SECOND INTERNATIONAL WORKSHOP ON STATISTICAL AND COMPUTATIONAL THEORIES OF VISION – MODELING, LEARNING, COMPUTING, AND SAMPLING

VANCOUVER, CANADA, JULY 13, 2001.

Robust Real-time Object Detection

Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

Abstract

This paper describes a visual object 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 learning algorithm, based on AdaBoost, which selects a small number of critical visual features and yields extremely efficient classifiers [6]. 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 object-like regions. A set of experiments in the domain of face detection are presented. The system yields face detection performace comparable to the best previous systems [18, 13, 16, 12, 1]. Implemented on a conventional desktop, face detection proceeds at 15 frames per second.

1. Introduction

This paper brings together new algorithms and insights to construct a framework for robust and extremely rapid object detection. This framework is demonstrated on, and in part motivated by, the task of face 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 [18, 13, 16, 12, 1]. 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 object 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. our detection system does not work directly with image intensities [10]. Like 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 [3] for texture mapping). The integral image can be computed from an image using a few operations per pixel. Once computed, any one of these Harr-like features can be computed at any scale or location in *constant* time.

The second contribution of this paper is a method for constructing a classifier by selecting a small number of important features using AdaBoost [6]. Within any image sub-window the total number of Harr-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, feature selection is achieved through a simple modification of the AdaBoost procedure: the weak learner is constrained so that each weak classifier returned can depend on only a single feature [2]. 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 [14, 9, 10].

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 focussing 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 an object might occur [19, 8, 1]. 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, object 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. The term supervised refers to the fact that the attentional operator is trained to detect examples of a particular class. In the domain of face detection it is possible to achieve fewer than 1% false negatives and 40% false positives using a classifier which can be evaluated in 20 simple operations (approximately 60 microprocessor instructions). The effect of this filter is to reduce by over one half the number of locations where the final detector must be evaluated.

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 Amit and Geman [1].

The complete face detection cascade has 32 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.[1] [13]

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 the Compaq iPaq handheld and have achieved detection at two frames per second (this device has a low power 200 mips *Strong Arm* processor which lacks floating point hardware).

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 variant 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 object 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.

[1]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.

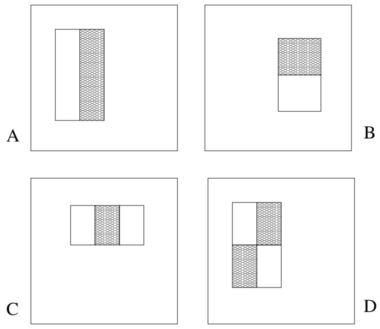

Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature,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

2 Features

Our object 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. [10]. 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 Figure 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 24x24, the exhaustive set of rectangle features is quite large, 45,396 . Note that unlike the Haar basis, the set of rectangle features is overcomplete[2].

2.1 Integral Image

Rectangle features can be computed very rapidly using an intermediate representation for the image which we call the integral image.[3] The integral image at location x, y contains the sum of the pixels above and to

[2]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 45,396 thousand features is many times over-complete.

[3]There is a close relation to "summed area tables" as used in graphics [3]. We choose a different name here in order to emphasize its use for the analysis of images, rather than for texture mapping.

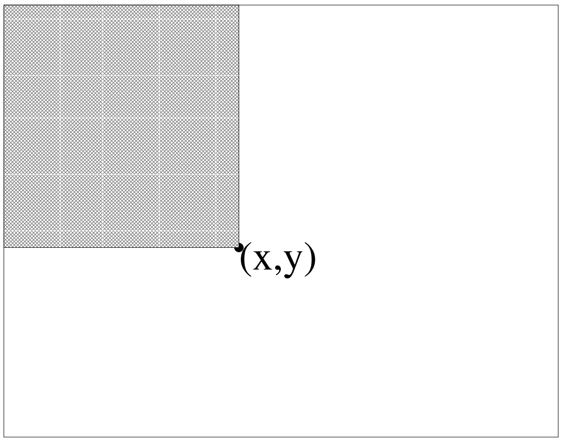

Figure 2: The value of the integral image at point (x, y) is the sum of all the pixels above and to the left.

the left of x, y, inclusive:

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

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

$$s(x,y) = s(x,y-1) + i(x,y) \quad (1)$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \quad (2)$$

(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 Figure 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. [17]. 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:

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

The authors go on to show that convolution can be significantly accelerated if the derivatives of f and g are

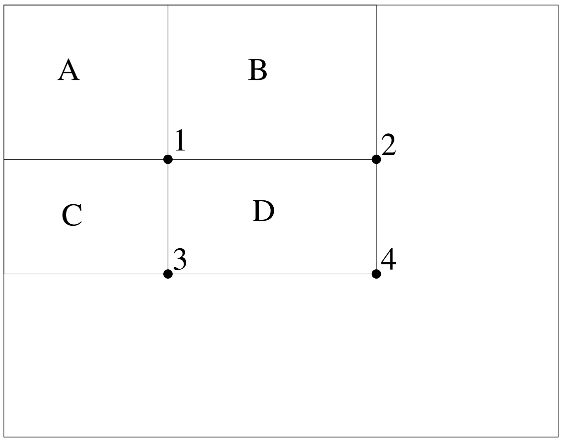

Figure 3: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can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

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

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

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 [5, 7]. Steerable filters, and their relatives, are excellent for the detailed analysis of boundaries, image compression, and texture analysis. In contrast rectangle features, while sensitive to the presence of edges, bars, and other simple image structure, are quite coarse. Unlike steerable filters the only orientations available are vertical and horizontal. It appears as though the set of rectangle features do however provide a rich image representation which supports effective learning. The extreme computational efficiency of rectangle features provides ample compensation for their limited flexibility.

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 object detection systems, our detector scans the input at many scales; starting at the base scale in which objects are detected at a size of 24x24 pixels, the image is scanned at 11 scales each a factor of 1.25 larger than the last. The conventional approach is to compute a pyramid of 11 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 on conventional hardware it is extremely difficult to compute a pyramid at 15 frames per second[4].

In contrast we have defined a meaningful set of 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 little 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, less time than is required to evaluate the 11 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 [18]. Rowley, Baluja, and Kanade use a small set of simple image features and a neural network [13]. Osuna, et al. used a support vector machine [9]. More recently Roth et al. have proposed a new and unusual image representation and have used the Winnow learning procedure [12].

Recall that there are 45,396 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 [6]. 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

[4]The total number of pixels in the 11 level pyramid is about $55 * 384 * 288 = 6082560$. Given that each pixel requires 10 operations to compute, the pyramid requires about 60,000,000 operations. About 900,000,000 operations per second are required to acheive a processing rate of 15 frames per second.

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.[5]

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 [15]. 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 [2] 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 ($h_j(x)$) thus consists of a feature (f_j), a threshold (θ_j) and a parity (p_j) indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

Here x is a 24x24 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.

[5] 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.

- 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,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error 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:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

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.

3.1 Learning Discussion

Many general feature selection procedures have been proposed (see chapter 8 of [20] 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. proposed a scheme for feature selection based on feature variance [10]. 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. propose a feature selection process based on the Winnow exponential perceptron learning rule [12]. 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 Figure 4). Given a detection rate of 95% the classifier yielded a false positive rate of 1 in 14084 on a testing dataset.

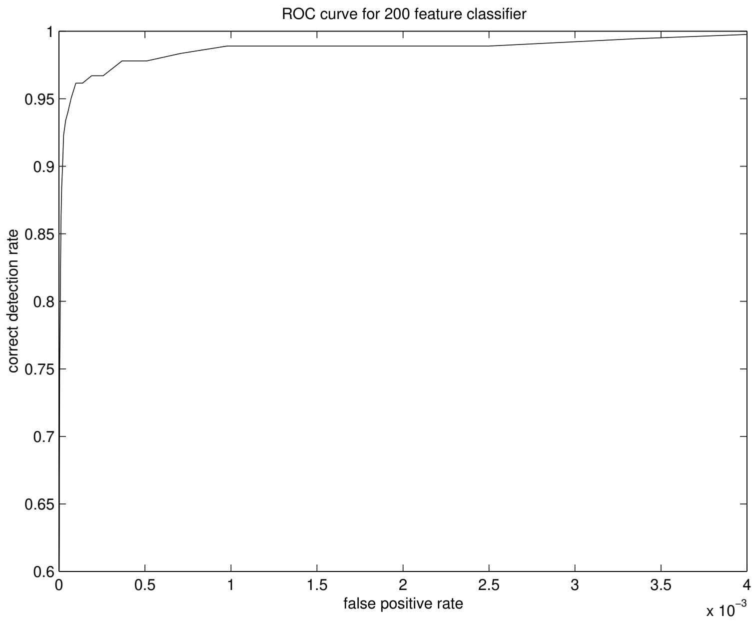

Figure 4: Reciever operating characteristic (ROC) curve for the 200 feature classifier.

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 Figure 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 object detection. In terms of detection, these results are compelling but not sufficient for many real-world tasks. In terms of computation, this classifier is probably faster than any other published system, requiring 0.7 seconds to scan an 384 by 288 pixel image. Unfortunately, the most straightforward technique for improving detection performance, adding features to the classifier,

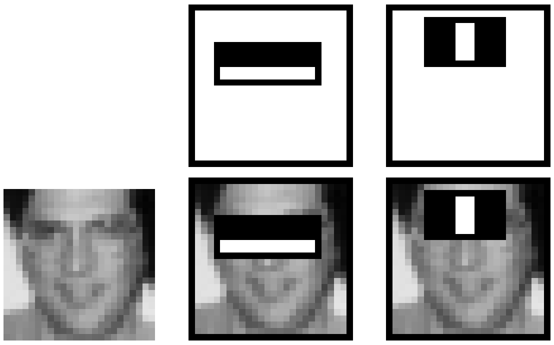

Figure 5: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature 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.

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 40%. See Figure 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 an object detection system. Nevertheless the classifier can significantly reduce the number 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, or a single layer perceptron, 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" [11] (see Figure 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.

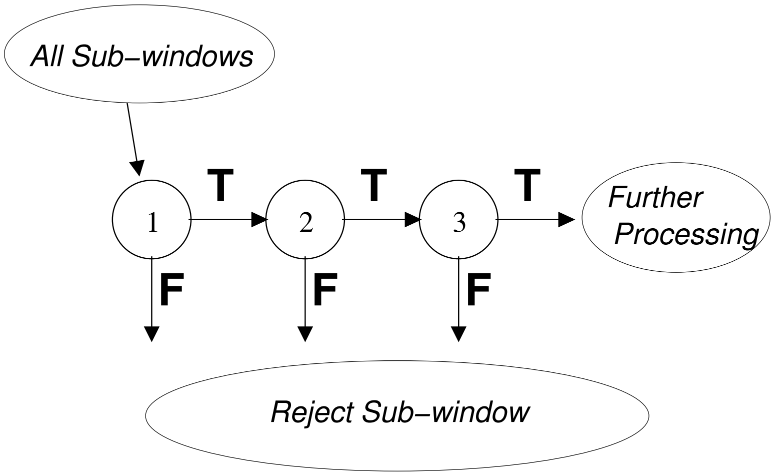

Figure 6: Schematic depiction of a the detection cascade. A series of classifiers are applied to every sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After 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.

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 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 reciever 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 the object of interest. 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 objects 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.

In practice a very simple framework is used to produce an effective classifier which is highly efficient. The user selects the minimum acceptable rates for f_i and 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 positves 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 the object. 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.

- 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 dectections into the set N

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

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 stage.

4.3 Detector Cascade Discussion

A notion similar to the cascade appears in the face detection system described by Rowley et al. [13]. 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 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

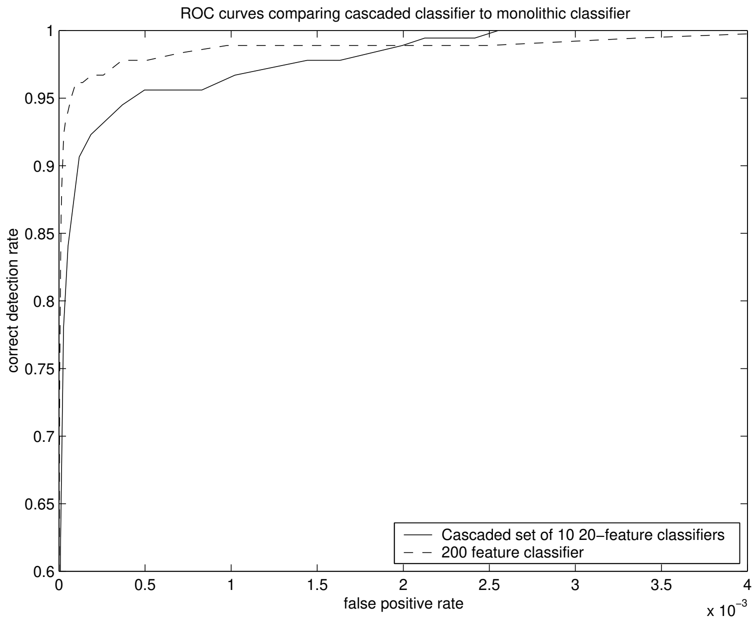

Figure 7: ROC curves comparing a 200-feature classifier with a cascaded classifier containing ten 20-feature classifiers. Accuracy is not significantly different, but the speed of the cascaded classifier is almost 10 times faster.

al.'s two network face system is the fastest existing face detector.[6] Our system uses a similar approach, but it extends this two stage cascade to include 32 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 [1]. 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 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 [4]. 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

[6]Other published detectors have either neglected to discuss performance in detail, or have never published detection and false positive rates on a large and difficult training set.

in their learning philosophy. The motivation for Fleuret and Geman's learning process is density estimation and density discrimination, 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. Unfortunately the paper does not report quantitative results of this kind. The included example images each have between 2 and 10 false positives.

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 Figure 8. Notice that these examples contain more of the head than the examples used by Rowley or et al. [13] or Sung [18]. 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 could be used to reject non-faces earlier in the detection cascade.

5.2 Structure of the Detector Cascade

The final detector is a 32 layer cascade of classifiers which included a total of 4297 features.

The first classifier in the cascade is constructed using two features and rejects about 60% of non-faces while correctly detecting close to 100% of faces. The next classifier has five features and rejects 80% of non-faces while detecting almost 100% of faces. The next three layers are 20-feature classifiers followed by two 50-feature classifiers followed by five 100-feature classifiers and then twenty 200-feature classifiers. 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 two, five and first twenty-feature classifiers were trained with the 4916 faces and 10,000 non-face sub-windows (also of size 24 by 24 pixels) using the Adaboost training procedure described in Table 1. The non-face sub-windows were collected by selecting random sub-windows from a set of 9500 images which

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

did not contain faces. Different sets of non-face sub-windows were used for training the different classifiers to ensure that they were somewhat independent and didn't use the same features.

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 32 layer detector was on the order of weeks on a single 466 MHz AlphaStation XP900. During this laborious training process several improvements to the learning algorithm were discovered. These improvements, which will be described elsewhere, yield a 100 fold decrease in training time.

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. Evaluated on the MIT+CMU test set [13], an average of 8 features out of a total of 4297 are evaluated per sub-window. This is possible because a large majority of sub-windows are rejected by the first or second layer in the cascade. 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 [13] and about 600 times faster than the Schneiderman-Kanade detector [16].

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 results were obtained using a set of scales 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 [13]. 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 Figure 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 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 [13], 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. Thus the detection rates listed in the table below for the Rowley-Baluja-Kanade detector are actually results for different versions of their detector. For the Roth-Yang-Ahuja detector [12], 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.

Detector \ False detections	10	31	50	65	78	95	110	167	422
Viola-Jones	78.3%	85.2%	88.8%	89.8%	90.1%	90.8%	91.1%	91.8%	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	-	90.1%	89.9%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-	-	-

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

The Sung and Poggio face detector [18] 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.

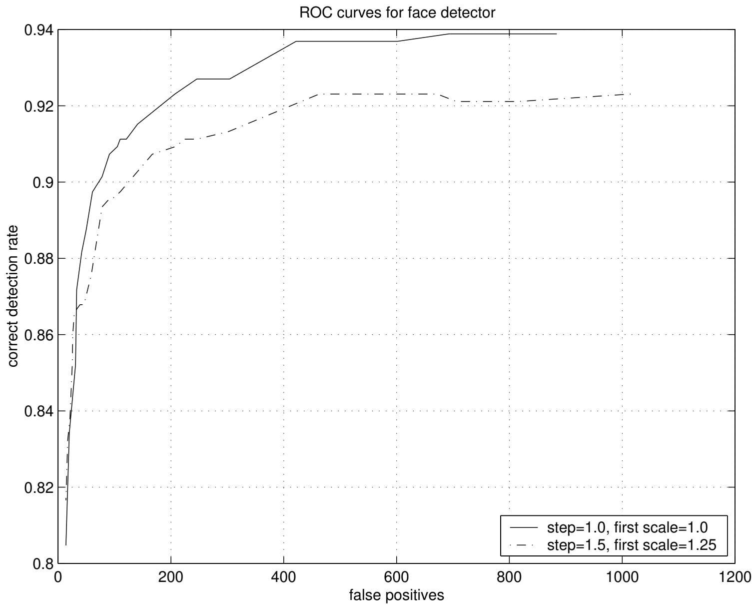

Figure 9: ROC curves for our face detector on the MIT+CMU test set. The detector was run once using a step size of 1.0 and starting scale of 1.0 (75,081,800 sub-windows scanned) and then again using a step size of 1.5 and starting scale of 1.25 (18,901,947 sub-windows scanned). In both cases a scale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

6 Conclusions

We have presented an approach for object detection which minimizes computation time while achieving high detection accuracy. The approach was used to construct a face detection system which is approximately 15 faster than any previous approach. Preliminary experiments, which will be described elsewhere, show that highly efficient detectors for other objects, such as pedestrians, 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 object detection systems must operate on muliple image scales. The integral image, by eliminating the need to compute a multi-scale image pyramid, reduces the initial image processing required for object detection significantly. In the domain of face detection the advantage is quite dramatic. Using the integral image, face detection is completed *before* an image pyramid can be computed.

While the integral image should also have immediate use for other systems which have used Harr-like features (such as Papageorgiou et al. [10]), it can foreseeably have impact on any task where Harr-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 technique for feature selection based on AdaBoost. An aggressive and effective technique for feature selection will 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 reduce 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., propose a more complex and heterogeneous mechanism for filtering [8]. Similarly Amit and Geman propose a hierarchical structure for detection in which the stages are quite different in structure and processing [1]. A homogenous 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.

7 Acknowledgements

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.

References

[1] Y. Amit, D. Geman, and K. Wilder. Joint induction of shape features and tree classifiers, 1997.

[2] Anonymous. Anonymous. In *Anonymous*, 2000.

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

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

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

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

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

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

[9] Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

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

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

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

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

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

[15] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

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

[17] Patrice Y. Simard, Lon Bottou, Patrick Haffner, and Yann Le Cun. Boxlets: a fast convolution algorithm for signal processing and neural networks. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 571–577, 1999.

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

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

[20] Andrew Webb. *Statistical Pattern Recognition*. Oxford University Press, New York, 1999.

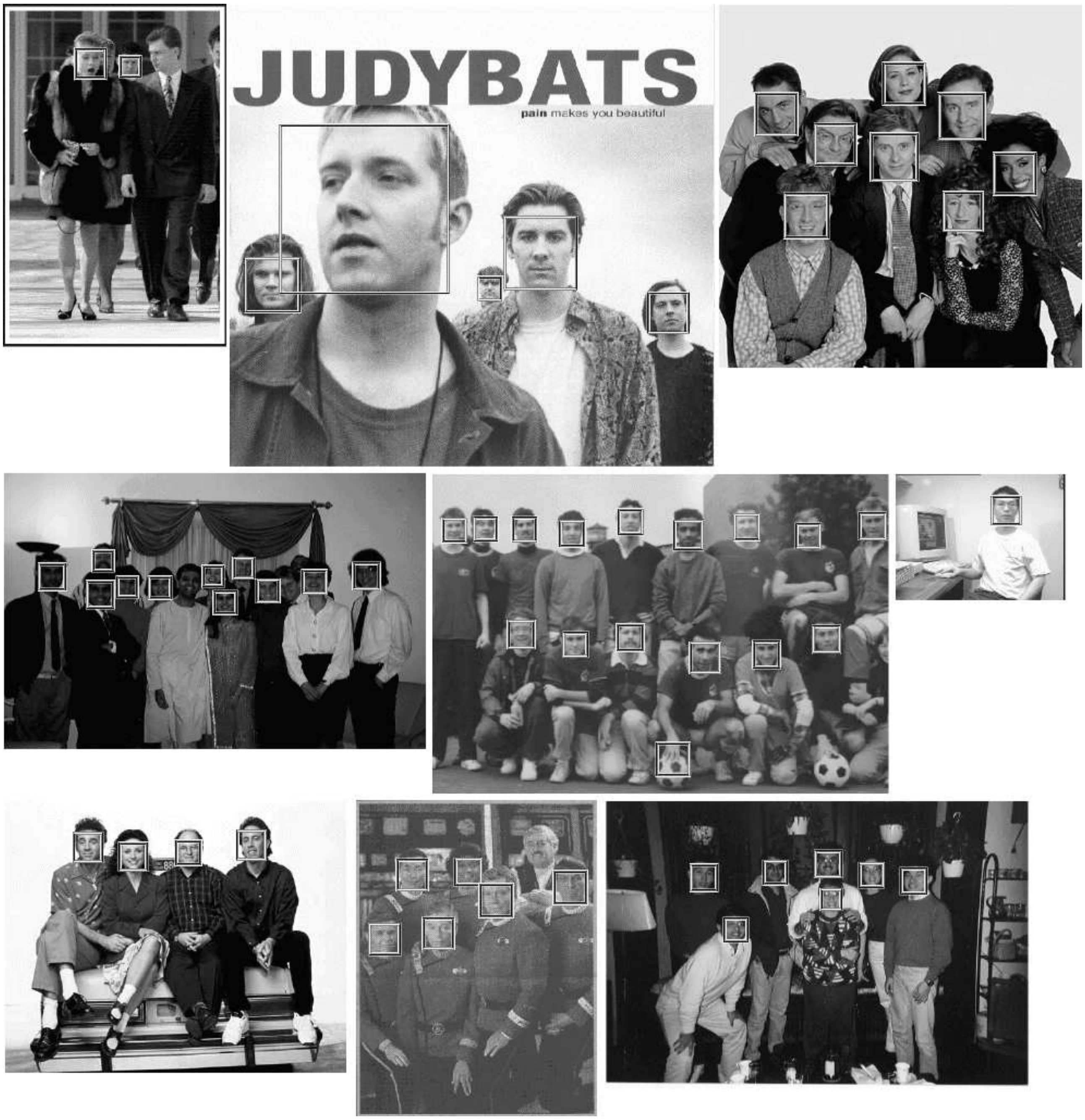

Figure 10: Output of our face detector on a number of test images from the MIT+CMU test set.