2007

# Rapid Face Detection using Independent Component Analysis

Aditya Rajgarhia '07
*Illinois Wesleyan University*

# Rapid Face Detection using Independent Component Analysis

by

**Aditya Rajgarhia**

Submitted to the Department of Mathematics and
Computer Science
in partial fulfillment of the requirements for Research
Honors

at

Illinois Wesleyan University

April 2007

# Abstract

Face detection is the task of determining the locations and sizes of human faces in arbitrary digital images, while ignoring any other objects to the greatest possible extent. A fundamental problem in computer vision, it has important applications in fields ranging from surveillance-based security to autonomous vehicle navigation. Although face detection has been studied for almost a decade, the results are not satisfactory for a variety of practical applications, and the topic continues to receive attention.

A commonly used approach for detecting faces is based on the techniques of "boosting" and "cascading", which allow for real-time face detection. However, systems based on boosted cascades have been shown to suffer from low detection rates in the later stages of the cascade. Yet, such face detectors are preferable to other methods due to their extreme computational efficiency.

In this thesis we introduce a novel variation of the boosting process that uses features extracted by Independent Component Analysis (ICA), which is a statistical technique that reveals the hidden factors that underlie sets of random variables or signals. The information describing a face may be contained in both linear as well as high-order dependencies among the image pixels. These high-order dependencies can be captured effectively by representation in ICA space. Moreover, it has been argued that the metric induced by ICA is superior to other methods in the sense that it may provide a representation that is more robust to the effect of noise such as variations in lightening. We propose that features extracted from such a representation may be boosted better in the later stages of the cascade, thus leading to improved detection rates while maintaining comparable speed. We present the results of our face detector, as well as comparisons with existing systems.

# Contents

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Face detection can be regarded as a specific case of object detection. In object detection, the task is to find the locations and sizes of all objects in an image that belong to a given class, regardless of the orientation, scale, and lighting conditions. Examples of classes to which object detection has been successfully employed include faces, eyes, pedestrians and cars. Early face detection algorithms focused on the detection of frontal human faces, whereas newer algorithms attempt to solve the more general and difficult problem of multi-view face detection. That is, the detection of faces that are either rotated along the axis from the face to the observer (in-plane rotation), or rotated along the vertical or left-right axis (out-of-plane rotation), or both. Moreover, current object detection systems concentrate on real-time detection, which have broader practical applications.

Many algorithms implement the face-detection task as a binary pattern-classification task. That is, the content of a given part of an image is transformed into features, after which a classifier trained on example faces decides whether that particular region of the image is a face, or not. Often, a window-sliding or "pyramid" technique is employed. That is, the above-mentioned classifier is used to classify smaller portions of in image, at all locations and scales, as either faces or non-faces.

A given natural image typically contains many more background patterns than face patterns. In fact, the number of background patterns may be 1,000 to 100,000 times larger than the number of face patterns. This means that if one desires a high face detection rate, combined with a low number of false detections in an image, one needs a very specific classifier. Publications in the field often use the rough guideline that a classifier should yield a 90% detection rate, combined with a false-positive rate in the order of $10^{-6}$.

Figure 1.1 illustrates the face detection technology. The red bounding squares mark the positions that were labeled as faces by the detector. However, we can see that some non-face images have also been labeled as faces. Such instances constitute the false positives. We can also identify one frontal face near the top-right corner that was not classified correctly; this would be an instance of a false negative.

The problem of face detection is closely related to, but distinct from, the task of face recognition. A face recognition system is a computer-driven application for automatically identifying a person from a digital image. It does that by comparing selected facial features in the given image and an existing face database. Usually face recognition systems first apply a face detector to locate the faces in an image, and then apply a separate recognition algorithm to identify the face.

This objective of this thesis is to improve the accuracy rate of an existing real-time face detection system, without significantly affecting the speed of the detector. We shall use a statistical technique called Independent Component Analysis to find useful features that define human faces. Our work currently concentrates on frontal faces, but can be extended to incorporate rotated views in the future. Preliminary results have been highly encouraging, with accuracy rates of over 95% for standard testing sets.

Figure 1.1: Output of a face detector on a given image. The red bounding squares indicated the detected faces.

## 1.1 Applications of Face Detection

Face detection is used in biometrics, often as a part of (or along with) a facial recognition system. It is also used in video surveillance, human computer interface and image database management. Direct applications of face detection are listed in table 1.1, while applications of face recognition are listed in table 1.2.

Table 1.1: Some applications of face detection.

| **Area** | **Specific Applications** |
|---|---|
| Surveillance | Secure ATM terminals |
| Entertainment | Auto-focus in a camera, Web-cam focus |
| Mobile | Video phone |
| Robotics | Autonomous vehicles, Industrial robots |
| Scientific applications | Atomic particle tracking, Medical imaging |

Table 1.2: Some applications of face recognition.

| Area | Specific Applications |
|---|---|
| Biometrics | Drivers' Licenses, Passports |
| Information Security | Desktop login, Internet security, Secure terminals |
| Surveillance | CCTV Control, Post-event analysis, Suspect tracking |
| Access Control | Vehicular access, Facility access |
| Entertainment | Film annotation |

Several applications such user interfaces, image databases, teleconferencing, film annotation, and video phone would require an extremely fast (possibly real-time) face detection system. Hence, the speed is a very important consideration is modern face detectors. Presently, numerous commercial face detection and recognitions systems are available.

## 1.2 Organization of Thesis

The remainder of the dissertation is organized as follows.

Chapter 2 will presents an existing real-time face detection system as well an an extension of the original system. These detection systems lead to the design of our own detector. In the beginning of the chapter, however, we first review some machine learning concepts that will be used extensively in this thesis.

Chapter 3 examines the statistical technique called Independent Component Analysis (ICA). We use this technique to model face images, and eventually to discriminate between face and non-face images. The mathematical foundations of ICA are briefly explained before describing a practical algorithm for performing ICA. A similar technique called Principal Component Analysis is also outlined.

Chapter 4 describes the major contribution of our face detection system. In particular, we explain how facial features extracted using ICA are used for the task of learning a face classifier. Two different approaches to the task are presented.

Chapter 5 contains the actual system tests, along with comparisons of the accuracy to existing algorithms when they have been applied to the same test sets. This section also describes the implementation details.

Chapter 6 summarizes the contributions of the thesis and points out directions for future work.

# Chapter 2

# Previous Work

Our work builds upon the detection system proposed in [Viola and Jones, 2001] and later extended in [Lienhart and Maydt, 2002]. This section will describe their respective algorithms. First, however, we explicate some relevant terminology and concepts.

## 2.1 Basic Concepts

### 2.1.1 Combining multiple learners

In any application, we can use one of several learning algorithms. Each learning algorithm dictates a certain model that comes with a set of assumptions. The performance of a learner may be fine-tuned to get the highest possible accuracy on a validation set, but this fine-tuning is a complex task and still there are instances on which even the best learner is not accurate enough. The idea is that there may be another learner that is accurate on these. By suitably combining multiple learners then, accuracy can be improved.

Since there is no point in combining learners that always make similar decisions, the aim is to be able to find a set of *base learners* who differ in their decisions so that they will complement each other.

### 2.1.2 Boosting

In boosting, we actively try to generate complementary base learners by training the next learner on the mistakes of the previous learners. The original boosting algorithm [Schapire, 1990] combines three weak learners to generate a strong learner. A weak learner has error probability less than $1/2$, which

makes it better than random guessing, and a strong learner has arbitrarily small error probability. Though it is quite successful, the disadvantage of boosting is that it requires a very large training sample.

[Freund and Schapire, 1996] proposed a variant, names *AdaBoost*, short for adaptive boosting, that uses the same training set over and over and thus need not be large. AdaBoost can also combine an arbitrarily large number of base learners. The following is a discussion of the original algorithm *AdaBoost.M1* (see Alg. 1). The idea is to modify the probabilities of drawing the instances as a function of the error. Let us say $p_j^t$ denotes the probability that the instance pair $(x^t, y^t)$ is drawn to train the $j$th base learner $d_j$. Initially, all $p_1^t = 1/N$. Then we add new base learners as follows, starting from $j = 1$: $\epsilon_j$ denotes the error rate of $d_j$. AdaBoost requires that $\epsilon_j < 1/2$, $\forall j$; if not, we stop adding new base learners. We define $\beta_j = \epsilon_j/(1-\varepsilon_j) < 1$, and we set $p_{j+1}^t = \beta_j p_j^t$ if $d_j$ correctly classifies $x^t$, otherwise $p_{j+1}^t = p_j^t$. Because $p_{j+1}^t$ is a probability, there is a normalization where we divide $p_{j+1}^t$ by $\sum_t p_{j+1}^t$, so that they sum up to 1. This has the effect that the probability of a correctly classified instance is decreased (with the amplitude of decrease proportional to the confidence of the previous learner, $p_1^t$), and the probability of a misclassified instance increases. Then a new sample of the same size is drawn from the original sample according to these modified probabilities, $p_{j+1}^t$, and is used to train $d_{j+1}$. This has the effect that $d_{j+1}$focuses more on instances misclassified by $d_j$. That is why the base learners are chosen to be simple and not very accurate, since otherwise the next training sample would contain only a few outlier and noisy instances repeated many times over. Now, once training is done, AdaBoost is a *voting* method. i.e. Given an instance, all $d_j$ decide and a weighted vote is taken where weights are proportional to the base learners' accuracies (on the training set): $w_j = log(1/\beta_j)$.

### 2.1.3 Cascading

The idea in cascaded classifiers is to have a sequence of base classifiers $d_j$ sorted in terms of their space or time complexity, or the cost of representation that they use, so that $d_{j+1}$ is costlier than $d_j$. Cascading is a multistage method and we use $d_j$ only if all the preceding learners, $d_k$, $k < j$ are not confident. Associated with each learner is a *confidence* $w_j$ such that we say $d_j$ is confident of its output and can be used if $w_j > \theta_j$ where $1/K < \theta_j < \theta_{j+1} < 1$ is the confidence threshold. In classification, the confidence function is set to the highest posterior, $w_j \equiv max_i d_{ji}$. We use learner $d_j$

**Algorithm 1** AdaBoost Algorithm

**Training:**

For all $\{x^t, r^t\}_{t=1}^N \in \chi$, initialize $p_1^t = 1/N$
For all base learners $d_j$, $j = 1, ..., L$
  Randomly draw $\chi_j$ from $\chi$ with probabilities $p_j^t$
  Train $d_j$ using $\chi_j$
  For each $(x^t, r^t) \in \chi_j$, calculate $y_j^t \leftarrow d_j(x^t)$
  Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t . 1(y_j^t \neq r^t)$
  If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop
  $\beta_j = \frac{\epsilon_j}{(1-\epsilon_j)}$
  For each $(x^t, r^t)$, decrease probabilities if correct:
    If $y_j^t = r^t$ then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
  Normalize probabilities:
    $Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

**Testing:**

Given $x$, calculate $d_j(x)$, $j = 1, ..., L$
Calculate class outputs, $i = 1, ..., K$:

$$y_i = \sum_{j=1}^{L} (log \frac{1}{\beta_j}) d_{ji}(x)$$

if all the previous learners are not confident:

$y_i = d_{ji}$ if $w_j > \theta_j$ and $\forall k < j$, $w_k < \theta_k$

Starting with $j = 1$, given a training set, we train $d_j$. Then we find all instances from a separate validation set on which $d_j$ is not confident, and these constitute the training set for $d_{j+1}$. Note that unlike in AdaBoost, we choose not only the misclassified instances but also the ones for which the previous base learner is not confident. The idea is that an early simple classifier handles the majority of instances, and a more complex classifier is only used for a small percentage, thereby not significantly increasing the overall complexity. Cascading thus stands between the two extremes of parametric and nonparametric classification. The former, a linear model, finds a single rule that should cover all the instances. A nonparametric classifier, on the other hand, stores the whole set of instances without generating any simple rule explaining them. Cascading generates a rule (or rules) to explain a large part of the instances as cheaply as possible and stores the rest as exceptions.

## 2.2 Robust Real-Time Face Detection

Viola and Jones described a face detection framework that is capable of processing images extremely rapidly while achieving high detection rates. There are three key contributions of this detection

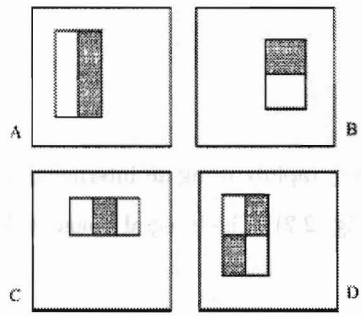


Figure 2.1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels that lie within the white rectangles is subtracted from the sum of pixels in the grey rectangles.Two-rectangle feature are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

framework. The first is the introduction of a new image representation called the "Integral Image" which allows the features used by the detector to be computed very quickly. The second is a simple and efficient classifier which is built using the AdaBoost learning algorithm 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.

### 2.2.1 Features

The detection procedure classifies images based on the value of simple features, as opposed to using the image pixels directly. The most common reason for doing so 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 task is to find suitable features for detecting objects in images.

The simple features used are reminiscent of those derived from Haar basis functions in [Papageorgiou et al., 1998]. More specifically, Viola and Jones use three kinds of features. The value of a *two-rectangle feature* is the difference between the sum of the pixels within the two rectangular regions. 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. Note that unlike the Haar basis, the set of rectangle features is over-complete[2.1].

[2.1] 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.

### 2.2.2 Integral Image

Rectangle features can be computed very rapidly using an intermediate representation for the image that is called the integral image (see Fig. 2.2). The integral image at location $x, y$ contains the sum of the pixels above and to the left of $x, y$ inclusive:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{2.2.1}$$

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

$$s(x,y) = s(x,y-1) + i(x,y) \tag{2.2.2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{2.2.3}$$

(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 calculated in four array references (see Fig. 2.3). Clearly the difference between two rectangular sums can be calculated in eight references. Since the two-rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, and eight and nine references in the cases of three and four-rectangle features respectively.

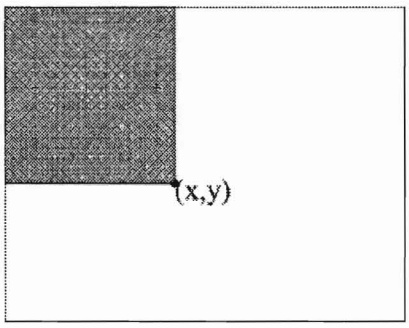


Figure 2.2: The value of the integral image at point (x,y) is the sum of all the pixels above and to the left.

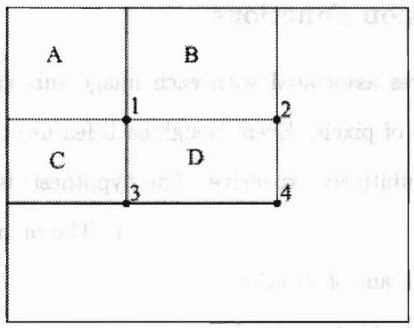


Figure 2.3: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A + B, at location 3 is A + C, and at location 4 is A + B + C + D. The sum within D can be computed as 4 + 1 - (2 + 3).

### 2.2.3 Feature Discussion

Rectangle features are somewhat primitive when compared with alternative such as steerable filters. Viola and Jones generated a very large and varied set of rectangle features. Typically the representation is about 400 times over-complete. This over-complete set provides features of arbitrary aspect ration and 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 the conventional approach which 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. In contrast, the meaningful set of rectangle features have the property that a single feature can be evaluated at any scale and location in a few operations. Moreover, 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 pyramid alone. Any procedure which requires a pyramid of this type will necessarily run slower than the integral image-based detector.

### 2.2.4 Learning Classification Functions

There are 160,000 rectangle features associated with each image sub-window of 24 x 24 pixels, a number far larger than the number of pixels. Even though each feature can be computed efficiently, computing the complete set is prohibitively expensive. The hypothesis is that a very small number of these features can be combined to form an effective classifier. The main challenge, then, is to find these features. In this system, a variant of AdaBoost is used to select the features and to train the classifier. The formal guarantees provided by the AdaBoost learning procedure are quite strong. It has been 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.

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. 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. For each feature, the weak learner describes 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 24 x 24 pixel sub-window of the image ($x$), a threshold ($\theta$) and a polarity ($p$) indicating the direction of the inequality:

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

The weak classifiers used (thresholded single features) can thus be viewed as single node decision trees, and the final strong classifier takes the form of a perceptron (a weighted combination of weak classifiers followed by a threshold).

**Algorithm 2** The boosting algorithm used by Viola and Jones. $T$ hypotheses are constructed 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.

**Training:**

Given example images $(x_1, y_1)$, ..., $(x_n, y_n)$ 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 $t = 1, ..., T$ :
Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
Select the best weak classifier with respect to the weighted error:
$\epsilon_t = min_{f,p,\theta} \sum_i w_{t,i} |h(x_i, f, p, \theta) - y_i|$.
Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where $f_t$, $p_t$ and $\theta_t$ are the minimizers of $\epsilon_t$. Then, $h_t(x)$ is the best weak classifier.
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 $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

**Testing:**

The final strong classifier is:

$$C(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = log \frac{1}{\beta_t}$

### 2.2.5 The Attentional Cascade

A cascade of classifiers is used, which achieves increased detection performance while radically reducing computation time. 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.

The overall form of the detection process is that of a degenerate decision tree, or cascade. 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 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. Hence, while a positive instance will trigger the evaluation of every classifier in the cascade, this is an exceedingly rare event.

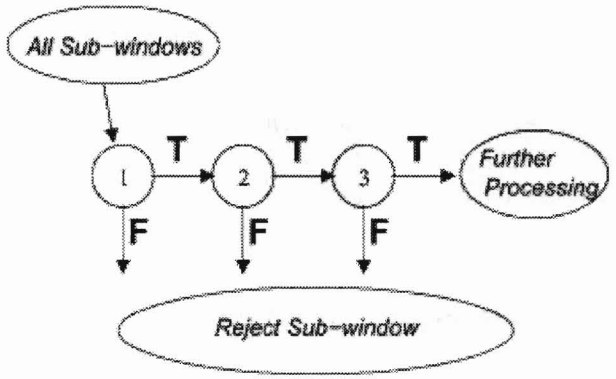


Figure 2.4: Schematic depiction of the attention cascade, A series of classifiers are applied to each sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After 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, or an alternative detection system.

### 2.2.6 Training a Cascade of Classifiers

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

$$F = \prod_{i=1}^{K} f_i, \tag{2.2.4}$$

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 $i$th classifier on the examples that get through to it. The detection rate is

$$D = \prod_{i=1}^{K} d_i, \tag{2.2.5}$$

where $D$ is that detection rate of the cascaded classifier, $K$ is the number of classifiers, and $d_i$ is the detection rate of the $i$th 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 this detection rate may sound a daunting task, it is made significantly easier by the fact that each stage need only achieve a false positive rate of about 30% (since $0.30^{10} \approx 6 \text{ x } 10^{-6}$).

The number of features evaluated when scanning real images is necessarily a probabilistic process. 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:

$$N = n_0 + \sum_{i=1}^{K} \left( n_i \prod_{j<i} p_j \right) \tag{2.2.6}$$

where $N$ is the expected number of features evaluated, $K$ is the number of classifiers, $p_i$ is the positive rate of the $i$th classifier, and $n_i$ are the number of features in the $i$th 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 algorithm described in section 3.3 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.

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 maximum acceptable rate for $f_i$ and the minimum acceptable rate for $d_i$. Each layer of the cascade is trained by AdaBoost 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 on a set of images which do not contain any instances of faces. This algorithm is given more precisely in the table below.

**Algorithm 3** The training algorithm for building a cascaded detector.

- User selects values for $f$, the maximum acceptable false positive rate per layer, and $d$, the maximum 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 $F_i > f.F_{i-1}$
    $n_i \leftarrow n_i + 1$
    * 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 $i$th classifier until the current cascaded classifier has a detection rate of at least $d.D_{i-1}$ (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 $N$.

### 2.2.7 Results of Viola and Jones

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 training set.

#### 2.2.7.1 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 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 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.

#### 2.2.7.2 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% faces. The next classifier has ten features and rejects 80% of non-faces while correctly detecting almost 100% of faces. The next two layers are 25-feature classifiers followed by three 50-feature classifiers with a variety of different numbers of features chosen according to Algorithm 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 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 the 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.

#### 2.2.7.3 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. Recall that $\sigma^2 = m^2 - \frac{1}{N}\sum x^2$, where $\sigma$ is the standard deviation, $m$ is the mean, and $x$ is the pixel value within the sub-window. The variance of an image sub-window can thus be computed quickly using a pair of integral images, since we already know how to compute the sum of pixels in a rectangle sub-window efficiently.

#### 2.2.7.4 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

| | 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%) | – | – | – |

Figure 2.5: Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

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 $\triangle$. This shifting process is affected by the scale of the detector: if the current scale is $s$ the window is shifted by $[\triangle s]$, where [] is the rounding operation.

#### 2.2.7.5 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. In practice, it makes sense to return one final detection per face. Toward this end it is useful to post-process the detected sub-windows in order to combine overlapping detections into a single detection.

The 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.

#### 2.2.7.6 Failure Modes

By observing the performance of the face detector on a number of test images, Viola and Jones noticed a few different failure modes. The face detector was trained on frontal, upright faces. Informal observations suggest that the face detector can detect faces that are tilted up to about $\pm 15$ degrees in plane and about $\pm 45$ degrees out of plane (toward a profile view). The detector becomes unreliable with more rotation than this. They also noticed that harsh backlighting in which the faces are very

dark while the background is relatively light sometimes causes failures. Finally, the detector fails on significantly occluded faces, particularly if the eyes are occluded.

## 2.3 An Extended Set of Haar-like Features for Rapid Object Detection

This paper by Lienhart and Maydt introduces a novel set of rotated Haar-like features, which significantly enrich the basic set of simple Haar-like features used in the Viola and Jones detector (hereafter referred to as *VJD*), and which can also be calculated very efficiently. At a given hit rate their face detector (*LMD*) shows an average a 10% lower false alarm rate by means of using these additional rotated features. They also present a novel post optimization procedure for a given boosted cascade, improving on average the false alarm rate further by 12.5%.

### 2.3.1 Feature Pool

Similar to this features used by *VJD*, The features used can be computed at any position and any scale in the same constant time. Only 8 table lookups are needed. The features mimic Haar-like features and early features of the visual pathway such as center surround and directional responses.

### 2.3.2 Feature Family

Let us assume that the basic unit for testing for the presence of an object is a window of $WxH$ pixels. A rectangle is specified by the tuple $r = (x, y, w, h, \alpha)$ with $0 \leq x, x + w \leq W$, $0 \leq y, y + h \leq H$, $x, y \geq 0$, $w, h > 0$, and $\alpha \in \{0^o, 45^o\}$ and its pixel sum is denoted by $RecSum(r)$. The raw feature set is then the set of all possible features of the form

$$feature_I = \sum_{i \in I = \{1,\ldots,N\}} \omega_i . RecSum(r_i), \tag{2.3.1}$$

where the weights $\omega_i \in \Re$, the rectangles $r_i$, and $N$ are arbitrarily chosen.

This raw feature set is almost infinitely large. For practical reasons, it is reduces as follows:

1. Only weighted combinations of pixel sums of two rectangles are considered (i.e., $N = 2$).

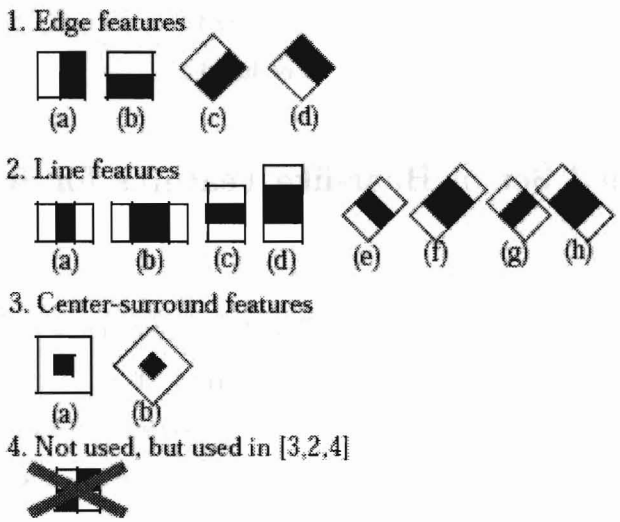


Figure 2.6: Feature prototypes of simple Haar-like features and center-surround features. Black areas have negative and white areas positive weights.

2. The weights have opposite signs, and are used to compensate for the difference in area size between those two rectangles. Thus, for non-overlapping rectangles we have $-\omega_0.Area(r_0) = \omega_1.Area(r_1)$. Without restrictions we can set $w_0 = 1$ and get $\omega_1 = Area(r_0)/Area(r_1)$.

These restrictions lead us to the 14 feature prototypes shown in Figure 5:

- Four edge features,
- Eight line features, and
- Two center-surround features

These prototypes are scaled independently in vertical and horizontal direction in order to generate a rich, over complete set of features.

#### 2.3.2.1 Fast Feature Computation

For upright rectangles the auxiliary image is the *Summed Area Table* $SAT(x, y)$, similar to the *Integral Image* used in VJD. $SAT(x, y)$ is defined as the sum of the pixels of the upright rectangle ranging from the top left corner at $(0, 0)$ to the bottom right corner at $(x, y)$:

$$SAT(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y'). \tag{2.3.2}$$

It can be easily calculated with one pass over all pixels from left to right and top to bottom by means of

$$SAT(x,y) = SAT(x,y-1) + SAT(x-1,y) + I(x,y) - SAT(x-1,y-1) \tag{2.3.3}$$

with $SAT(-1,y) = 0$ and $SAT(x,-1) = 0$. From this, the pixel sum of any upright rectangle $r = (x,y,w,h,0)$ can be determined by four table lookups:

$$RecSum(r) = SAT(x-1,y-1)+SAT(x+w-1,y+h-1)-SAT(x-1,y+h-1)-SAT(x+w-1,y-1) \tag{2.3.4}$$

For $45^o$ rotated rectangles the auxiliary image is defined as the *Rotated Summed Area Table* $RSAT(x,y)$. It gives the sum of the pixels of the rectangle rotated by $45^o$ with the right most corner at $(x,y)$ and extending till the boundaries of the image:

$$RSAT(x,y) = \sum_{x' \le x, x' \le x-|y-y'|} I(x',y') \tag{2.3.5}$$

It can be calculated with two passes over all pixels. The first pass from left to right and top to bottom determines

$$RSAT(x,y) = RSAT(x-1,y-1) + RSAT(x-1,y) + I(x,y) - RSAT(x-2,y-1) \tag{2.3.6}$$

with

$$RSAT(-1,y) = RSAT(-2,y) = RSAT(x,-1) = 0, \tag{2.3.7}$$

whereas the second pass from the right to left and bottom to top calculates

$$RSAT(x,y) = RSAT(x,y) + RSAT(x-1,y+1) - RSAT(x-2,y) \tag{2.3.8}$$

From this the pixel sum of any rotated rectangle $r = (x,y,w,h,45^o)$ can be determined by four table lookups:

$$RecSum(r) = RSAT(x+w,y+w)+RSAT(x-h,y+h)-RSAT(x,y)-RSAT(x+w-h,w+w+h) \tag{2.3.9}$$

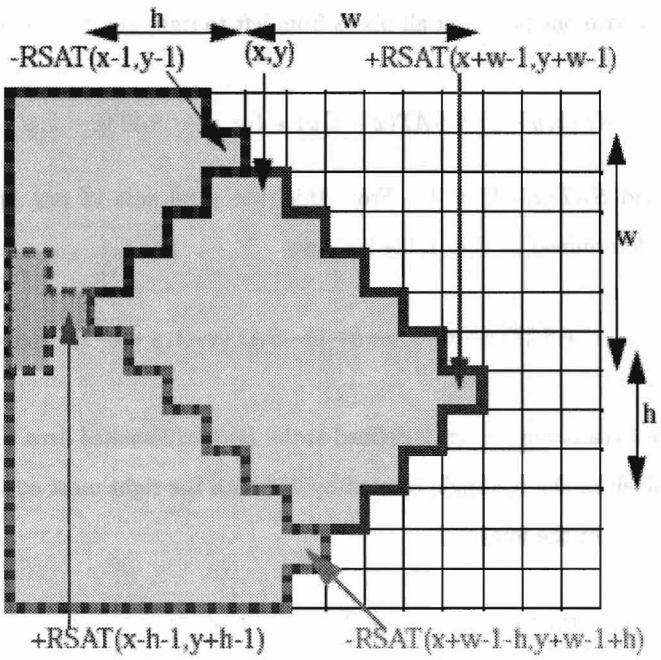


Figure 2.7: Calculation scheme for rotated areas.

### 2.3.3 Cascade of Classifiers

Similar to *VJD*, a cascade of classifiers is a degenerate decision tree where at each stage a classifier is trained to detect almost all objects of interest while rejecting a certain fraction of the non-object patterns. Lienhart and Maydt trained each stage to eliminate 50% (i.e. the false positive rate) of the non-face patterns while falsely eliminating only 0.2% (i.e. the detection rate) of the frontal face patterns; 13 stages were trained. Hence, in the optimal case, we can expect a false alarm rate of about $0.5^{13} \approx 1.2e-04$ and a hit rate of about $0.998^{13} \approx 0.97$. Each stage was trained using the Discrete AdaBoost algorithm.

### 2.3.4 Experimental Results

As compared to the original *VJD*, it was shown that the overall performance could be improved by about 23.8%, of which 10% could be attributed to the extended rotated features, and 12.5% to the stage post-optimization.

# Chapter 3

# Independent Component Analysis

A common problem encountered in fields such as statistics, data analysis, and signal processing is finding a suitable representation of multivariate data. For reasons of computation and conceptual simplicity, the representation is often sought as a linear transformation of the original data. In other words, each component of the representation is a linear combination of the original variables. Well-known linear transformation methods include principal component analysis, factor analysis, and projection pursuit. Independent Component Analysis (ICA) is a recently developed linear transformation, in which the desired representation is one that maximizes the statistical independence of the components of the representation. Such a representation appears to capture the essential structure of the data in many applications, such as feature extraction and signal separation. neural activity research, telecommunications, brain images, stock prices, and face recognition.

ICA is of interest to a wide variety of scientists and engineers because it seems to reveal the driving forces which underlie a set of observed phenomena. In each application, a large set of signals is measured, and it is known that each signal depends on several distinct underlying factors, which provide the driving forces behind the changes in the measured signals. In other words, ICA is essentially a method for extracting useful information from data.

As a motivating example, consider the "cocktail party problem." Here, $n$ speakers are speaking simultaneously at a party, and any microphone placed in the room records only an overlapping combination of the $n$ speakers' voices. Say there are $n$ different microphones placed in the room. Each microphone is at a different distance from each of the speakers, hence each microphone records a different combination of the speakers' voices. The task, then, is to separate out the original $n$ speakers'

speech signals.

## 3.1 How ICA Works

ICA is based on the simple, generic, and physically realistic assumption that if different signals are from different physical sources, then those signals are statistically independent. ICA takes advantage of the fact that the implications of this assumption can be reversed, leading to a new assumption which is logically unwarranted but one that works well in practice, i.e., if statistically independent signals can be extracted from signal mixtures, then these extracted signals must be from different sources. Accordingly, ICA separates signal mixtures into statistically independent signals.

To formalize the problem, assume that we observe $n$ linear mixtures $x_1, ..., x_n$ of $n$ independent components

$$x_j = a_{j1}s_1 + a_{j2}s_2 + ... + a_{jn}s_n \tag{3.1.1}$$

for all $j$. In the ICA model defined above, we assume that each mixture $x_j$ as well as each independent component $s_k$ is a random variable. The observed values $x_j$, e.g., the microphone signals in the cocktail party problem, are then a ample of this random variable

It is convenient to use vector-matrix notation instead of the sums like in the previous equation. Let us denote by $\mathbf{x}$ the random vector whose elements are the mixtures $x_1, ..., x_n$, and likewise by $\mathbf{s}$ the random vector with elements $s_1, ..., s_n$. Let us denote by $\mathbf{A}$ the *mixing matrix* with elements $a_{ij}$. Using this vector-matrix notation, the above mixing model is written as

$$\mathbf{x} = \mathbf{A}\mathbf{s}. \tag{3.1.2}$$

Sometimes we need the columns of matrix $\mathbf{A}$; denoting them by $\mathbf{a_j}$, the model can also be written as

$$\mathbf{x} = \sum_{i=1}^{n} \mathbf{a_i} s_i. \tag{3.1.3}$$

The ICA model is a generative model, which means that it describes how the observed data are generated by a process of mixing the components $s_i$. The independent components are latent variables, meaning that they cannot be directly observed. Also, the mixing matrix is assumed to be unknown. All we observe is the random vector $\mathbf{x}$, and we must estimate both $\mathbf{A}$ and $\mathbf{s}$ using it. After estimating

$\mathbf{A}$, we can compute its inverse, say $\mathbf{W}$(*unmixing matrix*), and obtain the independent components as

$$\mathbf{s} = \mathbf{W}\mathbf{x}. \tag{3.1.4}$$

ICA is very closely related to the method called *blind source separation* (BSS). A "source" here means an original signal, i.e. independent component, like the speaker in a cocktail party problem. "Blind" means that we know very little, if anything, about the mixing matrix, and make few assumptions about the source signals. ICA is one method for performing blind source separation.

## 3.2 Assumptions and Ambiguities

In the ICA model described above, the following ambiguities hold:

1. We cannot determine the exact source amplitudes of the independent components, but only their amplitudes relative to each other. The reason is that both $\mathbf{s}$ and $\mathbf{A}$ being unknown, any scalar multiplier in one of the sources $s_i$ could always be cancelled by dividing the corresponding column $\mathbf{a_i}$ of $\mathbf{A}$ by the same scalar. This ambiguity is, fortunately, insignificant in most applications.

2. We cannot determine the order of the independent components. The reason is that, again, both $\mathbf{s}$ and $\mathbf{A}$ being unknown, we can freely change the order of the terms in the sum in (3.1.1). Formally, a permutation matrix $\mathbf{P}$ and its inverse can be substituted in the model to give $\mathbf{x} = \mathbf{A}\mathbf{P}^{-1}\mathbf{P}\mathbf{S}$. The elements of $\mathbf{P}\mathbf{s}$ are the original independent variables $s_j$, but in another order. The matrix $\mathbf{A}\mathbf{P}^{-1}$ is just a new unknown mixing matrix.

The following assumptions also need to hold:

1. There must be at least as many different signal mixtures as there are source signals. If there are more source signal mixtures than signal mixtures, then BSS methods do not perform well. However, in practice this issue seldom arises.

2. The independent components must be non-Gaussian. To see the difficulty with Gaussian data, consider an example in which $n = 2$ and $s \sim N(0, \mathbf{I})$ where $\mathbf{I}$ is the $2x2$ identity matrix. Note that the contours of the density of the standard normal distribution $N(0, \mathbf{I})$ are circles centered on the origin, and the density is rotationally symmetric. Now suppose we observe some $\mathbf{x} = \mathbf{A}\mathbf{s}$. The distribution of $\mathbf{x}$ will also be Gaussian, with zero mean and covariance $E[\mathbf{x}\mathbf{x}^{\mathbf{T}}] =$

$E[\mathbf{A}\mathbf{s}\mathbf{s}^T\mathbf{A}^T] = \mathbf{A}\mathbf{A}^T$. Now, let $\mathbf{R}$ be an arbitrary orthogonal matrix, so that $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$, and let $\mathbf{A}' = \mathbf{A}\mathbf{R}$. Then, if the data had been mixed using $\mathbf{A}'$ instead of $\mathbf{A}$, we would have observed $\mathbf{x}' = \mathbf{A}'\mathbf{s}$. The distribution of $\mathbf{x}'$ is also Gaussian, with zero mean and covariance $E[\mathbf{x}'(\mathbf{x}')^T] = E[\mathbf{A}'\mathbf{s}\mathbf{s}^T(\mathbf{A}')^T] = E[\mathbf{A}\mathbf{R}\mathbf{s}\mathbf{s}^T(\mathbf{A}\mathbf{R})^T] = \mathbf{A}\mathbf{R}\mathbf{R}^T\mathbf{A}^T = \mathbf{A}\mathbf{A}^T$. Hence, whether the mixing matrix is $\mathbf{A}$ or $\mathbf{A}'$, we would observe the data from a $N(0, \mathbf{A}\mathbf{A}^T)$ distribution[3.1]. Thus, there is no way to tell if the sources were mixed using $\mathbf{A}$ or $\mathbf{A}'$. So, there is an arbitrary rotational component in the mixing matrix that cannot be determined from the data, and we cannot recover the original sources.

## 3.3 Principles of ICA Estimation

Intuitively speaking, the key to estimating the ICA model is nongaussianity. The Central Limit Theorem, a classical result in probability theory, says that the distribution of a sum of independent random variables tends to be a Gaussian distribution, under certain conditions. Thus, a sum of two independent random variables usually has a distribution that is closer to Gaussian than either of the two original random variables.

Now, to estimate one of the independent components, we consider a linear combination of the $x_i$; let us denote this by $y = \mathbf{w}^T\mathbf{x} = \sum_i w_i x_i$, where $\mathbf{w}$ is a vector to be determined. If $\mathbf{w}$ were one of the rows of the inverse of $\mathbf{A}$, this linear combination would actually equal one of the independent components (see eq. 3.1.2). The question is: How can we use the Central Limit Theorem to determine $\mathbf{w}$ so that it would equal one of the rows of the inverse of $\mathbf{A}$? In practice, we cannot determine such a $\mathbf{w}$ exactly, because we have no knowledge of matrix $\mathbf{A}$, but we can find an estimator that gives a good approximation.

Since a sum of independent random variables is more Gaussian than the original variables, choosing as $\mathbf{w}$ a vector that maximizes the nongaussianity of $\mathbf{w}^T\mathbf{x}$ gives us one of the independent components. In fact, the optimization landscape for nongaussianity in the $n$-dimensional space of vectors $\mathbf{w}$ has $2n$ local maxima, two for each independent component, corresponding to $s_i$ and $-s_i$ (recall that the independent components can be estimated only up to a multiplicative sign). Thus, to find several independent components, we need to find all these local maxima. As we shall see, this is not difficult, since the different independent components are uncorrelated.

[3.1] The property that has been used is that if $\mathbf{s}$ is Gaussian, then $\mathbf{s}\mathbf{s}^T = \mathbf{I}$

## 3.4 Measures of Nongaussianity

To use nongaussianity in ICA estimation, we must have a quantitative measure of the nongaussianity of a random variable, say $y$. To simplify things, let us assume that $y$ is centered (zero mean) and has variance equal to one.[3.1] Below, we briefly explicate the most common measures of nonguassianity as described in [Hyvarinen and Oja, 2000], before describing the FastICA algorithm.

### 3.4.1 Kurtosis

The classical measure of nongaussianity is kurtosis, which is defined as following

$$kurt(y) = E[y^4] - 3(E[y^2])^2. \tag{3.4.1}$$

However, since we assumed that $y$ is of unit variance, the right hand side simplifies to $E[y^4] - 3$. For a Guassian random variable, the kurtosis is zero, while for most non-Gaussian random variables, it is non-zero.

Random variables that have a negative kurtosis are called subgaussian, and those with positive kurtosis are called supergaussian. Typically, supergaussian random variables have a "spiky" pdf with heavy tails (e.g. the Laplace distribution). Subgaussian random variables, on the other hand, have a "flat" pdf (e.g. the Uniform distribution).

The main reason why kurtosis, or rather its absolute value, has been used widely as a measure of nongaussianity in ICA is due to its computational and theoretical simplicity. However, kurtosis also has some drawbacks in practice, when its value has to be estimated from a measured sample. The main problem is that kurtosis can be very sensitive to outliers. In other words, kurtosis is not a robust measure of nongaussianity.

### 3.4.2 Negentropy

A second very important measure of nongaussianity is given by negentropy. Negentropy is based on the information theoretic quantity of differential entropy. The entropy of a random variable can be interpreted as the degree of information that the observation of the variable gives. The more "random," i.e. unpredictable and unstructured the variable is, the larger is its entropy.

[3.1] In fact, this is a pre-processing step for ICA, as we will see.

Differential entropy is the entropy for continuous-valued random variables, and it is defined for a random vector $\mathbf{y}$with density $f(\mathbf{y})$ as

$$H(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) d\mathbf{y}. \tag{3.4.2}$$

A fundamental result of information theory is that a *gaussian variable has the largest entropy among all random variables of equal variance.* This means that entropy could be used as a measure of nongaussianity.

Now, to obtain a measure of nongaussianity that is zero for a gaussian variable and is always nonnegative, one often uses a slightly modified version of the definition of differential entropy, called negentropy. Negentropy $J$ is defined as follows

$$J(\mathbf{y}) = H(\mathbf{y}_{gauss}) - H(\mathbf{y}) \tag{3.4.3}$$

where $\mathbf{y}_{gauss}$ is a Gaussian random variable of the same covariance matrix as that of $\mathbf{y}$.

The advantage of using negentropy as a measure of nongaussianity is that it is well justified by statistical theory. In fact, negentropy is in some sense the optimal estimator of nongaussianity, as far as statistical properties are concerned. The problem in using negentropy is, however, that it is computationally very difficult. Estimating negentropy using the definition would require an estimate of the pdf. Therefore, simpler approximations of negentropy are very useful, as is discussed below.

The classical method of approximating negentropy is using higher-order moments, for example as

$$J(y) \approx \frac{1}{12} E[y^3]^2 + \frac{1}{48} kurt(y)^2. \tag{3.4.4}$$

The random variably $y$ is assumed to be of zero mean and unit variance (i.e., standardized). However, this approximation suffers from the non-robustness encountered with kurtosis. To avoid this problem, new approximations were developed in [Hyvarinen, 1998], based on the maximum entropy principle In general, we obtain the following approximation:

$$J(y) \approx \sum_{i=1}^{p} k_i [E[G_i(y)] - E[G_i(v)]]^2, \tag{3.4.5}$$

where $k_i$ are some positive constants, and $v$ is a standardized Gaussian variable. The variable $y$

is assumed to be standardized as well, and the functions $G_i$ are some non-quadratic functions as described in [Hyvarinen, 1998].

In the case where we use only one non-quadratic function $G$, the approximation becomes

$$J(y) \propto [E[G(y)] - E[G(v)]]^2 \tag{3.4.6}$$

for practically any non-quadratic function $G$. This is just a generalization of the moment-based approximation in (3.4.1), if $y$ is symmetric. Indeed, taking $G(y) = y^4$, one obtains exactly (3.4.1). But the point here is that by choosing $G$ wisely, one obtains approximations of negentropy that are much better than the one given by (3.4.1). In particular, choosing a $G$ that does not grow too fast, one obtains more robust estimators. The following choices of $G$ have proved to be very useful:

$$G_1(u) = \frac{1}{a_1} log\, cosh(a_1 u),\ G_2(u) = -exp(-u^2/2) \tag{3.4.7}$$

where $1 \leq a_1 \leq 2$ is some suitable constant.

Thus, we obtain approximations of negentropy that give a very good compromise between the properties of the two classical nongaussianity measures given by kurtosis and negentropy. They are conceptually simple, fast to computer, yet have appealing statistical properties, especially robustness. A practical algorithm based on these contrast functions will be presented in Section (**number**).

### 3.4.3 Minimization of Mutual Information

Another approach for ICA estimation, inspired by information theory, is minimization of mutual information. In particular, this approach gives a rigorous justification for the heuristics principles used above.

Using the concept of differential entropy, we define the mutual information $I$ between $m$ random variables, $y_i = 1, ..., m$ (for an invertible linear transformation $\mathbf{y} = \mathbf{W}\mathbf{x}$) as follows

$$I(y_1, y_2, ..., y_m) = \sum_{i=1}^{m} H(y_i) - H(\mathbf{y}). \tag{3.4.8}$$

Mutual information is a natural measure of the dependence between random variables. It is always non-negative, and zero if and only if the variables are statistically independent. If we constrain the $y_i$

to be *uncorrelated* and of unit variance, then we can obtain

$$I(y_1, , y_2, ..., y_n) = C - \sum_i J(y_i), \tag{3.4.9}$$

where $C$ is a constant. This shows the fundamental relation between negentropy and mutual information.

Using this approach, we define the ICA of a random vector $\mathbf{x}$ as an invertible transformation as in (3.1.2), where the matrix $\mathbf{W}$ is determined so that the mutual information of the transformed components $s_i$ is minimized. It is obvious from (3.4.2) that finding such an invertible transformation that minimizes the mutual information is roughly equivalent to finding directions in which *negentropy is maximized.* Rigorously speaking, (3.4.2) shows that ICA estimation by minimization of mutual information is equivalent to maximizing the sum of the nongaussianities of the estimates, when the *estimates are constrained to be uncorrelated.* This constraint is, in fact, not necessary, but simplifies the computations considerably.

### 3.4.4 Maximum Likelihood Estimation (MLE)

A very popular approach for estimating the ICA model is MLE, which is essentially equivalent to minimization of mutual information. MLE is a statistical method used to make inferences about the parameters of the underlying probability distribution from a given data set. It is essentially equivalent to minimization of mutual information. If we consider $\mathbf{W} = (\mathbf{w_1}, ..., \mathbf{w_n})^T$ to be the parameters of the random variable $\mathbf{x}$ (denoting the signal mixture), then the log-likelihood takes the form:

$$L(\mathbf{W}) = \sum_{t=1}^{T} \sum_{i=1}^{n} log(f_i(\mathbf{w}_i^T \mathbf{x}(t))) + T.log|det\mathbf{W}| \tag{3.4.10}$$

where the $f_i$ are the density functions of the $s_i$ (here assumed to be known). To see the connection between likelihood and mutual information, consider the expectation of the log-likelihood:

$$\frac{1}{T} E[L] = \sum_{i=1}^{n} E[log(f_i(\mathbf{w}_i^T \mathbf{x}(t)))] + log|det\mathbf{W}|. \tag{3.4.11}$$

Actually, if the $f_i$ were equal to the actual distributions of $\mathbf{w}_i^T\mathbf{x}$, the first term would be equal to $-\sum_i H(\mathbf{w}_i^T\mathbf{x})$. Thus, the likelihood would be equal, up to an additive constant, to the negative of mutual information.

### 3.4.5 The Infomax Principle

Another related contrast function was derived from a neural network viewpoint. This was based on maximizing the output entropy of a neural network with non-linear inputs. Assume that $\mathbf{x}$ is the linear input to the neural network whose outputs are of the form $\Phi_i(\mathbf{w}^T\mathbf{x})$, where the $\Phi_i$ are some non-linear scalar functions, and the $\mathbf{w}_i$ are the weight vectors of the neurons. One then wants to maximize the entropy of the outputs:

$$L_2 = H(\Phi_1(\mathbf{w}^T\mathbf{x}), ..., \Phi_n(\mathbf{w}^T\mathbf{x})). \tag{3.4.12}$$

If the $\Phi_i$ are well chosen, this framework also enables the estimation of the ICA model. In fact, it has been proved that the principle of network entropy maximization, or "infomax," is equivalent to maximum likelihood estimation.

## 3.5 Temporal and Spatial ICA

Typically in ICA, each of $M$ temporal signal mixtures is measured over $N$ time steps, and $M$ temporal source signals are recovered as $\mathbf{y} = \mathbf{W}\mathbf{x}$, where each source signal is independent *over time* of every other source signal. However, when considering temporal sequences of images, each image consists of a set of pixels, and each row of the data array $\mathbf{x}$ is the temporal sequence of one pixel over time. Thus, each column of $\mathbf{x}$ is an image recorded at one point in time. When we treat the rows of $\mathbf{x}$ as mixtures, we use ICA to find a set of independent temporal signals. On the other hand, if we treat the columns of $\mathbf{x}$ as mixtures, then the set of signals found by ICA are spatially independent images.

Thus, ICA can be used in one of two complementary ways to extract either temporal source signals from the rows of $\mathbf{x}$ using temporal ICA (tICA), or spatial source signals from the rows of $\mathbf{x}^T$ using spatial ICA (sICA).

[Bartlett and Movellan, 2002] defined temporal and spatial ICA as follows, and we shall use the same notation. Architecture I treats the images as random variables and the pixels as outcomes, while Architecture II treats the pixels as random variables and the images as outcomes. Let $\mathbf{x}$ be a data matrix with $n_r$ rows and $n_c$ columns. We can think of each column of $\mathbf{x}$ as outcomes (independent trials) of a random experiment. We think of the $i$th row of $\mathbf{x}$ as the specific value taken by a random variable $\mathbf{x}_i$ across $n_c$independent trials. this defines an empirical probability distribution for $\mathbf{x}_1, ..., \mathbf{x}_{n_r}$,

in which each column of $\mathbf{x}$ is given probability mass $1/n_c$. Independence is then defined with respect to such a distribution. For example, we say that rows $i$ and $j$ of $\mathbf{x}$ are independent if it is not possible to predict the values taken by $\mathbf{x}_j$ across columns from the corresponding values taken by $\mathbf{x}_i$.

Our goal is to find a good set of basis images to represent a database of faces. We organize each image in the database as a long vector with as many dimensions as number of pixels in the image. There are at least two ways in which ICA can be applied to this problem.

1. We can organize our database into a matrix $\mathbf{x}$ where each row vector is a different image (see Fig. 3.1, left). In this approach, images are random variables and pixels are trials so that it makes sense to talk about independence of images or functions of images. Two images $i$ and $j$ are independent if, when moving across pixels, it is not possible to predict the value taken by the pixel in image $j$ based on the value taken by the same pixel in image $i$.

2. We can transpose $\mathbf{x}$ and organize our data so that images are in the columns of $\mathbf{x}$ (see Fig. 3.1, right). In this approach, pixels are random variables and images are trials. Here, it makes sense to talk about independence of pixels or functions of pixels. For example, pixel $i$ and $j$ would be independent if, when moving across the entire set of images, it is not possible to predict the value taken by pixel $i$ based on the corresponding value taken by pixel $j$ on the same image.

## 3.6 Comparison to other strategies

A closely related technique to ICA is Principal Component Analysis (PCA). However, PCA in itself has widely been used for similar purposes as ICA, particularly for applications requiring feature extraction. PCA is also reducing multidimensional data sets to lower dimensions for analysis, which is why it is used as a pre-processing step for ICA, as mentioned earlier.

Technically speaking, PCA is an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA can be used for dimensionality reduction in a data set while retaining those characteristics of the data set that contribute most to its variance, by keeping lower-order principal components and ignoring higher-order ones. Such low-order components often contain

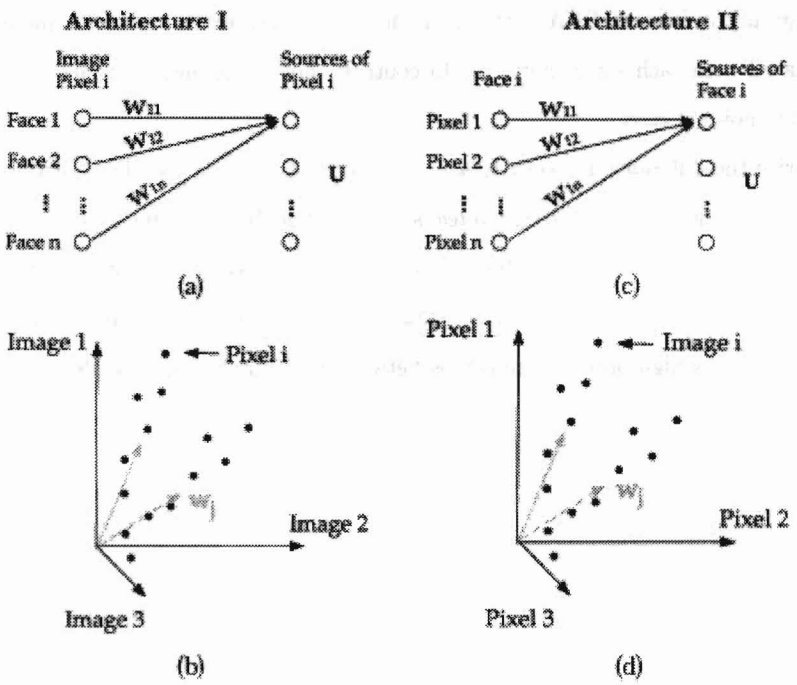


Figure 3.1: Two architectures for performing ICA on images. (a) Architecture I for finding statistically independent basis images. Performing source separation on the face images produces IC images in the rows of $\mathbf{u}$. (b) The gray values at pixel location $i$ are plotted for each face image. ICA in architecture I finds weight vectors in the directions of statistical dependencies among the pixel locations. (c) Architecture II for finding a factorial code. Performing source separation on the pixels produced a factorial code in the columns of the output matrix $\mathbf{u}$. (d) Each face image is plotted according to the gray values taken on at each pixel location. ICA in architecture II finds weight vectors in the directions of statistical dependencies among the face images.

the "most important" aspects of the data. But this is not necessarily the case, depending on the application. In fact, our hypothesis is that ICA is better at extracting facial features as compared to PCA. In Chap. 5, we also present results from a face detector based on PCA.

Another related method is Factor Analysis (FA), which is a data reduction technique used to explain variability among observed random variables in terms of fewer unobserved random variables called factors. The observed variables are modeled as linear combinations of the factors, plus "error" terms. FA is essentially a form of PCA with the addition of these extra terms for modeling the sensor noise associated with each signal mixture. In contrast, both ICA and PCA are based on the assumption that such noise is zero.

We can summarize the difference between ICA and PCA/FA by saying that PCA/FA decompose a set of signal mixtures into a set of *uncorrelated* signals, while ICA decomposes a set of signal mixtures into a set of *independent* signals. This difference is critical because the signals extracted by PCA/FA are under-constrained relative to those extracted by ICA, since by definition of statistical independence, ICA captures high-order relationships between the source signals while PCA/FA only capture the linear relationships.

# Chapter 4

# Boosting in ICA Feature Space

In Sec. 2.2.4, we described the boosting process in Haar-like feature space. The classification power of the described system is limited when the weak classifiers derived from simple local features become too weak to be boosted, especially in the later stages of the cascade training. Empirically, it has been observed in **[Zhang, Li and Gatica-Perez, 2004]** that when the discriminating power of a strong classifier reaches a certain point, e.g. a detection rate of 90% and a false alarm rate of $10^{-6}$, non-face examples become very similar to the face examples in terms of the Haar-like features. The histograms of the face and non-face examples for any feature can barely be differentiated, and the empirical probability of misclassification of the weak classifiers approaches 50%. At this stage, boosting becomes ineffective because the weak learners are too weak to be boosted. This issue has been discussed in the past in [Valiant, 1984]. One way to address this problem is to use better weaker classifiers in a different feature space, which is more powerful. We propose to boost in ICA coefficient space. As we show, weak classifiers in this global feature space have sufficient classification power for boosting to be effective in the later stages of the cascade.

First, we shall explicate the two architectures for performing ICA, as mentioned in Sec. 3.5.

## 4.1 Architecture 1: Statistically Independent Basis Images

The goal here is to find a set of statistically independent basis images. We organize the face mixtures in matrix $\mathbf{x}$ so that the images are in rows and the pixels are in columns. In this approach, ICA finds a matrix W such that the rows of $\mathbf{u} = \mathbf{W}\mathbf{x}$ are as statistically independent as possible. The

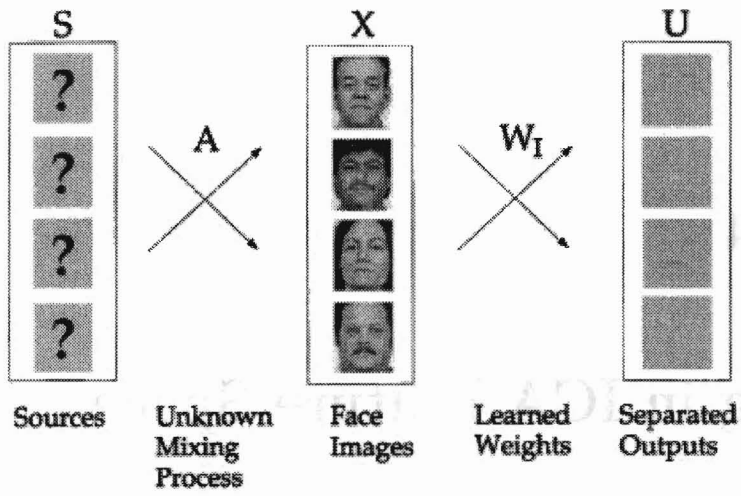


Figure 4.1: Image synthesis model for Architecture I. To find a set of IC images, the images in $\mathbf{x}$ are considered to be a linear combination of statistically independent basis images, $\mathbf{s}$, where $\mathbf{A}$ is an unknown mixing matrix. The basis images are estimates as the learned ICA output $\mathbf{u}$.

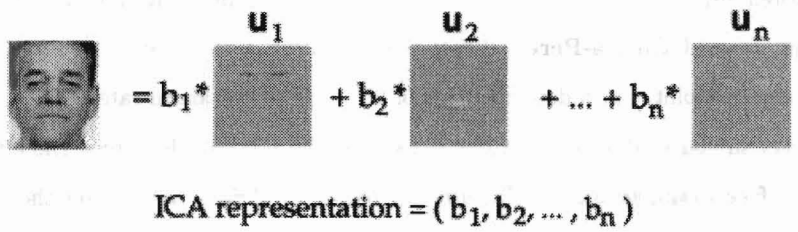


Figure 4.2: The independent basis image representation consists of the coefficients, $\mathbf{b}$, for the linear combination of independent basis images, $\mathbf{u}$, that comprised each face image $\mathbf{x}$.

source images estimated by the rows of $\mathbf{u}$ are then used as basis images to represent faces. Face image representation consists of the coordinates of these images with respect to the image basis defined by the rows of $\mathbf{u}$, as shown in Fig. 4.2. These coordinates are contained in the mixing matrix $\mathbf{A} = \mathbf{W}^{-1}$.

The number of IC's found by the FastICA algorithm corresponds to the dimensionality of the input. In order to have control over the number of ICs extracted, instead of performing ICA directly on the $n_r$ original images, we perform ICA on a set of $m$ linear combinations of those images, where $m < n_r$. Recall that the ICA model assumes that the images in $\mathbf{x}$ are a linear combination of a set of unknown statistically independent sources. Thus, the ICA model is unaffected by replacing the original images with a linear combination of those images.

We choose for these linear combinations the first $m$ PC eigenvectors of the images set. PCA (see

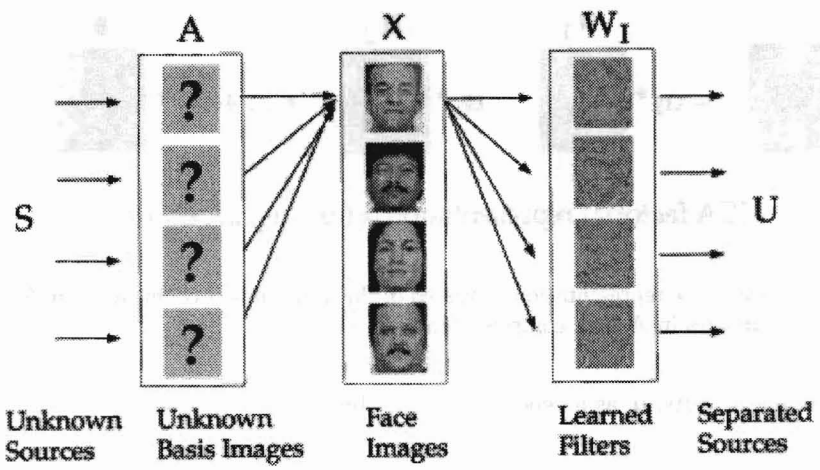


Figure 4.3: Image synthesis model for Architecture II. Each image in the dataset was considered to be a linear combination of underlying basis images in the matrix **A**. The basis images were each associated with a set of independent causes, given by a vector of coefficients in **s**. The basis images were estimated by $\mathbf{A} = \mathbf{W}^{-1}$, where **W** is the learned ICA weight matrix.

Sec. 3.6) on the images set in which the pixels locations are treated as observations and each face images a measure, provides the linear combination of the parameters (images) that accounts for the maximum variability in the observations (pixels). Moreover, the PCA vectors in the input did not discard the higher-order relationships. These relationships still exist in the data, but are simply not separated.

## 4.2 Architecture II: A Factorial Face Code

The goal in Architecture 1 was to find a set of spatially independent basis images. Now, although the basis images obtained in that architecture are approximately independent, the coefficients that code each feature are not necessarily independent. Architecture II uses ICA to find a representation in which the coefficients used to code images are statistically independent, i.e., a factorial face code. [Barlow, 1992] and [Atick, 1992] have discussed the advantages of factorial codes for encoding complex objects that are characterized by high-order combinations of features.

We organize the data matrix **x** such that rows represent different pixels and columns represent different images. This corresponds to treating the columns of $\mathbf{A} = \mathbf{W}^{-1}$ as a set of basis images (see Fig. 4.3). The ICA representations are then in the columns of $\mathbf{u} = \mathbf{W}\mathbf{x}$. Each column of **u** contains the coefficients of the basis images in **A** for reconstructing each image in **x** (see Fig. 4.4). ICA

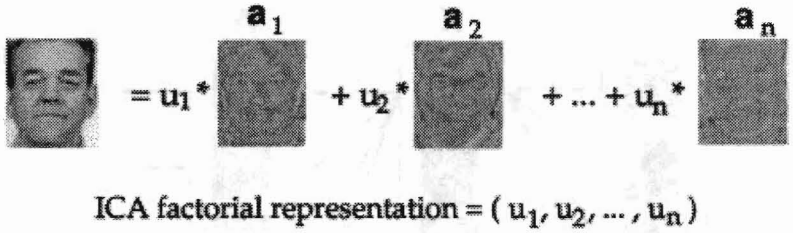


Figure 4.4: The factorial code representation consisted of the independent coefficients, $\mathbf{u}$, for the linear combination of basis images in $\mathbf{A}$ that comprised each face image $\mathbf{x}$.

attempts to make the outputs, $\mathbf{u}$, as independent as possible.

## 4.3 Boosting ICA Features

In AdaBoost learning, each weak classifier is constructed based on the histogram of a single feature derived from ICA coefficients $(b_1, b_2, ..., b_m)$. At each round of boosting, one ICA coefficient, the one which is most effective for discriminating between face and non-face classes, is selected by AdaBoost.

As stated earlier, the distributions of the two classes in the Haar-like feature space almost completely overlap in the later stages of the cascade training. In that case, we propose to switch feature spaces and construct weak features in the ICA space. We do need to address the question of which stage in the cascade we should switch from the Haar-like features to the ICA features. It is quite evident that ICA features are much more computationally expensive than Haar-like features. Now, if we used ICA features in early stages of boosting, we would have to extract ICA features from a very large number of sub-windows, and the speed of the face detection system would be too slow for real-time performance. On the other hand, if we used ICA features in very late stages of boosting, the performance improvement gained from their superiority would be limited. Therefore, we shall determine the switching stage based on the trade off between speed and performance improvement.

# Chapter 5

# Experimental Results

This section describes the final face detection system. First, we provide the implementation details for our system. The discussion includes details on the structure and training of the detector, as well as results on large real-world testing sets. We also consider the importance the size and quality of the training data set towards creating an accurate classifier, and present results for two training sets of different sizes.

## 5.1 Implementation

The objective of this thesis was to describe a novel face detection system which is based on an existing state-of-the-art face detector. Many experiments were accomplished in this work. Due to limitations in processing power, only the most important parts of the detector were able to be tested. In particular, we were unable to train a cascade of classifiers. However, an AdaBoost classifier based on ICA features was implemented, and the results are presented in the following sections.

We chose to use C/C++ for implementing our system due to performance considerations as well as the availability of external libraries in these languages. All coding, training, and testing was done using *Gentoo Linux* with the 2.6 version of the kernel.

Initially, we explored the possibility of modifying the *Intel OpenCV* library by adding ICA features, since it is a standard library in computer vision, and the code was written in the C language to be highly efficient. Unfortunately, lack of proper documentation in the source code did not allow us to achieve the required modifications in the available time frame. Nonetheless, OpenCV proved to be

extremely useful for understanding the original Viola Jones detection system as well as the Lienhart and Maydt extension, since these detectors are implemented fully in the library. The source code for OpenCV is available free of charge at `http://sourceforge.net/projects/opencvlibrary/`.

The *MultiBoost* C++ library was hence used for training the AdaBoost classifier. We chose this library due to its extremely flexible design, which enabled us to easily extend it to ICA features. Moreover, Haar features are an integral part of the library. The source code for MultiBoost can be obtained for free at `http://sourceforge.net/projects/multiboost`.

For performing FastICA as well as other mathematical functions, we used *IT++*, which is an C++ library composed of classes and functions for linear algebra, signal processing and telecommunication systems. Templated vector and matrix classes are the core of the IT++ library, making its functionality similar to that of MATLAB and GNU Octave. IT++ also makes an extensive use of existing open source libraries (but not only) for increased functionality, speed and accuracy. In particular BLAS, CBLAS, LAPACK, ATLAS and FFTW libraries can be used. In our case, the *Intel MKL* library was used to provide these additional libraries to IT++.

The IT++ source code can be obtained freely at `http://itpp.sourceforge.net/`, while the Intel MKL library can be obtained (for non-commercial use on the Linux platform) from the official Intel website at `http://www.intel.com`.

All experiments were performed on a computer with a Pentium P4-2.8 Ghz processor and 1.5 GB of RAM.

## 5.2 Training Datasets

Typically, researchers compile their own training images from random crawls of the WWW, but such a task was unfeasible given our time and resources. The training data set we used is the the MIT-CBCL face database, which is publicly available at `http://cbcl.mit.edu/software-datasets/FaceData2.html`. Because our object and pattern detection approach is learning-based, how well the system eventually performs depends heavily on the quality of training examples it receives.The MIT-CBCL data set is not ideal for the purpose of training a classifier due a low resolution of 19 x 19 pixels. In fact, [Viola and Jones, 2001] report that an increased resolution of 24 x 24 pixels results in higher accuracy of the face detector. However, the data set will serve our purpose of comparing our detection system with other state-of-the-art systems, which we shall train using the same training set.

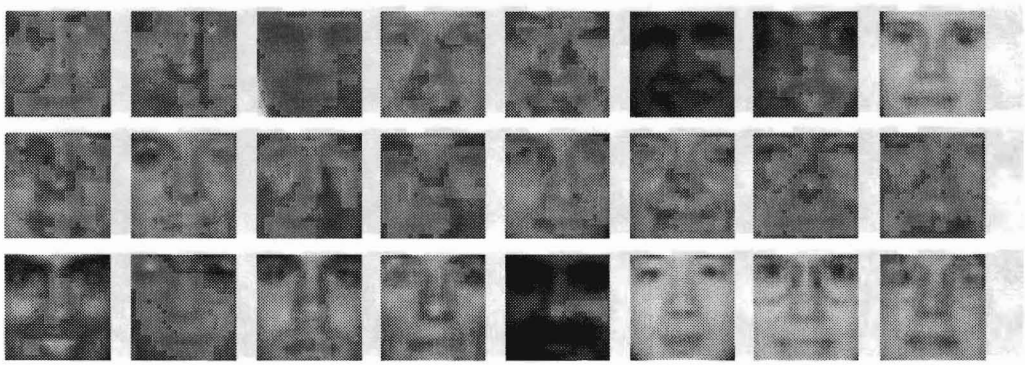

Figure 5.1: Example face images from the training set.

The original MIT-CBCL training set contains 2,429 face images and 4,548 non-face images in 19 x 19 grayscale PGM format images. The training faces are only roughly aligned, i.e., they were cropped manually around each face just above the eyebrows and about half-way between the mouth and the chin. The non-face images are random background images of the same size and format.

We also performed training with an extended version of the MIT-CBCL data set. The original images were randomly mirrored, rotated, translated and scaled by small amounts to obtain a set of 17,495 faces and 113,939 non-face images. Although the additional images are just variants of the original ones, the performance of the classifier is affected significantly, as shown subsequently. This leads us to believe that an even larger training set would be more beneficial.

## 5.3 Image Processing

All face and non-face images in the training set were histogram equalized to increase the local contrasts of the images. Often, usable data of the image is represented by close contrast values. Through histogram equalization, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast without affecting the global contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

Consider a discrete grayscale image, and let $n_i$ be the number of occurrences of gray level $i$. The probability $p$ of an occurrence of a pixel of level $i$ in the image is

$$p(x_i) = \frac{n_i}{n}, \; i \in 0, 1, ..., L-1 \tag{5.3.1}$$

Figure 5.2: Face images from Fig. 5.1 after histogram equalization.

where $L$ is the total number of gray levels in the image, 256 for the images. $n$ is the total number of pixels in the image, which is 361 for a 19 x 19 pixel image.

Let us also define $c$ as the cumulative distribution function corresponding to $p$, defined as

$$c(i) = \sum_{j=0}^{i} p(x_j), \tag{5.3.2}$$

also known as the image's accumulated normalized histogram.

We would like to create a transformation of the form $y = T(x)$ that will produce a level $y$ for each level $x$ in the original image, such that the cumulative probability function of $y$ will be linearized across the value range. The transformation is defined as

$$y_i = T(x_i) = c(i). \tag{5.3.3}$$

Notice that the transformation $T$ maps the levels into the domain of 0...1. In order to map the values back into their original domain, the following simple transformation needs to be applied on the result:

$$y_i^{'} = y_i(max_i - min_i) + min_i \tag{5.3.4}$$

where $max_i$ and $min_i$ are the maximum and minimum intensity values respectively in image $i$. Thus, an image which is transformed using its cumulative histogram yields an output histogram that is flat.

## 5.4 ICA and Haar Features

Two face detection systems were trained: One using Haar features (we call this *H-Boost*) and the other using Architecture I ICA features (we call this *I-Boost*). We trained both types of classifiers with several different numbers of features ranging from 50 features to 350 features. In the following sections, we shall present the results for H-Boost and I-Boost classifiers trained using 200 Haar-like and ICA features respectively, since the classifiers based on 200 features performed better than the classifiers based on smaller or larger number of features. Nonetheless, with the availability of greater processing power in the future, we would like to experiment more to find the optimal number of features.

For training the I-Boost system, we first extracted the ICA features from the 2,429 face images in the MIT-CBCL training set. Next, all the 2,429 face images and the 4,548 non-face images from the training set were projected onto the set of ICA features to obtain the ICA coefficients of these images. AdaBoost was performed on the coefficients of these 6,977 training images to produce the strong classifier. The extended training set of 17,495 faces and 113,939 non-faces was similarly projected onto the ICA basis extracted from the 2,249 face images to produce another strong classifier.

While experimenting with different numbers of faces from which we extract the ICA features, we found that larger numbers of faces result in better performance of the detector. However, extracting the independent components is a very memory-intensive task, and our memory limitations did not allow us to use more than 5,000 images. We chose to use the 2,429 images from the training set due to reasons of uniformity. In the future, we would like to use the 17,495 face images from the extended training set to extract the ICA features.

During testing, a given image is similarly projected on the above-mentioned ICA features to obtain the ICA coefficients for that image. The AdaBoost classifier then uses these coefficients to predict the class of the test image.

For training the H-Boost system, we first created the integral image representation for the training set, and then performed AdaBoost on the Haar-like features that are obtained using this integral image.

One point to note is that the AdaBoost learning procedure attempts only to minimize errors, and is not specifically designed to achieve high detection rates at the expense of large false positive rates. A 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

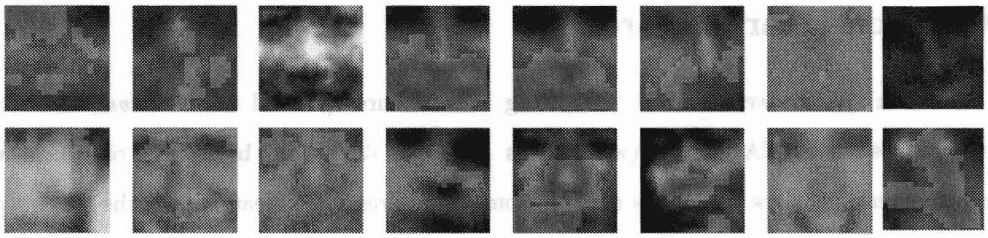

Figure 5.3: Example face images from the testing set.

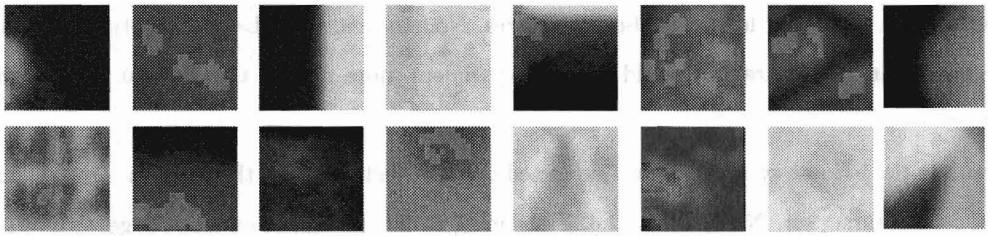

Figure 5.4: Example non-face images from the testing set.

lower detection rate. Lower thresholds yield classifiers with more false positives and a higher detection rate. We vary the threshold over a wide range in order to evaluate the detector, as presented in the following section.

## 5.5 Experiments on Real-World Test Sets

A number of experiments were performed to evaluate the system. We tested our system on the MIT-CBCL face test set, which consists of 472 faces and 23,573 non-faces. The testing images are of the same size as the training images, and are also cropped similarly. Considerable pose and lightning variations are represented by the test set, as can be seen in Fig. 5.4. The test face images are clearly more challenging to identify as compared to the training ones seen in Fig. 5.1, even for a human.

Fig. 5.5 shows the performance of our detection system (I-Boost) as well as that of a detector based on Haar-like features (H-Boost). Note that the H-Boost detector used is not the same as the Viola-Jones detector, since it is not cascaded. Clearly, the I-Boost detector performs better than H-Boost for all false positive rates. Moreover, using the extended training set significantly improves the accuracy.

| | *False Positive Rate (%)* | | | | |
|---|---|---|---|---|---|
| ***Detector*** | 1.00 | 2.00 | 3.00 | 5.00 | 10.00 |
| I-Boost (original training set) | 3.0 % | 16.7 % | 28.6 % | 38.6 % | 50.4 % |
| I-Boost (extended training set) | 12.7 % | 26.5 % | 38.6 % | 62.7 % | 75.0 % |
| H-Boost (original training set) | 2.0 % | 5.7 % | 9.1 % | 14.8 % | 29.2 % |

Figure 5.5: Detection rates for various numbers of false positives on the MIT-CBCL test set containing 472 faces and 23,573 non-faces.

# Chapter 6

# Conclusions

In this thesis we introduced a novel algorithm for detecting faces, based on features derived from Independent Component Analysis. Motivated by the fact that the weak learners based on the simple Haar-like features are too weak in the later stages of the cascade, we propose to boost ICA features in the later stages. The global ICA feature space complements the local Haar-like feature space. The algorithm selects the most effective features from ICA features using AdaBoost.

Various experiments were performed to show the advantage of using ICA features for face detection. The results can be stated as follows:

- ICA features are better at discriminating between face and non-face images as compared to Haar-like features.
- Increasing the size of the training set as well as the size of images for ICA feature extraction significantly improves the detection rate for a given false positive rate.

Although we have not yet implemented the cascaded detector, the results from the AdaBoost classifier show that our system achieves high accuracy on the MIT-CBCL test set. Most importantly, though, we have showed that ICA features are, in fact, better than Haar-like features at discriminating between faces and non-faces. Hence, we are very optimistic that a cascaded detections system which combines Haar-like and ICA features would demonstrate higher accuracy than a detector based only on Haar-like features. The computational efficiency of FastICA, coupled with the fact that the majority of images are rejected in the early stages of the cascade, should ensure that performance is not affected ostensibly.

Using machine learning to solve problems related to face detection is a relatively recent field of research. In our work we have investigated a very small aspect of it, and even the problems that we addressed warrant further research.

## 6.1 Future Work

A larger training set would be essential for the detector to be of practical use. In particular, the number of non-face images would have to be drastically increased in order to decrease false positives. Moreover, as mentioned earlier, using a larger number of face images to extract ICA features would also improve the accuracy.

Implementing the cascade is required in order to achieve the ultimate aim of our work, i.e., to improve the accuracy of the Viola-Jones detector while maintaining real-time detection speed. Training the cascade in feasible time, of course, would require significant processing power.

We would also like to compare our system with other state-of-the-art detection systems such as those based on Neural Networks and Support Vector Machines.

It was mentioned in Sec. 5.4 that we have used the Architecture I ICA features in the I-Boost classifier. Another task in the future would be to implement Architecture II features as described in Sec. 4.2 and to compare the results.

# Chapter 7

# Acknowledgements

In the last year that I have been working on my thesis, I have received support and encouragement from several people. Firstly, I thank Professor Weiyu Zhu for all his advice, as well as for the time that he has spent discussing this project with me. I would like to express my appreciation to all the Professors in the Mathematics and Computer Science department for guiding me throughout my college years.

I am very grateful to all the various people on the internet who have answered dozens of technical as well as conceptual questions. A special acknowledgement for the developers of the open-source libraries that I have used for this project. These people spent long hours out of their free time writing the enormous amounts of code that I have used, without which this work would not have been accomplished.

Finally, I would like to thank Professor Gabe Spalding for permitting me to be absent from the class during my final semester, in order to concentrate on my thesis! He has been very supportive of all my aims and has to some extent (unknowingly) inspired me to get involved in research.

# Bibliography


[Viola and Jones, 2001] P. Viola and M. Jones. Robust Real-time Object Detection. In *International Journal of Computer Vision*, pages 137-154, 2001.

[Lienhart and Maydt, 2002] R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. In *IEEE International Conference on Image Processing*, pages 900-903, Rochester, New York, 2002.

[Schapire, 1990] R. E. Schapire. The Strength of Weak Learnability. In *Machine Learning*, pages 197-227, 1990.

[Freund and Schapire, 1996] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148-156, 1996.

[Papageorgiou et al., 1998] C. P. Papageorgiou, M. Oren, T. Poggio. A General Framework for Object Detection. In *Proceedings of International Conference on Computer Vision*, Bombay, India, January 1998.

[Hyvarinen and Oja, 2000] A. Hyvarinen and E. Oja. Independent component analysis: Algorithms and applications. In *Neural Networks*, pages 411-430, 2000.

[Hyvarinen, 1998] A. Hyvarinen. New approximations of differential entropy for independent component analysis and projection pursuit. In *Advances in Neural Information Processing Systems*, volume 10, pages 273-279. MIT Press, 1998.

[Bartlett and Movellan, 2002] M.S. Bartlett, J.R. Movellan, T.J. Sejnowski. Face Recognition by Independent Component Analysis. In *IEEE Trans. on Neural Networks*, pages 1450-1464, November 2002.

[Zhang, Li and Gatica-Perez, 2004] D. Zhang, S. Z. Li, and D. Gatica-Perez. Real-Time Face Detection Using Boosting Learning in Hierarchical Feature Spaces. In *Proceedings of International Conference on Pattern Recognition*, Cambridge, August 2004.

[Valiant, 1984] L. Valiant. A Theory of the Learnable. In *Communications of ACM*, 1984.

[Atick, 1992] J.J. Atick. Could information theory provide an ecological theory of sensory processing? In *Network*, page 213-251, 1992.

[Barlow, 1992] H.B. Barlow. Unsupervised learning. In *Neural Computation*, page 295-311, 1989.