristic (ROC) curve for the 200 feature classifier.

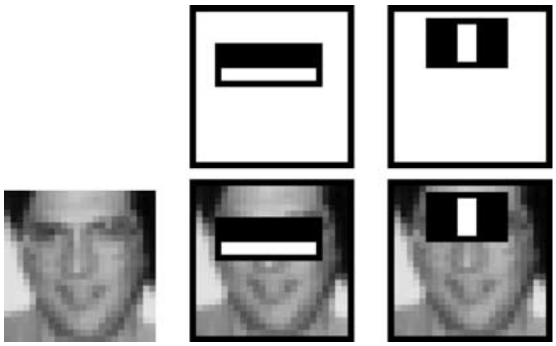

Figure 5. The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

features to the classifier, directly increases computation time.

4. The Attentional Cascade

This section describes an algorithm for constructing a cascade of classifiers which achieves increased detection performance while radically reducing computation time. The key insight is that smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances. Simpler classifiers are used to reject the majority of sub-windows before more complex classifiers are called upon to achieve low false positive rates.

Stages in the cascade are constructed by training classifiers using AdaBoost. Starting with a two-feature strong classifier, an effective face filter can be obtained by adjusting the strong classifier threshold to minimize false negatives. The initial AdaBoost threshold, $\frac{1}{2}\sum_{t=1}^{T}\alpha_t$, is designed to yield a low error rate on the training data. A lower threshold yields higher detection rates and higher false positive rates. Based on performance measured using a validation training set, the two-feature classifier can be adjusted to detect 100% of the faces with a false positive rate of 50%. See Fig. 5 for a description of the two features used in this classifier.

The detection performance of the two-feature classifier is far from acceptable as a face detection system. Nevertheless the classifier can significantly reduce the number of sub-windows that need further processing with very few operations:

1. Evaluate the rectangle features (requires between 6 and 9 array references per feature).
2. Compute the weak classifier for each feature (requires one threshold operation per feature).
3. Combine the weak classifiers (requires one multiply per feature, an addition, and finally a threshold).

A two feature classifier amounts to about 60 microprocessor instructions. It seems hard to imagine that any simpler filter could achieve higher rejection rates. By comparison, scanning a simple image template would require at least 20 times as many operations per sub-window.

The overall form of the detection process is that of a degenerate decision tree, what we call a "cascade" (Quinlan, 1986) (see Fig. 6). A positive result from the first classifier triggers the evaluation of a second classifier which has also been adjusted to achieve very high detection rates. A positive result from the second classifier triggers a third classifier, and so on. A negative outcome at any point leads to the immediate rejection of the sub-window.

The structure of the cascade reflects the fact that within any single image an overwhelming majority of sub-windows are negative. As such, the cascade attempts to reject as many negatives as possible at the earliest stage possible. While a positive instance will

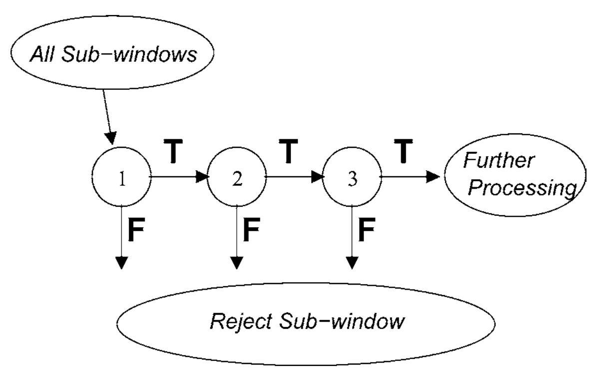

Figure 6. Schematic depiction of a the detection cascade. A series of classifiers are applied to every sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After several stages of processing the number of sub-windows have been reduced radically. Further processing can take any form such as additional stages of the cascade (as in our detection system) or an alternative detection system.

trigger the evaluation of every classifier in the cascade, this is an exceedingly rare event.

Much like a decision tree, subsequent classifiers are trained using those examples which pass through all the previous stages. As a result, the second classifier faces a more difficult task than the first. The examples which make it through the first stage are "harder" than typical examples. The more difficult examples faced by deeper classifiers push the entire receiver operating characteristic (ROC) curve downward. At a given detection rate, deeper classifiers have correspondingly higher false positive rates.

4.1. *Training a Cascade of Classifiers*

The cascade design process is driven from a set of detection and performance goals. For the face detection task, past systems have achieved good detection rates (between 85 and 95 percent) and extremely low false positive rates (on the order of 10^{-5} or 10^{-6}). The number of cascade stages and the size of each stage must be sufficient to achieve similar detection performance while minimizing computation.

Given a trained cascade of classifiers, the false positive rate of the cascade is

$$F = \prod_{i=1}^{K} f_i,$$

where F is the false positive rate of the cascaded classifier, K is the number of classifiers, and f_i is the false positive rate of the ith classifier on the examples that get through to it. The detection rate is

$$D = \prod_{i=1}^{K} d_i,$$

where D is the detection rate of the cascaded classifier, K is the number of classifiers, and d_i is the detection rate of the ith classifier on the examples that get through to it.

Given concrete goals for overall false positive and detection rates, target rates can be determined for each stage in the cascade process. For example a detection rate of 0.9 can be achieved by a 10 stage classifier if each stage has a detection rate of 0.99 (since $0.9 \approx 0.99^{10}$). While achieving this detection rate may sound like a daunting task, it is made significantly easier by the fact that each stage need only achieve a false positive rate of about 30% ($0.30^{10} \approx 6 \times 10^{-6}$).

The number of features evaluated when scanning real images is necessarily a probabilistic process. Any given sub-window will progress down through the cascade, one classifier at a time, until it is decided that the window is negative or, in rare circumstances, the window succeeds in each test and is labelled positive. The expected behavior of this process is determined by the distribution of image windows in a typical test set. The key measure of each classifier is its "positive rate", the proportion of windows which are labelled as potentially containing a face. The expected number of features which are evaluated is:

$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_j \right)$$

where N is the expected number of features evaluated, K is the number of classifiers, p_i is the positive rate of the ith classifier, and n_i are the number of features in the ith classifier. Interestingly, since faces are extremely rare, the "positive rate" is effectively equal to the false positive rate.

The process by which each element of the cascade is trained requires some care. The AdaBoost learning procedure presented in Section 3 attempts only to minimize errors, and is not specifically designed to achieve high detection rates at the expense of large false positive rates. One simple, and very conventional, scheme for trading off these errors is to adjust the threshold of the perceptron produced by AdaBoost. Higher thresholds yield classifiers with fewer false positives and a lower detection rate. Lower thresholds yield classifiers with more false positives and a higher detection rate. It is not clear, at this point, whether adjusting the threshold in this way preserves the training and generalization guarantees provided by AdaBoost.

The overall training process involves two types of tradeoffs. In most cases classifiers with more features will achieve higher detection rates and lower false positive rates. At the same time classifiers with more features require more time to compute. In principle one could define an optimization framework in which

- the number of classifier stages,
- the number of features, n_i, of each stage,
- the threshold of each stage

are traded off in order to minimize the expected number of features N given a target for F and D. Unfortunately finding this optimum is a tremendously difficult problem.

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

Table 2. The training algorithm for building a cascaded detector.

- User selects values for f, the maximum acceptable false positive rate per layer and d, the minimum acceptable detection rate per layer.
- User selects target overall false positive rate, F_{target}.
- P = set of positive examples
- N = set of negative examples
- $F_0 = 1.0$; $D_0 = 1.0$
- $i = 0$
- while $F_i > F_{target}$
  - $i \leftarrow i + 1$
  - $n_i = 0$; $F_i = F_{i-1}$
  - while $F_i > f \times F_{i-1}$
    - $n_i \leftarrow n_i + 1$
    - Use P and N to train a classifier with n_i features using AdaBoost
    - Evaluate current cascaded classifier on validation set to determine F_i and D_i.
    - Decrease threshold for the ith classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects F_i)
  - $N \leftarrow \emptyset$
  - If $F_i > F_{target}$ then evaluate the current cascaded detector on the set of non-face images and put any false detections into the set N

In practice a very simple framework is used to produce an effective classifier which is highly efficient. The user selects the maximum acceptable rate for f_i and the minimum acceptable rate for d_i. Each layer of the cascade is trained by AdaBoost (as described in Table 1) with the number of features used being increased until the target detection and false positive rates are met for this level. The rates are determined by testing the current detector on a validation set. If the overall target false positive rate is not yet met then another layer is added to the cascade. The negative set for training subsequent layers is obtained by collecting all false detections found by running the current detector on a set of images which do not contain any instances of faces. This algorithm is given more precisely in Table 2.

4.2. *Simple Experiment*

In order to explore the feasibility of the cascade approach two simple detectors were trained: a monolithic 200-feature classifier and a cascade of ten 20-feature classifiers. The first stage classifier in the cascade was trained using 5000 faces and 10000 non-face sub-windows randomly chosen from non-face images. The second stage classifier was trained on the same 5000 faces plus 5000 false positives of the first classifier. This process continued so that subsequent stages were trained using the false positives of the previous stage.

The monolithic 200-feature classifier was trained on the union of all examples used to train all the stages of the cascaded classifier. Note that without reference to the cascaded classifier, it might be difficult to select a set of non-face training examples to train the monolithic classifier. We could of course use all possible sub-windows from all of our non-face images, but this would make the training time impractically long. The sequential way in which the cascaded classifier is trained effectively reduces the non-face training set by throwing out easy examples and focusing on the "hard" ones.

Figure 7 gives the ROC curves comparing the performance of the two classifiers. It shows that there is little difference between the two in terms of accuracy. However, there is a big difference in terms of speed. The cascaded classifier is nearly 10 times faster since its first stage throws out most non-faces so that they are never evaluated by subsequent stages.

4.3. *Detector Cascade Discussion*

There is a hidden benefit of training a detector as a sequence of classifiers which is that the effective number of negative examples that the final detector sees can be very large. One can imagine training a single large classifier with many features and then trying to speed up its running time by looking at partial sums of features and stopping the computation early if a partial sum is below the appropriate threshold. One drawback of such an approach is that the training set of negative examples would have to be relatively small (on the order of 10,000 to maybe 100,000 examples) to make training feasible. With the cascaded detector, the final layers of the cascade may effectively look through hundreds of millions of negative examples in order to find a set of 10,000 negative examples that the earlier layers of the cascade fail on. So the negative training set is much larger and more focused on the hard examples for a cascaded detector.

A notion similar to the cascade appears in the face detection system described by Rowley et al. (1998). Rowley et al. trained two neural networks. One network was moderately complex, focused on a small region of the image, and detected faces with a low false positive rate. They also trained a second neural network which

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

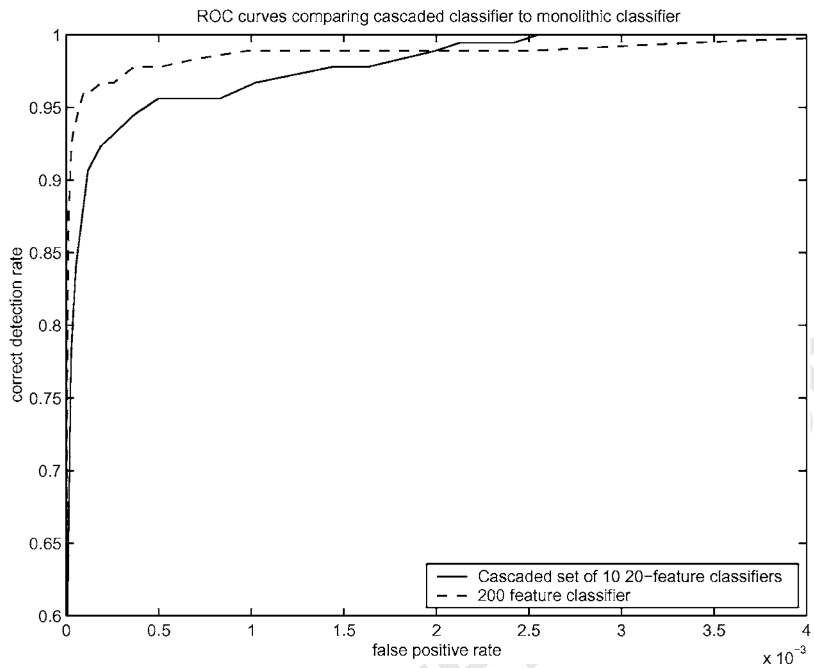

Figure 7. ROC curves comparing a 200-feature classifier with a cascaded classifier containing ten 20-feature classifiers. Accuracy is not significantly different, but the speed of the cascaded classifier is almost 10 times faster.

was much faster, focused on a larger regions of the image, and detected faces with a higher false positive rate. Rowley et al. used the faster second network to prescreen the image in order to find candidate regions for the slower more accurate network. Though it is difficult to determine exactly, it appears that Rowley et al.'s two network face system is the fastest existing face detector.[7] Our system uses a similar approach, but it extends this two stage cascade to include 38 stages.

The structure of the cascaded detection process is essentially that of a degenerate decision tree, and as such is related to the work of Amit and Geman (1999). Unlike techniques which use a fixed detector, Amit and Geman propose an alternative point of view where unusual co-occurrences of simple image features are used to trigger the evaluation of a more complex detection process. In this way the full detection process need not be evaluated at many of the potential image locations and scales. While this basic insight is very valuable, in their implementation it is necessary to first evaluate some feature detector at every location. These features are then grouped to find unusual co-occurrences. In practice, since the form of our detector and the features that it uses are extremely efficient, the amortized cost of evaluating our detector at *every scale and location* is much faster than finding and grouping edges throughout the image.

In recent work Fleuret and Geman (2001) have presented a face detection technique which relies on a "chain" of tests in order to signify the presence of a face at a particular scale and location. The image properties measured by Fleuret and Geman, disjunctions of fine scale edges, are quite different than rectangle features which are simple, exist at all scales, and are somewhat interpretable. The two approaches also differ radically in their learning philosophy. Because Fleuret and Geman's learning process does not use negative examples their approach is based more on density estimation, while our detector is purely discriminative. Finally the false positive rate of Fleuret and Geman's approach appears to be higher than that of previous approaches like Rowley et al. and this approach. In the published paper the included example images each had between 2 and 10 false positives. For many practical tasks, it is important that the expected number of false positives in any image be less than one (since in many tasks the expected number of true positives is less than one as well). Unfortunately the paper does not report quantitative detection and false positive results on standard datasets.

5. Results

This section describes the final face detection system. The discussion includes details on the structure and training of the cascaded detector as well as results on a large real-world testing set.

5.1. Training Dataset

The face training set consisted of 4916 hand labeled faces scaled and aligned to a base resolution of 24 by 24 pixels. The faces were extracted from images downloaded during a random crawl of the world wide web. Some typical face examples are shown in Fig. 8. The training faces are only roughly aligned. This was done by having a person place a bounding box around each face just above the eyebrows and about half-way between the mouth and the chin. This bounding box was then enlarged by 50% and then cropped and scaled to 24 by 24 pixels. No further alignment was done (i.e. the eyes are not aligned). Notice that these examples contain more of the head than the examples used by Rowley et al. (1998) or Sung and Poggio (1998). Initial experiments also used 16 by 16 pixel training images in which the faces were more tightly cropped, but got slightly worse results. Presumably the 24 by 24 examples include extra visual information such as the contours of the chin and cheeks and the hair line which help to improve accuracy. Because of the nature of the features used, the larger sized sub-windows do not slow performance. In fact, the additional information contained in the larger sub-windows can be used to reject non-faces earlier in the detection cascade.

5.2. Structure of the Detector Cascade

The final detector is a 38 layer cascade of classifiers which included a total of 6060 features.

The first classifier in the cascade is constructed using two features and rejects about 50% of non-faces while correctly detecting close to 100% of faces. The next classifier has ten features and rejects 80% of non-faces while detecting almost 100% of faces. The next two layers are 25-feature classifiers followed by three 50-feature classifiers followed by classifiers with a

Figure 8. Example of frontal upright face images used for training.

variety of different numbers of features chosen according to the algorithm in Table 2. The particular choices of number of features per layer was driven through a trial and error process in which the number of features were increased until a significant reduction in the false positive rate could be achieved. More levels were added until the false positive rate on the validation set was nearly zero while still maintaining a high correct detection rate. The final number of layers, and the size of each layer, are not critical to the final system performance. The procedure we used to choose the number of features per layer was guided by human intervention (for the first 7 layers) in order to reduce the training time for the detector. The algorithm described in Table 2 was modified slightly to ease the computational burden by specifying a minimum number of features per layer by hand and by adding more than 1 feature at a time. In later layers, 25 features were added at a time before testing on the validation set. This avoided having to test the detector on the validation set for every single feature added to a classifier.

The non-face sub-windows used to train the first level of the cascade were collected by selecting random sub-windows from a set of 9500 images which did not contain faces. The non-face examples used to train subsequent layers were obtained by scanning the partial cascade across large non-face images and collecting false positives. A maximum of 6000 such non-face sub-windows were collected for each layer. There are approximately 350 million non-face sub-windows contained in the 9500 non-face images.

Training time for the entire 38 layer detector was on the order of weeks on a single 466 MHz AlphaStation XP900. We have since parallelized the algorithm to make it possible to train a complete cascade in about a day.

5.3. *Speed of the Final Detector*

The speed of the cascaded detector is directly related to the number of features evaluated per scanned sub-window. As discussed in Section 4.1, the number of features evaluated depends on the images being scanned. Since a large majority of the sub-windows are discarded by the first two stages of the cascade, an average of 8 features out of a total of 6060 are evaluated per sub-window (as evaluated on the MIT + CMU (Rowley et al., 1998). On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds (using a starting scale of 1.25 and a step size of 1.5 described below). This is roughly 15 times faster than the Rowley-Baluja-Kanade detector (Rowley et al., 1998) and about 600 times faster than the Schneiderman-Kanade detector (Schneiderman and Kanade, 2000).

5.4. *Image Processing*

All example sub-windows used for training were variance normalized to minimize the effect of different lighting conditions. Normalization is therefore necessary during detection as well. The variance of an image sub-window can be computed quickly using a pair of integral images. Recall that $\sigma^2 = m^2 - \frac{1}{N}\sum x^2$, where σ is the standard deviation, m is the mean, and x is the pixel value within the sub-window. The mean of a sub-window can be computed using the integral image. The sum of squared pixels is computed using an integral image of the image squared (i.e. two integral images are used in the scanning process). During scanning the effect of image normalization can be achieved by post multiplying the feature values rather than operating on the pixels.

5.5. *Scanning the Detector*

The final detector is scanned across the image at multiple scales and locations. Scaling is achieved by scaling the detector itself, rather than scaling the image. This process makes sense because the features can be evaluated at any scale with the same cost. Good detection results were obtained using scales which are a factor of 1.25 apart.

The detector is also scanned across location. Subsequent locations are obtained by shifting the window some number of pixels Δ. This shifting process is affected by the scale of the detector: if the current scale is s the window is shifted by $[s\Delta]$, where [] is the rounding operation.

The choice of Δ affects both the speed of the detector as well as accuracy. Since the training images have some translational variability the learned detector achieves good detection performance in spite of small shifts in the image. As a result the detector sub-window can be shifted more than one pixel at a time. However, a step size of more than one pixel tends to decrease the detection rate slightly while also decreasing the number of false positives. We present results for two different step sizes.

5.6. Integration of Multiple Detections

Since the final detector is insensitive to small changes in translation and scale, multiple detections will usually occur around each face in a scanned image. The same is often true of some types of false positives. In practice it often makes sense to return one final detection per face. Toward this end it is useful to postprocess the detected sub-windows in order to combine overlapping detections into a single detection.

In these experiments detections are combined in a very simple fashion. The set of detections are first partitioned into disjoint subsets. Two detections are in the same subset if their bounding regions overlap. Each partition yields a single final detection. The corners of the final bounding region are the average of the corners of all detections in the set.

In some cases this postprocessing decreases the number of false positives since an overlapping subset of false positives is reduced to a single detection.

5.7. Experiments on a Real-World Test Set

We tested our system on the MIT + CMU frontal face test set (Rowley et al., 1998). This set consists of 130 images with 507 labeled frontal faces. A ROC curve showing the performance of our detector on this test set is shown in Fig. 9. To create the ROC curve the threshold of the perceptron on the final layer classifier is adjusted from $+\infty$ to $-\infty$. Adjusting the threshold to $+\infty$ will yield a detection rate of 0.0 and a false positive rate of 0.0. Adjusting the threshold to $-\infty$, however, increases both the detection rate and false positive rate, but only to a certain point. Neither rate can be higher than the rate of the detection cascade minus the final layer. In effect, a threshold of $-\infty$ is equivalent to removing that layer. Further increasing the detection and false positive rates requires decreasing the threshold of the next classifier in the cascade. Thus, in order to construct a complete ROC curve, classifier layers are removed. We use the *number* of false positives as opposed to the *rate* of false positives for the *x*-axis of the ROC curve to facilitate comparison with other systems. To compute the false positive rate, simply divide by the total number of sub-windows scanned. For the case of $\Delta = 1.0$ and starting scale = 1.0, the number of sub-windows scanned is 75,081,800. For $\Delta = 1.5$ and starting scale = 1.25, the number of sub-windows scanned is 18,901,947.

Unfortunately, most previous published results on face detection have only included a single operating

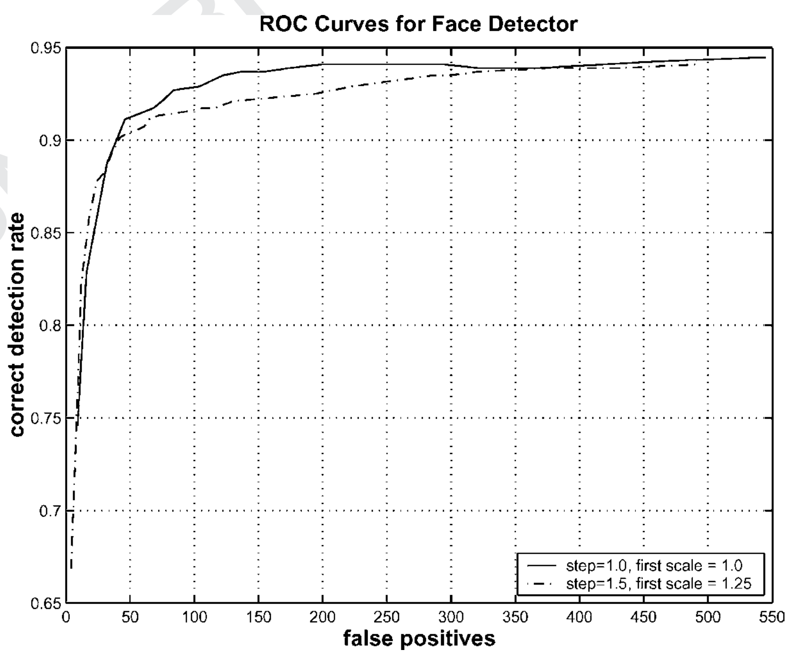

Figure 9. ROC curves for our face detector on the MIT + CMU test set. The detector was run once using a step size of 1.0 and starting scale of 1.0 (75,081,800 sub-windows scanned) and then again using a step size of 1.5 and starting scale of 1.25 (18,901,947 sub-windows scanned). In both cases a scale factor of 1.25 was used.

Table 3. Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2%	93.7%	–
Rowley-Baluja-Kanade	83.2%	86.0%	–	–	–	89.2%	90.1%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–

regime (i.e. single point on the ROC curve). To make comparison with our detector easier we have listed our detection rate for the same false positive rate reported by the other systems. Table 3 lists the detection rate for various numbers of false detections for our system as well as other published systems. For the Rowley-Baluja-Kanade results (Rowley et al., 1998), a number of different versions of their detector were tested yielding a number of different results. While these various results are not actually points on a ROC curve for a particular detector, they do indicate a number of different performance points that can be achieved with their approach. They did publish ROC curves for two of their detectors, but these ROC curves did not represent their best results. For the Roth-Yang-Ahuja detector (Roth et al., 2000), they reported their result on the MIT + CMU test set minus 5 images containing line drawn faces removed. So their results are for a subset of the MIT + CMU test set containing 125 images with 483 faces. Presumably their detection rate would be lower if the full test set was used. The parentheses around their detection rate indicates this slightly different test set. The Sung and Poggio face detector (Sung and Poggio, 1998) was tested on the MIT subset of the MIT + CMU test set since the CMU portion did not exist yet. The MIT test set contains 23 images with 149 faces. They achieved a detection rate of 79.9% with 5 false positives. Our detection rate with 5 false positives is 77.8% on the MIT test set.

Figure 10 shows the output of our face detector on some test images from the MIT + CMU test set.

5.7.1. A Simple Voting Scheme Further Improves Results. The best results were obtained through the combination of three detectors trained using different initial negative examples, slightly different weighting on negative versus positive errors, and slightly different criteria for trading off false positives for classifier size. These three systems performed similarly on the final task, but in some cases errors were different. The detection results from these three detectors were combined by retaining only those detections where at least 2 out of 3 detectors agree. This improves the final detection rate as well as eliminating more false positives. Since detector errors are not uncorrelated, the combination results in a measurable, but modest, improvement over the best single detector.

5.7.2. Failure Modes. By observing the performance of our face detector on a number of test images we have noticed a few different failure modes.

The face detector was trained on frontal, upright faces. The faces were only very roughly aligned so there is some variation in rotation both in plane and out of plane. Informal observation suggests that the face detector can detect faces that are tilted up to about ± 15 degrees in plane and about ± 45 degrees out of plane (toward a profile view). The detector becomes unreliable with more rotation than this.

We have also noticed that harsh backlighting in which the faces are very dark while the background is relatively light sometimes causes failures. It is interesting to note that using a nonlinear variance normalization based on robust statistics to remove outliers improves the detection rate in this situation. The problem with such a normalization is the greatly increased computational cost within our integral image framework.

Finally, our face detector fails on significantly occluded faces. If the eyes are occluded for example, the detector will usually fail. The mouth is not as important and so a face with a covered mouth will usually still be detected.

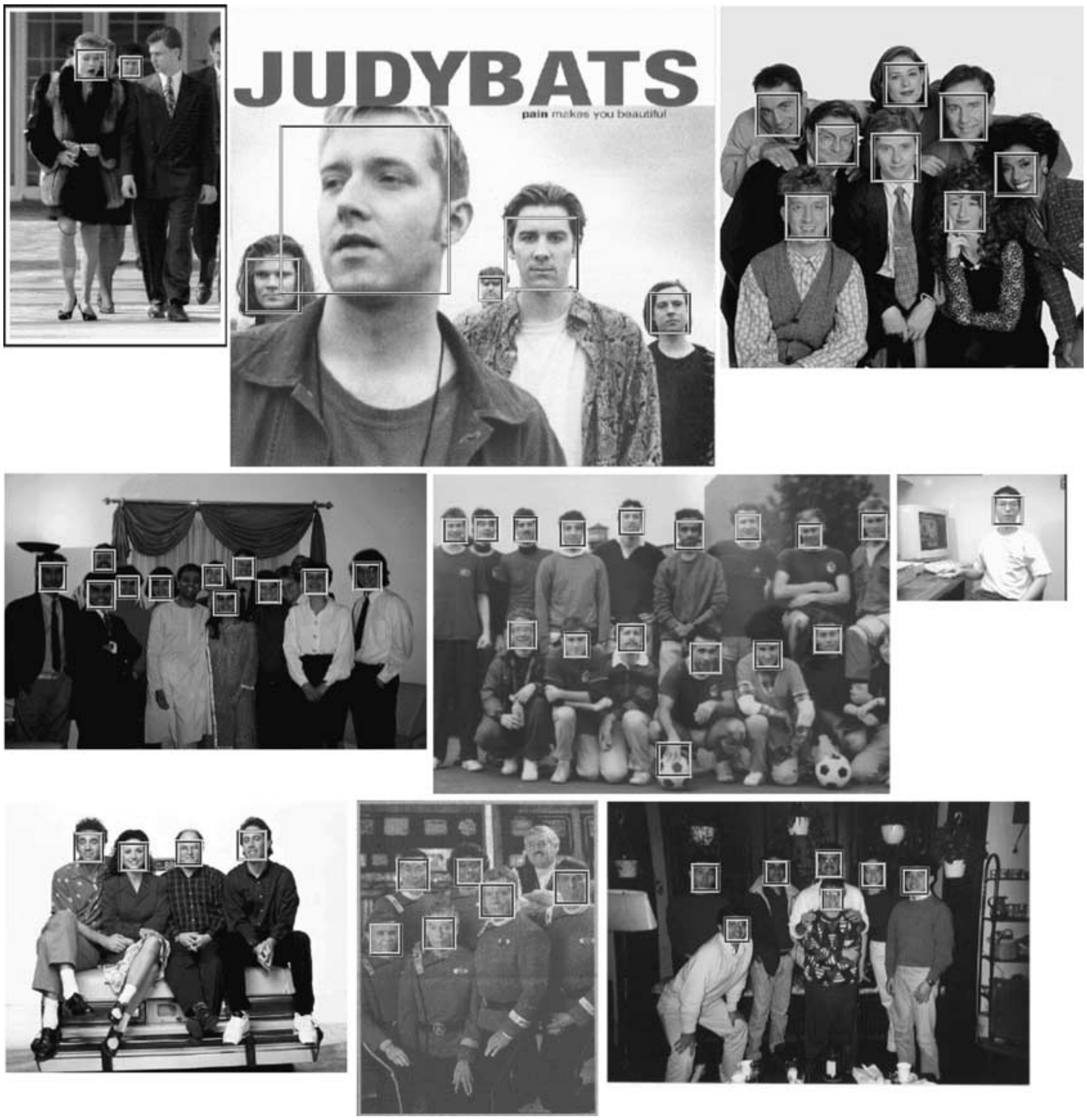

Figure 10. Output of our face detector on a number of test images from the MIT + CMU test set.

6. Conclusions

We have presented an approach for face detection which minimizes computation time while achieving high detection accuracy. The approach was used to construct a face detection system which is approximately 15 times faster than any previous approach. Preliminary experiments, which will be described elsewhere, show that highly efficient detectors for other objects, such as pedestrians or automobiles, can also be constructed in this way.

This paper brings together new algorithms, representations, and insights which are quite generic and may well have broader application in computer vision and image processing.

The first contribution is a new a technique for computing a rich set of image features using the integral image. In order to achieve true scale invariance, almost all face detection systems must operate on multiple image scales. The integral image, by eliminating the need to compute a multi-scale image pyramid, reduces the initial image processing required for face detection

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

significantly. Using the integral image, face detection is completed in almost the same time as it takes for an image pyramid to be computed.

While the integral image should also have immediate use for other systems which have used Haar-like features such as Papageorgiou et al. (1998), it can foreseeably have impact on any task where Haar-like features may be of value. Initial experiments have shown that a similar feature set is also effective for the task of parameter estimation, where the expression of a face, the position of a head, or the pose of an object is determined.

The second contribution of this paper is a simple and efficient classifier built from computationally efficient features using AdaBoost for feature selection. This classifier is clearly an effective one for face detection and we are confident that it will also be effective in other domains such as automobile or pedestrian detection. Furthermore, the idea of an aggressive and effective technique for feature selection should have impact on a wide variety of learning tasks. Given an effective tool for feature selection, the system designer is free to define a very large and very complex set of features as input for the learning process. The resulting classifier is nevertheless computationally efficient, since only a small number of features need to be evaluated during run time. Frequently the resulting classifier is also quite simple; within a large set of complex features it is more likely that a few critical features can be found which capture the structure of the classification problem in a straightforward fashion.

The third contribution of this paper is a technique for constructing a cascade of classifiers which radically reduces computation time while improving detection accuracy. Early stages of the cascade are designed to reject a majority of the image in order to focus subsequent processing on promising regions. One key point is that the cascade presented is quite simple and homogeneous in structure. Previous approaches for attentive filtering, such as Itti et al. (1998) propose a more complex and heterogeneous mechanism for filtering. Similarly Amit and Geman (1999) propose a hierarchical structure for detection in which the stages are quite different in structure and processing. A homogeneous system, besides being easy to implement and understand, has the advantage that simple tradeoffs can be made between processing time and detection performance.

Finally this paper presents a set of detailed experiments on a difficult face detection dataset which has been widely studied. This dataset includes faces under a very wide range of conditions including: illumination, scale, pose, and camera variation. Experiments on such a large and complex dataset are difficult and time consuming. Nevertheless systems which work under these conditions are unlikely to be brittle or limited to a single set of conditions. More importantly conclusions drawn from this dataset are unlikely to be experimental artifacts.

Acknowledgments

The authors would like to thank T.M. Murali, Jim Rehg, Tat-jen Cham, Rahul Sukthankar, Vladimir Pavlovic, and Thomas Leung for the their helpful comments. Henry Rowley was extremely helpful in providing implementations of his face detector for comparison with our own.

Notes

1. Supervised refers to the fact that the attentional operator is trained to detect examples of a particular class.
2. Henry Rowley very graciously supplied us with implementations of his detection system for direct comparison. Reported results are against his fastest system. It is difficult to determine from the published literature, but the Rowley-Baluja-Kanade detector is widely considered the fastest detection system and has been heavily tested on real-world problems.
3. A complete basis has no linear dependence between basis elements and has the same number of elements as the image space, in this case 576. The full set of 160,000 features is many times over-complete.
4. There is a close relation to "summed area tables" as used in graphics (Crow, 1984). We choose a different name here in order to emphasize its use for the analysis of images, rather than for texture mapping.
5. The availability of custom hardware and the appearance of special instruction sets like Intel MMX can change this analysis. It is nevertheless instructive to compare performance assuming conventional software algorithms.
6. In the case where the weak learner is a perceptron learning algorithm, the final boosted classifier is a two layer perceptron. A two layer perceptron is in principle much more powerful than any single layer perceptron.
7. Among other published face detection systems some are potentially faster. These have either neglected to discuss performance in detail, or have never published detection and false positive rates on a large and difficult training set.

References

Amit, Y. and Geman, D. 1999. A computational model for visual selection. *Neural Computation*, 11:1691–1715.

P1: ***
International Journal of Computer Vision KL2213-05/5263672 January 10, 2004 20:17

Crow, F. 1984. Summed-area tables for texture mapping. In *Proceedings of SIGGRAPH*, 18(3):207–212.

Fleuret, F. and Geman, D. 2001. Coarse-to-fine face detection. *Int. J. Computer Vision*.

Au: Pls. provide vol & page range

Freeman, W.T. and Adelson, E.H. 1991. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906.

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

Greenspan, H., Belongie, S., Gooodman, R., Perona, P., Rakshit, S., and Anderson, C. 1994. Overcomplete steerable pyramid filters and rotation invariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Itti, L., Koch, C., and Niebur, E. 1998. A model of saliency-based visual attention for rapid scene analysis. *IEEE Patt. Anal. Mach. Intell.*, 20(11):1254-–1259.

John, G., Kohavi, R., and Pfeger, K. 1994. Irrelevant features and the subset selection problem. In *Machine Learning Conference Proceedings*.

Osuna, E., Freund, R., and Girosi, F. 1997a. Training support vector machines: An application to face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Osuna, E., Freund, R., and Girosi, F. 1997b. Training support vector machines: an application to face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Papageorgiou, C., Oren, M., and Poggio, T. 1998. A general framework for object detection. In *International Conference on Computer Vision*.

Quinlan, J. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

Roth, D., Yang, M., and Ahuja, N. 2000. A snowbased face detector. In *Neural Information Processing* 12.

Rowley, H., Baluja, S., and Kanade, T. 1998. Neural network-based face detection. *IEEE Patt. Anal. Mach. Intell.*, 20:22–38.

Schapire, R.E., Freund, Y., Bartlett, P., and Lee, W.S. 1997. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning.*

Schapire, R.E., Freund, Y., Bartlett, P., and Lee, W.S. 1998. Boosting the margin: A new explanation for the effectiveness of voting methods. *Ann. Stat.*, 26(5):1651–1686.

Schneiderman, H. and Kanade, T. 2000. A statistical method for 3D object detection applied to faces and cars. In *International Conference on Computer Vision*.

Simard, P.Y., Bottou, L., Haffner, P., and LeCun, Y. (1999). Boxlets: A fast convolution algorithm for signal processing and neural networks. In M. Kearns, S. Solla, and D. Cohn (Eds.), *Advances in Neural Information Processing Systems*, vol. 11, pp. 571–577.

Sung, K. and Poggio, T. 1998. Example-based learning for view-based face detection. *IEEE Patt. Anal. Mach. Intell.*, 20:39–51.

Tieu, K. and Viola, P. 2000. Boosting image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Tsotsos, J., Culhane, S., Wai, W., Lai, Y., Davis, N., and Nuflo, F. 1995. Modeling visual-attention via selective tuning. *Artificial Intelligence Journal*, 78(1/2):507–545.

Webb, A. 1999. Statistical Pattern Recognition. Oxford University Press: New York.