Published in Image Processing On Line on 2014–06–26.
Submitted on 2013–08–31, accepted on 2014–05–09.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2014.104

An Analysis of the Viola-Jones Face Detection Algorithm

Yi-Qing Wang

CMLA, ENS Cachan, France (yiqing.wang@cmla.ens-cachan.fr)

Abstract

In this article, we decipher the Viola-Jones algorithm, the first ever real-time face detection system. There are three ingredients working in concert to enable a fast and accurate detection: the integral image for feature computation, Adaboost for feature selection and an attentional cascade for efficient computational resource allocation. Here we propose a complete algorithmic description, a learning code and a learned face detector that can be applied to any color image. Since the Viola-Jones algorithm typically gives multiple detections, a post-processing step is also proposed to reduce detection redundancy using a robustness argument.

Source Code

The source code and the online demo are accessible at the IPOL web page of this article[1].

Keywords: face detection; Viola-Jones algorithm; integral image; Adaboost

1 Introduction

A face detector has to tell whether an image of arbitrary size contains a human face and if so, where it is. One natural framework for considering this problem is that of binary classification, in which a classifier is constructed to minimize the misclassification risk. Since no objective distribution can describe the actual prior probability for a given image to have a face, the algorithm must minimize both the false negative and false positive rates in order to achieve an acceptable performance.

This task requires an accurate numerical description of what sets human faces apart from other objects. It turns out that these characteristics can be extracted with a remarkable committee learning algorithm called Adaboost, which relies on a committee of weak classifiers to form a strong one through a voting mechanism. A classifier is weak if, in general, it cannot meet a predefined classification target in error terms.

An operational algorithm must also work with a reasonable computational budget. Techniques such as integral image and attentional cascade make the Viola-Jones algorithm [10] highly efficient: fed with a real time image sequence generated from a standard webcam, it performs well on a standard PC.

[1] http://dx.doi.org/10.5201/ipol.2014.104

2 Algorithm

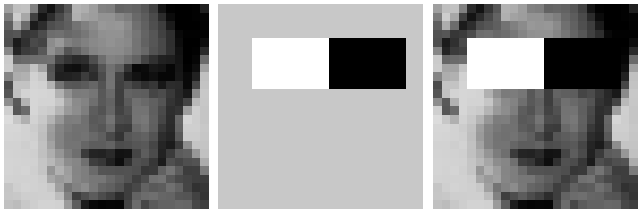
To study the algorithm in detail, we start with the image features for the classification task.

2.1 Features and Integral Image

The Viola-Jones algorithm uses Haar-like features, that is, a scalar product between the image and some Haar-like templates. More precisely, let I and P denote an image and a pattern, both of the same size $N \times N$ (see Figure 1). The feature associated with pattern P of image I is defined by

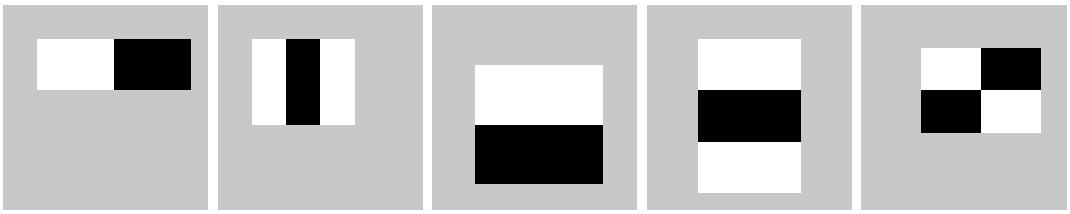
$$\sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j) \text{ is white}} - \sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j) \text{ is black}}.$$

To compensate the effect of different lighting conditions, all the images should be mean and variance normalized beforehand. Those images with variance lower than one, having little information of interest in the first place, are left out of consideration.



(a) (b) (c)

Figure 1: Haar-like features. Here as well as below, the background of a template like (b) is painted gray to highlight the pattern's support. Only those pixels marked in black or white are used when the corresponding feature is calculated.



(a) (b) (c) (d) (e)

Figure 2: Five Haar-like patterns. The size and position of a pattern's support can vary provided its black and white rectangles have the same dimension, border each other and keep their relative positions. Thanks to this constraint, the number of features one can draw from an image is somewhat manageable: a 24×24 image, for instance, has 43200, 27600, 43200, 27600 and 20736 features of category (a), (b), (c), (d) and (e) respectively, hence 162336 features in all.

In practice, five patterns are considered (see Figure 2 and Algorithm 1). The derived features are assumed to hold all the information needed to characterize a face. Since faces are by and large regular by nature, the use of Haar-like patterns seems justified. There is, however, another crucial element which lets this set of features take precedence: the integral image which allows to calculate them at a very low computational cost. Instead of summing up all the pixels inside a rectangular

Algorithm 1 Computing a 24×24 image's Haar-like feature vector

Input: a 24×24 image with zero mean and unit variance
Output: a $d \times 1$ scalar vector with its feature index $\mathfrak{f}$ ranging from 1 to d
Set the feature index $\mathfrak{f} \leftarrow 0$
Compute feature type (a)
for all (i, j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w, h) such that $i + h - 1 \le 24$ and $j + 2w - 1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
    compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
    record this feature parametrized by $(1, i, j, w, h)$: $S_1 - S_2$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (b)
for all (i, j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w, h) such that $i + h - 1 \le 24$ and $j + 3w - 1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
    compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
    compute the sum S_3 of the pixels in $[i, i + h - 1] \times [j + 2w, j + 3w - 1]$
    record this feature parametrized by $(2, i, j, w, h)$: $S_1 - S_2 + S_3$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (c)
for all (i, j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w, h) such that $i + 2h - 1 \le 24$ and $j + w - 1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
    compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
    record this feature parametrized by $(3, i, j, w, h)$: $S_1 - S_2$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (d)
for all (i, j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w, h) such that $i + 3h - 1 \le 24$ and $j + w - 1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
    compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
    compute the sum S_3 of the pixels in $[i + 2h, i + 3h - 1] \times [j, j + w - 1]$
    record this feature parametrized by $(4, i, j, w, h)$: $S_1 - S_2 + S_3$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (e)
for all (i, j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w, h) such that $i + 2h - 1 \le 24$ and $j + 2w - 1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
    compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
    compute the sum S_3 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
    compute the sum S_4 of the pixels in $[i + h, i + 2h - 1] \times [j + w, j + 2w - 1]$
    record this feature parametrized by $(5, i, j, w, h)$: $S_1 - S_2 - S_3 + S_4$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for

window, this technique mirrors the use of cumulative distribution functions. The integral image II of I

$$\mathrm{II}(i,j) := \begin{cases} \sum_{1\le s\le i}\sum_{1\le t\le j} I(s,t), & 1 \le i \le N \text{ and } 1 \le j \le N \\ 0, & \text{otherwise} \end{cases},$$

is so defined that

$$\sum_{N_1\le i\le N_2}\sum_{N_3\le j\le N_4} I(i,j) = \mathrm{II}(N_2,N_4) - \mathrm{II}(N_2,N_3-1) - \mathrm{II}(N_1-1,N_4) + \mathrm{II}(N_1-1,N_3-1), \quad (1)$$

holds for all $N_1 \le N_2$ and $N_3 \le N_4$. As a result, computing an image's rectangular local sum requires at most four elementary operations given its integral image. Moreover, obtaining the integral image itself can be done in linear time: setting $N_1 = N_2$ and $N_3 = N_4$ in (1), we find

$$I(N_1,N_3) = \mathrm{II}(N_1,N_3) - \mathrm{II}(N_1,N_3-1) - \mathrm{II}(N_1-1,N_3) + \mathrm{II}(N_1-1,N_3-1).$$

Hence a recursive relation which leads to Algorithm 2.

Algorithm 2 Integral Image

1: **Input:** an image I of size $N \times M$.
2: **Output:** its integral image II of the same size.
3: Set $\mathrm{II}(1,1) = I(1,1)$.
4: **for** $i = 1$ to N **do**
5: **for** $j = 1$ to M **do**
6: $\mathrm{II}(i,j) = I(i,j) + \mathrm{II}(i,j-1) + \mathrm{II}(i-1,j) - \mathrm{II}(i-1,j-1)$ and II is defined to be zero whenever its argument (i,j) ventures out of I's domain.
7: **end for**
8: **end for**

As a side note, let us mention that once the useful features have been selected by the boosting algorithm, one needs to scale them up accordingly when dealing with a bigger window (see Algorithm 3). Smaller windows, however, will not be looked at.

2.2 Feature Selection with Adaboost

How to make sense of these features is the focus of Adaboost [1].

Some terminology. A classifier maps an observation to a label valued in a finite set. For face detection, it assumes the form of $f : \mathbb{R}^d \mapsto \{-1, 1\}$, where 1 means that there is a face and -1 the contrary (see Figure 3) and d is the number of Haar-like features extracted from an image. Given the probabilistic weights $w_\cdot \in \mathbb{R}_+$ assigned to a training set made up of n observation-label pairs (x_i, y_i), Adaboost aims to iteratively drive down an upper bound of the empirical loss

$$\sum_{i=1}^{n} w_i 1_{y_i \ne f(x_i)},$$

under mild technical conditions (see Appendix A). Remarkably, the decision rule constructed by Adaboost remains reasonably simple so that it is not prone to overfitting, which means that the empirically learned rule often generalizes well. For more details on the method, we refer to [2, 3]. Despite its groundbreaking success, it ought to be said that Adaboost does not learn what a face

Algorithm 3 Feature Scaling

Input: an $e \times e$ image with zero mean and unit variance ($e \geq 24$)
Parameter: a Haar-like feature type and its parameter (i, j, w, h) as defined in Algorithm 1
Output: the feature value
if *feature type* (a) **then**
  set the original feature support size $a \leftarrow 2wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, h \leftarrow \mathrm{J}he/24\mathrm{K}$ where $\mathrm{J}z\mathrm{K}$ defines the nearest integer to $z \in \mathbb{R}_+$
  $w \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 2we/24\mathrm{K}/2,\ 2\kappa \leq e - j + 1\}$
  compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
  compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
  return the scaled feature $\frac{(S_1 - S_2)a}{2wh}$
end if
if *feature type* (b) **then**
  set the original feature support size $a \leftarrow 3wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, h \leftarrow \mathrm{J}he/24\mathrm{K}$
  $w \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 3we/24\mathrm{K}/3,\ 3\kappa \leq e - j + 1\}$
  compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
  compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
  compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + 2w, j + 3w - 1]$
  return the scaled feature $\frac{(S_1 - S_2 + S_3)a}{3wh}$
end if
if *feature type* (c) **then**
  set the original feature support size $a \leftarrow 2wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, w \leftarrow \mathrm{J}we/24\mathrm{K}$
  $h \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 2he/24\mathrm{K}/2,\ 2\kappa \leq e - i + 1\}$
  compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
  compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
  return the scaled feature $\frac{(S_1 - S_2)a}{2wh}$
end if
if *feature type* (d) **then**
  set the original feature support size $a \leftarrow 3wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, w \leftarrow \mathrm{J}we/24\mathrm{K}$
  $h \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 3he/24\mathrm{K}/3,\ 3\kappa \leq e - i + 1\}$
  compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
  compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
  compute the sum S_3 of the pixels in $[i + 2h, i + 3h - 1] \times [j, j + w - 1]$
  return the scaled feature $\frac{(S_1 - S_2 + S_3)a}{3wh}$
end if
if *feature type* (e) **then**
  set the original feature support size $a \leftarrow 4wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}$
  $w \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 2we/24\mathrm{K}/2,\ 2\kappa \leq e - j + 1\}$
  $h \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 2he/24\mathrm{K}/2,\ 2\kappa \leq e - i + 1\}$
  compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$
  compute the sum S_2 of the pixels in $[i + h, i + 2h - 1] \times [j, j + w - 1]$
  compute the sum S_3 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$
  compute the sum S_4 of the pixels in $[i + h, i + 2h - 1] \times [j + w, j + 2w - 1]$
  return the scaled feature $\frac{(S_1 - S_2 - S_3 + S_4)a}{4wh}$
end if

(a) (b)

Figure 3: Some supervised examples: (a) positive examples (b) negative examples. All of them are 24×24 grayscale images. See Section 2.4 for more on this dataset.

should look like all by itself because it is humans, rather than the algorithm, who perform the labeling and the first round of feature selection, as described in the previous section.

The building block of the Viola-Jones face detector is a *decision stump*, or a depth one decision tree, parametrized by a feature $\mathfrak{f} \in \{1, \cdots, d\}$, a threshold $\mathfrak{t} \in \mathbb{R}$ and a toggle $\mathcal{T} \in \{-1, 1\}$. Given an observation $x \in \mathbb{R}^d$, a decision stump h predicts its label using the following rule

$$h(x) = (1_{\pi_{\mathfrak{f}} x \geq \mathfrak{t}} - 1_{\pi_{\mathfrak{f}} x < \mathfrak{t}})\mathcal{T} = (1_{\pi_{\mathfrak{f}} x \geq \mathfrak{t}} - 1_{\pi_{\mathfrak{f}} x < \mathfrak{t}})1_{\mathcal{T}=1} + (1_{\pi_{\mathfrak{f}} x < \mathfrak{t}} - 1_{\pi_{\mathfrak{f}} x \geq \mathfrak{t}})1_{\mathcal{T}=-1} \in \{-1, 1\}, \tag{2}$$

where $\pi_{\mathfrak{f}} x$ is the feature vector's $\mathfrak{f}$-th coordinate. Several comments follow:

1. Any additional pattern produced by permuting black and white rectangles in an existing pattern (see Figure 2) is superfluous. Because such a feature is merely the opposite of an existing feature, only a sign change for $\mathfrak{t}$ and $\mathcal{T}$ is needed to have the same classification rule.

2. If the training examples are sorted in ascending order of a given feature $\mathfrak{f}$, a linear time exhaustive search on the threshold and toggle can find a decision stump using this feature that attains the lowest empirical loss

$$\sum_{i=1}^{n} w_i 1_{y_i \neq h(x_i)}, \tag{3}$$

on the training set (see Algorithm 4). Imagine a threshold placed somewhere on the real line, if the toggle is set to 1, the resulting rule will declare an example x positive if $\pi_{\mathfrak{f}} x$ is greater than the threshold and negative otherwise. This allows us to evaluate the rule's empirical error, thereby selecting the toggle that fits the dataset better (lines 8–16 of Algorithm 4).

Since margin

$$\min_{i:\, y_i=-1} |\pi_{\mathfrak{f}} x_i - \mathfrak{t}| + \min_{i:\, y_i=1} |\pi_{\mathfrak{f}} x_i - \mathfrak{t}|,$$

and risk, or the expectation of the empirical loss (3), are closely related [3, 6, 7], of two decision stumps having the same empirical risk, the one with a larger margin is preferred (line 14 of Algorithm 4). Thus in the absence of duplicates, there are $n + 1$ possible thresholds and the one with the smallest empirical loss should be chosen. However it is possible to have the same feature values from different examples and extra care must be taken to handle this case properly (lines 27–32 of Algorithm 4).

Algorithm 4 Decision Stump by Exhaustive Search

Input: n training examples arranged in ascending order of feature $\pi_{\mathfrak{f}} x_i$: $\pi_{\mathfrak{f}} x_{i_1} \leq \pi_{\mathfrak{f}} x_{i_2} \leq \cdots \leq \pi_{\mathfrak{f}} x_{i_n}$, probabilistic example weights $(w_k)_{1 \leq k \leq n}$.
Output: the decision stump's threshold τ, toggle $\mathcal{T}$, error $\mathcal{E}$ and margin $\mathcal{M}$.
Initialization: $\tau \leftarrow \min_{1 \leq i \leq n} \pi_{\mathfrak{f}} x_i - 1$, $\mathcal{M} \leftarrow 0$ and $\mathcal{E} \leftarrow 2$ (an arbitrary upper bound of the empirical loss).
Sum up the weights of the positive (resp. negative) examples whose $\mathfrak{f}$-th feature is bigger than the present threshold: $W_1^+ \leftarrow \sum_{i=1}^n w_i 1_{y_i=1}$ (resp. $W_{-1}^+ \leftarrow \sum_{i=1}^n w_i 1_{y_i=-1}$).
Sum up the weights of the positive (resp. negative) examples whose $\mathfrak{f}$-th feature is smaller than the present threshold: $W_1^- \leftarrow 0$ (resp. $W_{-1}^- \leftarrow 0$).
Set iterator $j \leftarrow 0$, $\widehat{\tau} \leftarrow \tau$ and $\widehat{\mathcal{M}} \leftarrow \mathcal{M}$.
while true **do**
  Select the toggle to minimize the weighted error: $\text{error}_+ \leftarrow W_1^- + W_{-1}^+$ and $\text{error}_- \leftarrow W_1^+ + W_{-1}^-$.
  if $\text{error}_+ < \text{error}_-$ **then**
    $\widehat{\mathcal{E}} \leftarrow \text{error}_+$ and $\widehat{\mathcal{T}} \leftarrow 1$.
  else
    $\widehat{\mathcal{E}} \leftarrow \text{error}_-$ and $\widehat{\mathcal{T}} \leftarrow -1$.
  end if
  if $\widehat{\mathcal{E}} < \mathcal{E}$ or $\widehat{\mathcal{E}} = \mathcal{E}$ & $\widehat{\mathcal{M}} > \mathcal{M}$ **then**
    $\mathcal{E} \leftarrow \widehat{\mathcal{E}}$, $\tau \leftarrow \widehat{\tau}$, $\mathcal{M} \leftarrow \widehat{\mathcal{M}}$ and $\mathcal{T} \leftarrow \widehat{\mathcal{T}}$.
  end if
  if $j = n$ **then**
    Break.
  end if
  $j \leftarrow j + 1$.
  while true **do**
    if $y_{i_j} = -1$ **then**
      $W_{-1}^- \leftarrow W_{-1}^- + w_{i_j}$ and $W_{-1}^+ \leftarrow W_{-1}^+ - w_{i_j}$.
    else
      $W_1^- \leftarrow W_1^- + w_{i_j}$ and $W_1^+ \leftarrow W_1^+ - w_{i_j}$.
    end if
    To find a new valid threshold, we need to handle duplicate features.
    if $j = n$ or $\pi_{\mathfrak{f}} x_{i_j} \neq \pi_{\mathfrak{f}} x_{i_{j+1}}$ **then**
      Break.
    else
      $j \leftarrow j + 1$.
    end if
  end while
  if $j = n$ **then**
    $\widehat{\tau} \leftarrow \max_{1 \leq i \leq n} \pi_{\mathfrak{f}} x_i + 1$ and $\widehat{\mathcal{M}} \leftarrow 0$.
  else
    $\widehat{\tau} \leftarrow (\pi_{\mathfrak{f}} x_{i_j} + \pi_{\mathfrak{f}} x_{i_{j+1}})/2$ and $\widehat{\mathcal{M}} \leftarrow \pi_{\mathfrak{f}} x_{i_{j+1}} - \pi_{\mathfrak{f}} x_{i_j}$.
  end if
end while

Algorithm 5 Best Stump

Input: n training examples, their probabilistic weights $(w_i)_{1\leq i\leq n}$, number of features d.
Output: the best decision stump's threshold, toggle, error and margin.
Set the best decision stump's error to 2.
for $\mathfrak{f} = 1$ to d **do**
    Compute the decision stump associated with feature $\mathfrak{f}$ using Algorithm 4.
    if this decision stump has a lower weighted error (3) than the best stump or a wider margin if the weighted error are the same **then**
        set this decision stump to be the best.
    end if
end for

Algorithm 6 Adaboost

Input: n training examples $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$, $1 \leq i \leq n$, number of training rounds T.
Parameter: the initial probabilistic weights $w_i(1)$ for $1 \leq i \leq n$.
Output: a strong learner/committee.
for $t = 1$ to T **do**
    Run Algorithm 5 to train a decision stump h_t using the weights $w_\cdot(t)$ and get its weighted error ϵ_t

$$\epsilon_t = \sum_{i=1}^{n} w_i(t) 1_{h_t(x_i)\neq y_i},$$

    if $\epsilon_t = 0$ and $t = 1$ **then**
        training ends and return $h_1(\cdot)$.
    else
        set $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$.
        update the weights

$$\forall i,\ \ w_i(t+1) = \frac{w_i(t)}{2}\Big(\frac{1}{\epsilon_t} 1_{h_t(x_i)\neq y_i} + \frac{1}{1-\epsilon_t} 1_{h_t(x_i)=y_i}\Big).$$

    end if
end for
Return the rule

$$f^T(\cdot) = \text{sign}\Big[\sum_{t=1}^{T} \alpha_t h_t(\cdot)\Big].$$

By adjusting individual example weights (Algorithm 6 line 10), Adaboost makes more effort to learn harder examples and adds more decision stumps (see Algorithm 5) in the process. Intuitively, in the final voting, a stump h_t with lower empirical loss is rewarded with a bigger say (a higher α_t, see Algorithm 6 line 9) when a T-member committee (vote-based classifier) assigns an example according to

$$f^T(\cdot) = \text{sign}\big[\sum_{t=1}^{T} \alpha_t h_t(\cdot)\big].$$

How the training examples should be weighed is explained in detail in Appendix A. Figure 4 shows an instance where Adaboost reduces false positive and false negative rates simultaneously as more and more stumps are added to the committee. For notational simplicity, we denote the empirical loss by

$$\sum_{i=1}^{n} w_i(1) 1_{y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \leq 0} := \mathbb{P}(f^T(X) \neq Y),$$

where (X, Y) is a random couple distributed according to the probability $\mathbb{P}$ defined by the weights $w_i(1)$, $1 \leq i \leq n$ set when the training starts. As the empirical loss goes to zero with T, so do both false positive $\mathbb{P}(f^T(X) = 1|Y = -1)$ and false negative rates $\mathbb{P}(f^T(X) = -1|Y = 1)$ owing to

$$\mathbb{P}(f^T(X) \neq Y) = \mathbb{P}(Y = 1)\mathbb{P}(f^T(X) = -1|Y = 1) + \mathbb{P}(Y = -1)\mathbb{P}(f^T(X) = 1|Y = -1).$$

Thus the detection rate

$$\mathbb{P}(f^T(X) = 1|Y = 1) = 1 - \mathbb{P}(f^T(X) = -1|Y = 1),$$

must tend to 1.

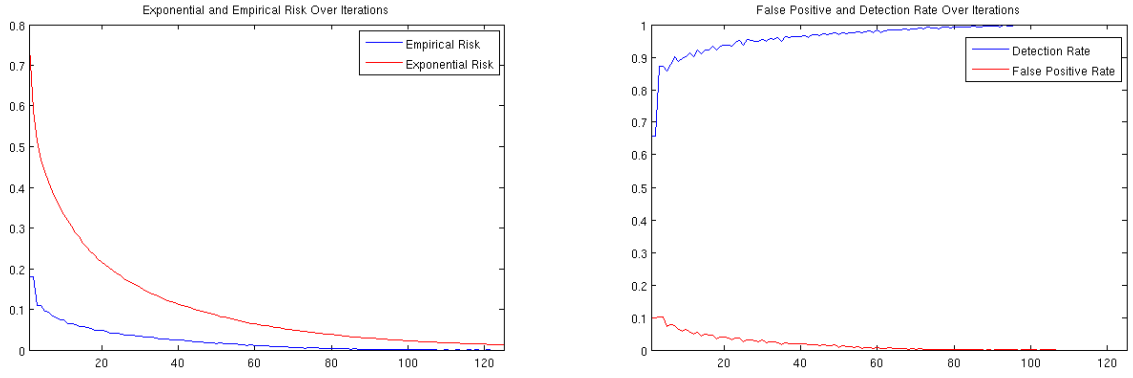
Thus the size T of the trained committee depends on the targeted false positive and false negative rates. In addition, let us mention that, given n_- negative and n_+ positive examples in a training pool, it is customary to give a negative (resp. positive) example an initial weight equal to $0.5/n_-$ (resp. $0.5/n_+$) so that Adaboost does not favor either category at the beginning.

2.3 Attentional Cascade

In theory, Adaboost can produce a single committee of decision stumps that generalizes well. However, to achieve that, an enormous negative training set is needed at the outset to gather all possible negative patterns. In addition, a single committee implies that all the windows inside an image have to go through the same lengthy decision process. There has to be another more cost-efficient way.

The prior probability for a face to appear in an image bears little relevance to the presented classifier construction because it requires both the empirical false negative and false positive rate to approach zero. However, our own experience tells us that in an image, a rather limited number of sub-windows deserve more attention than others. This is true even for face-intensive group photos. Hence the idea of a multi-layer attentional cascade which embodies a principle akin to that of Shannon coding: the algorithm should deploy more resources to work on those windows more likely to contain a face while spending as little effort as possible on the rest.

Each layer in the attentional cascade is expected to meet a training target expressed in false positive and false negative rates: among n negative examples declared positive by all of its preceding layers, layer l ought to recognize at least $(1 - \gamma_l)n$ as negative and meanwhile try not to sacrifice its performance on the positives: the detection rate should be maintained above $1 - \beta_l$.



(a) (b)

Figure 4: Algorithm 6 ran with equally weighted 2500 positive and 2500 negative examples. Figure (a) shows that the empirical risk and its upper bound, interpreted as the exponential loss (see Appendix A), decrease steadily over iterations. This implies that false positive and false negative rates must also decrease, as observed in (b).

At the end of the day, only the generalization error counts which unfortunately can only be estimated with some validation examples that Adaboost is not allowed to see at the training phase. Hence in Algorithm 10 at line 10, a conservative choice is made as to how one assesses the error rates: the higher false positive rate obtained from training and validation is used to evaluate how well the algorithm has learned to distinguish faces from non-faces. The false negative rate is assessed in the same way.

It should be kept in mind that Adaboost by itself does not favor either error rate: it aims to reduce both simultaneously rather than one at the expense of the other. To allow flexibility, one additional control $\mathfrak{s} \in [-1, 1]$ is introduced to shift the classifier

$$f_{\mathfrak{s}}^{T}(\cdot) = \operatorname{sign}\Big[\sum_{t=1}^{T} \alpha_t \big(h_t(\cdot) + \mathfrak{s}\big)\Big], \tag{4}$$

so that a strictly positive $\mathfrak{s}$ makes the classifier more inclined to predict a face and vice versa.

To enforce an efficient resource allocation, the committee size should be small in the first few layers and then grow gradually so that a large number of easy negative patterns can be eliminated with little computational effort (see Figure 5).

Appending a layer to the cascade means that the algorithm has learned to reject a few new negative patterns previously viewed as difficult, all the while keeping more or less the same positive training pool. To build the next layer, more negative examples are thus required to make the training process meaningful. To replace the detected negatives, we run the cascade on a large set of gray images with no human face and collect their false positive windows. The same procedure is used for constructing and replenishing the validation set (see Algorithm 7). Since only 24×24 sized examples can be used in the training phase, those bigger false positives are down-sampled (Algorithm 8) and recycled using Algorithm 9.

Assume that at layer l, a committee of T_l weak classifiers is formed along with a shift $\mathfrak{s}$ so that the classifier's performance on training and validation set can be measured. Let us denote the achieved false positive and false negative rate by $\widehat{\gamma}_l$ and $\widehat{\beta}_l$. Depending on their relation with the targets γ_l and β_l, four cases are presented:

1. If the layer training target is fulfilled (Algorithm 10 line 11: $\widehat{\gamma}_l \leq \gamma_l$ and $\widehat{\beta}_l \leq \beta_l$), the algorithm moves on to training the next layer if necessary.

Algorithm 7 Detecting faces with an Adaboost trained cascade classifier

Input: an $M \times N$ grayscale image I and an L-layer cascade of shifted classifiers trained using Algorithm 10
Parameter: a window scale multiplier $\mathfrak{c}$
Output: $\mathcal{P}$, the set of windows declared positive by the cascade
Set $\mathcal{P} = \{[i, i+e-1] \times [j, j+e-1] \subset I : e = \mathrm{J}24\mathfrak{c}^{\kappa}\mathrm{K}, \kappa \in \mathbb{N}\}$
for $l = 1$ to L **do**
  for every window in $\mathcal{P}$ **do**
    Remove the windowed image's mean and compute its standard deviation.
    if the standard deviation is bigger than 1 **then**
      divide the image by this standard deviation and compute its features required by the shifted classifier at layer l with Algorithm 3
      if the cascade's l-th layer predicts negative **then**
        discard this window from $\mathcal{P}$
      end if
    else
      discard this window from $\mathcal{P}$
    end if
  end for
end for
Return $\mathcal{P}$

Algorithm 8 Downsampling a square image

Input: an $e \times e$ image I ($e > 24$)
Output: a downsampled image O of dimension 24×24
Blur I using a Gaussian kernel with standard deviation $\sigma = 0.6\sqrt{(\frac{e}{24})^2 - 1}$
Allocate a matrix O of dimension 24×24
for $i = 0$ to 23 **do**
  for $j = 0$ to 23 **do**
    Compute the scaled coordinates $\tilde{i} \leftarrow \frac{e-1}{25}(i+1)$, $\tilde{j} \leftarrow \frac{e-1}{25}(j+1)$
    Set $\tilde{i}_{max} \leftarrow \min(\mathrm{J}\tilde{i}\mathrm{K}+1, e-1)$, $\tilde{i}_{min} \leftarrow \max(0, \mathrm{J}\tilde{i}\mathrm{K})$, $\tilde{j}_{max} \leftarrow \min(\mathrm{J}\tilde{j}\mathrm{K}+1, e-1)$, $\tilde{j}_{min} \leftarrow \max(0, \mathrm{J}\tilde{j}\mathrm{K})$
    Set $O(i,j) = \frac{1}{4}\big[I(\tilde{i}_{max}, \tilde{j}_{max}) + I(\tilde{i}_{min}, \tilde{j}_{max}) + I(\tilde{i}_{min}, \tilde{j}_{min}) + I(\tilde{i}_{max}, \tilde{j}_{min})\big]$
  end for
end for
Return O

Algorithm 9 Collecting false positive examples for training a cascade's $(L+1)$-th layer

Input: a set of grayscale images with no human faces and an L-layer cascade of shifted classifiers
Parameter: a window scale multiplier $\mathfrak{c}$
Output: a set of false positive examples $\mathcal{V}$
for every grayscale image **do**
  Run Algorithm 7 to get all of its false positives $\mathcal{Q}$
  for every windowed image in $\mathcal{Q}$ **do**
    if the window size is bigger than 24×24 **then**
      downsample this subimage using Algorithm 8 and run Algorithm 7 on it
      if the downsampled image remains positive **then**
        accept this false positive to $\mathcal{V}$
      end if
    else
      accept this false positive to $\mathcal{V}$
    end if
  end for
end for
Return $\mathcal{V}$

2. If there is room to improve the detection rate (Algorithm 10 line 13: $\widehat{\gamma}_l \leq \gamma_l$ and $\widehat{\beta}_l > \beta_l$), $\mathfrak{s}$ is increased by u, a prefixed unit.

3. If there is room to improve the false positive rate (Algorithm 10 line 20: $\widehat{\gamma}_l > \gamma_l$ and $\widehat{\beta}_l \leq \beta_l$), $\mathfrak{s}$ is decreased by u.

4. If both error rates fall short of the target (Algorithm 10 line 27: $\widehat{\gamma}_l > \gamma_l$ and $\widehat{\beta}_l > \beta_l$), the algorithm, if the current committee does not exceed a prefixed layer specific size limit, trains one more member to add to the committee.

Special attention should be paid here: the algorithm could alternate between case 2 and 3 and create a dead loop. A solution is to halve the unit u every time it happens until u becomes smaller than 10^{-5}. When it happens, one more round of training at this layer is recommended.

As mentioned earlier, to prevent a committee from growing too big, the algorithm stops refining its associated layer after a layer dependent size limit is breached (Algorithm 10 line 28). In this case, the shift $\mathfrak{s}$ is set to the smallest value that satisfies the false negative requirement. A harder learning case is thus deferred to the next layer. This strategy works because Adaboost's inability to meet the training target can often be explained by the fact that a classifier trained on a limited number of examples might not generalize well on the validation set. However, those hard negative patterns should ultimately appear and be learned if the training goes on, albeit one bit at a time.

To analyze how well the cascade does, let us assume that at layer l, Adaboost can deliver a classifier $f_{l,\mathfrak{s}_l}^{T_l}$ with false positive γ_l and detection rate $1-\beta_l$. In probabilistic terms, it means

$$\mathbb{P}(f_{l,\mathfrak{s}_l}^{T_l}(X)=1|Y=1) \geq 1-\beta_l \text{ and } \mathbb{P}(f_{l,\mathfrak{s}_l}^{T_l}(X)=1|Y=-1) \leq \gamma_l,$$

where by abuse of notation we keep $\mathbb{P}$ to denote the probability on some image space. If Algorithm 10 halts at the end of L iterations, the decision rule is

$$f_{cascade}(X) = 2\big(\prod_{l=1}^{L} 1_{f_{l,\mathfrak{s}_l}^{T_l}(X)=1} - \frac{1}{2}\big).$$

Algorithm 10 Attentional Cascade

Input: n training positives, m validation positives, two sets of gray images with no human faces to draw training and validation negatives, desired overall false positive rate γ_o, and targeted layer false positive and detection rate γ_l and $1-\beta_l$.
Parameter: maximum committee size at layer l: $N_l = \min(10l+10, 200)$.
Output: a cascade of committees.
Set the attained overall false positive rate $\widehat{\gamma}_o \leftarrow 1$ and layer count $l \leftarrow 0$.
Randomly draw $10n$ negative training examples and m negative validation examples.
while $\widehat{\gamma}_o > \gamma_o$ **do**
  $u \leftarrow 10^{-2}$, $l \leftarrow l+1$, $\mathfrak{s}_l \leftarrow 0$, and $T_l \leftarrow 1$.
  Run Algorithm 6 on the training set to produce a classifier $f_l^{T_l} = \operatorname{sign}\big[\sum_{t=1}^{T_l} \alpha_t h_t\big]$.
  Run the $\mathfrak{s}_l$-shifted classifier $f_{l,\mathfrak{s}_l}^{T_l} = \operatorname{sign}\big[\sum_{t=1}^{T_l} \alpha_t\big(h_t + \mathfrak{s}_l\big)\big]$, on both the training and validation set to obtain the empirical and generalized false positive (resp. false negative) rate γ_e and γ_g (resp. β_e and β_g).
  $\widehat{\gamma}_l \leftarrow \max(\gamma_e, \gamma_g)$ and $\widehat{\beta}_l \leftarrow \max(\beta_e, \beta_g)$.
  if $\widehat{\gamma}_l \le \gamma_l$ and $1-\widehat{\beta}_l \ge 1-\beta_l$ **then**
    $\widehat{\gamma}_o \leftarrow \widehat{\gamma}_o \times \widehat{\gamma}_l$.
  else if $\widehat{\gamma}_l \le \gamma_l$, $1-\widehat{\beta}_l < 1-\beta_l$ and $u > 10^{-5}$ (there is room to improve the detection rate) **then**
    $\mathfrak{s}_l \leftarrow \mathfrak{s}_l + u$.
    if the trajectory of $\mathfrak{s}_l$ is not monotone **then**
      $u \leftarrow u/2$.
      $\mathfrak{s}_l \leftarrow \mathfrak{s}_l - u$.
    end if
    Go to line 9.
  else if $\widehat{\gamma}_l > \gamma_l$, $1-\widehat{\beta}_l \ge 1-\beta_l$ and $u > 10^{-5}$ (there is room to improve the false positive rate) **then**
    $\mathfrak{s}_l \leftarrow \mathfrak{s}_l - u$.
    if the trajectory of $\mathfrak{s}_l$ is not monotone **then**
      $u \leftarrow u/2$.
      $\mathfrak{s}_l \leftarrow \mathfrak{s}_l + u$.
    end if
    Go to line 9.
  else
    if $T_l > N_l$ **then**
      $\mathfrak{s}_l \leftarrow -1$
      while $1-\widehat{\beta}_l < 0.99$ **do**
        Run line 9 and 10.
      end while
      $\widehat{\gamma}_o \leftarrow \widehat{\gamma}_o \times \widehat{\gamma}_l$.
    else
      $T_l \leftarrow T_l + 1$ (Train one more member to add to the committee.)
      Go to line 8.
    end if
  end if
  Remove the false negatives and true negatives detected by the current cascade

$$f_{cascade}(X) = 2\Big(\prod_{p=1}^{l} 1_{f_{p,\mathfrak{s}_p}^{T_p}(X)=1} - \frac{1}{2}\Big).$$

  Use this cascade with Algorithm 9 to draw some false positives so that there are n training negatives and m validation negatives for the next round.
end while
Return the cascade.

A window is thus declared positive if and only if all its component layers hold the same opinion

$$\begin{aligned}
&\mathbb{P}(f_{cascade}(X)=1|Y=-1)\\
=&\mathbb{P}(\bigcap_{l=1}^{L}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=-1)\\
=&\mathbb{P}(f_{L,\mathfrak{s}_L}^{T_L}(X)=1|\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}\text{ and }Y=-1)\mathbb{P}(\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=-1)\\
\leq&\gamma_l\mathbb{P}(\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=-1)\\
\leq&\gamma_l^L.
\end{aligned}$$

Likewise, the overall detection rate can be estimated as follows

$$\begin{aligned}
&\mathbb{P}(f_{cascade}(X)=1|Y=1)\\
=&\mathbb{P}(\bigcap_{l=1}^{L}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=1)\\
=&\mathbb{P}(f_{L,\mathfrak{s}_L}^{T_L}(X)=1|\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}\text{ and }Y=1)\mathbb{P}(\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=1)\\
\geq&(1-\beta_l)\mathbb{P}(\bigcap_{l=1}^{L-1}\{f_{l,\mathfrak{s}_l}^{T_l}(X)=1\}|Y=1)\\
\geq&(1-\beta_l)^L.
\end{aligned}$$

In other words, if the empirically obtained rates are any indication, a faceless window will have a probability higher than $1-\gamma_l$ to be labeled as such at each layer, which effectively directs the cascade classifier's attention on those more likely to have a face (see Figure 6).

2.4 Dataset and Experiments

A few words on the actual cascade training carried out on a 8-core Linux machine with 48G memory. We first downloaded 2897 different images without human faces from [4, 9, 5], National Oceanic and Atmospheric Administration (NOAA) Photo Library[2] and European Southern Observatory[3]. They were divided into two sets containing 2451 and 446 images respectively for training and validation. 1000 training and 1000 validation positive examples from an online source[4] were used. The training process lasted for around 24 hours before producing a 31-layer cascade. It took this long because it became harder to get 2000 false positives (1000 for training and 1000 for validation) using Algorithm 9 with a more discriminative cascade: the algorithm needed to examine more images before it could come across enough good examples. The targeted false positive and false negative rate for each layer were set to 0.5 and 0.995 respectively and Figure 7 shows how the accumulated false positive rate as defined at line 12 and 33 of Algorithm 10 evolves together with the committee size. The fact that the later layers required more intensive training also contributed to a long training phase.

[2] http://www.photolib.noaa.gov/

[3] http://www.eso.org/public/images/

[4] http://www.cs.wustl.edu/~pless/559/Projects/faceProject.html

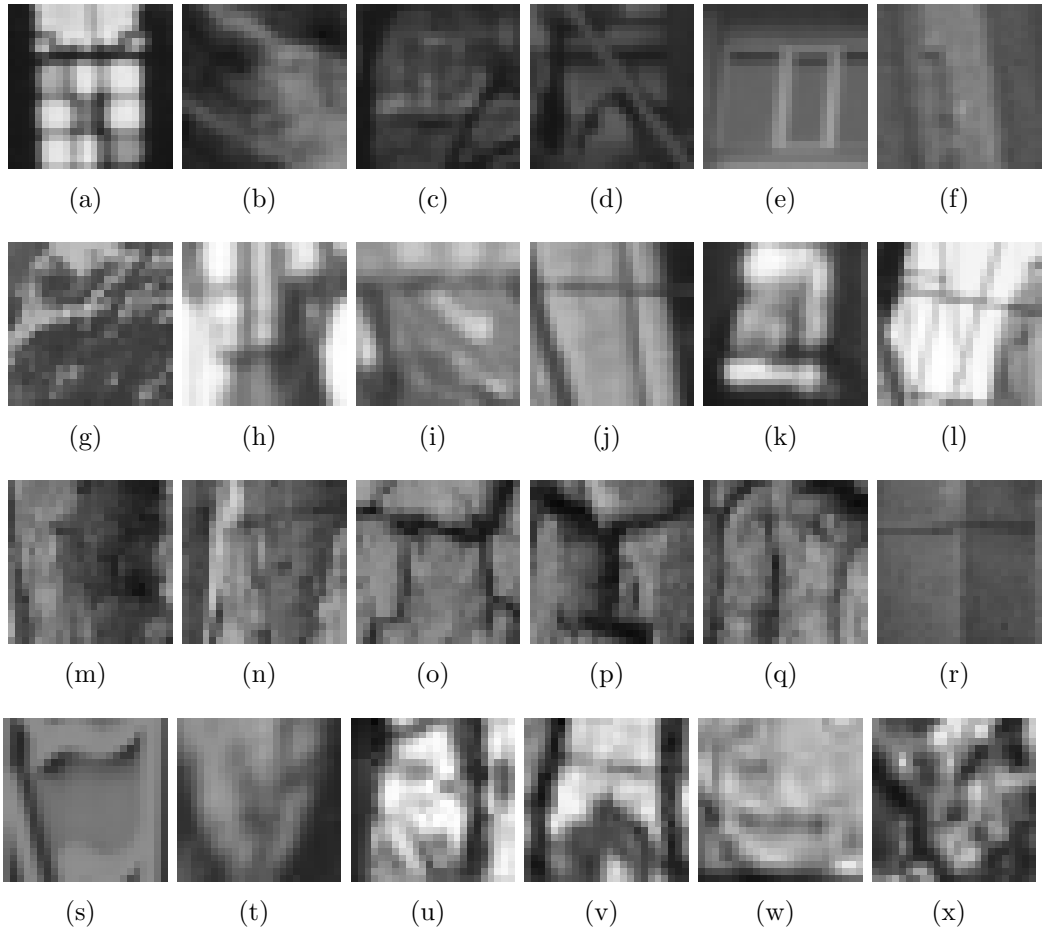


Figure 5: A selection of negative training examples at round 21 (a) (b) (c) (d) (e) (f), round 26 (g) (h) (i) (j) (k) (l), round 27 (m) (n) (o) (p) (q) (r), round 28 (s) (t) (u) (v) (w) (x). Observe how the negative training examples become increasingly difficult to discriminate from real faces.

3 Post-Processing

Figure 6(d) and Figure 8(a) show that the same face can be detected multiple times by a correctly trained cascade. This should come as no surprise as the positive examples (see Figure 3(a)) do allow a certain flexibility in pose and expression. On the contrary, many false positives do not enjoy this stability, despite the fact that taken out of context, some of them do look like a face (also see Figure 5). This observation lends support to the following *detection confidence* based heuristics for further reducing false positives and cleaning up the detected result (see Algorithm 11):

1. A detected window contains a face if and only if a sufficient number of other *adjacent* detected windows of the *same* size confirm it. To require windows of exactly the same size is not stringent because the test window sizes are quantified (see Algorithm 7 line 2). In this implementation, the window size multiplier is 1.5. Two $e \times e$ detected windows are said to be adjacent if and only if between the upper left corners of these two windows there is a path formed by the upper left corners of some detected windows of the same size. This condition is easily checked with the connected component algorithm[5] [8]. The number of test windows detecting a face is presumed to grow linearly with its size. This suggests the quotient of the cardinality of a connected component of adjacent windows by their common size e as an adequate confidence measure.

[5]We obtained a version from http://alumni.media.mit.edu/~rahimi/connected/.

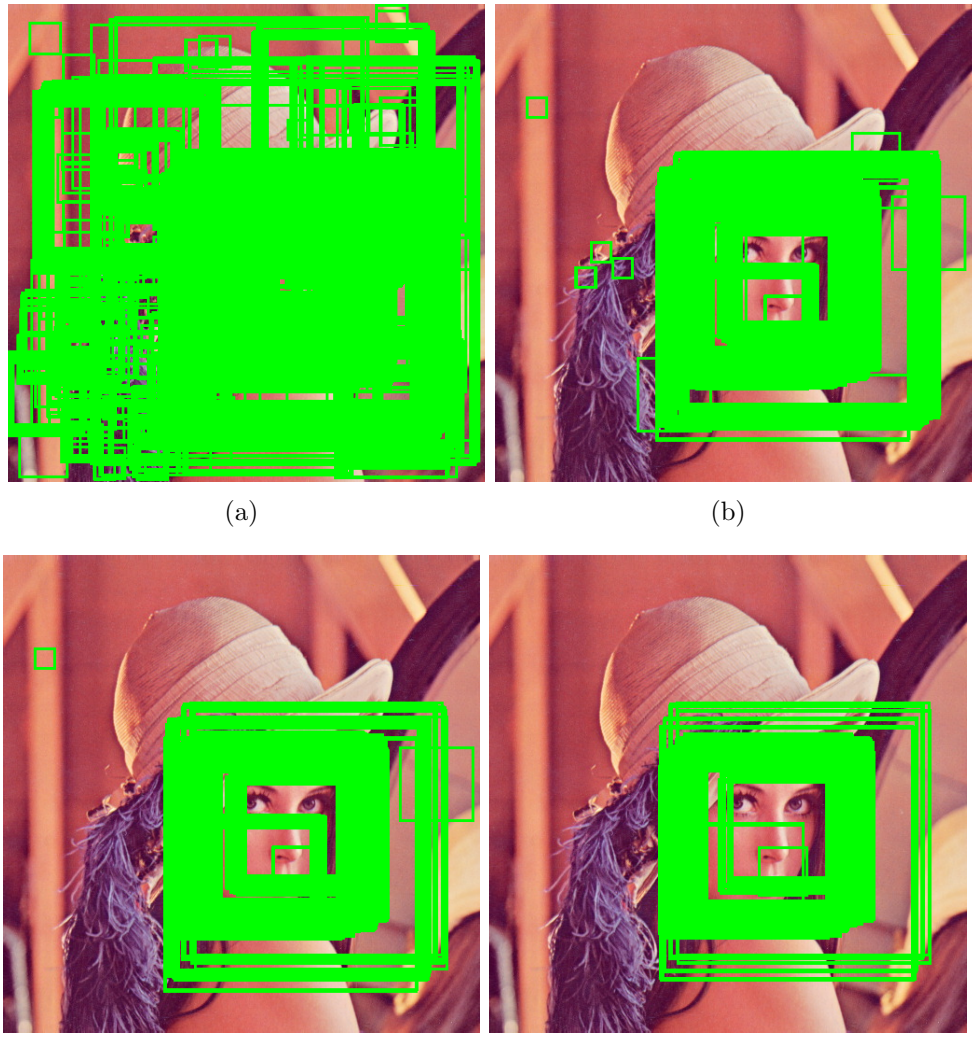


Figure 6: How the trained cascade performs with (a) 16 layers, (b) 21 layers, (c) 26 layers and (d) 31 layers: the more layers, the less false positives.

This quotient is then compared to the scale invariant threshold empirically set at $3/24$, which means that to confirm the detection of a face of size 24×24, three adjacent detected windows are sufficient.

2. It is possible for the remaining detected windows to overlap after the previous test. In this case, we distinguish two scenarios (Algorithm 11 line 15–25):

   (a) If the smaller window's center is outside the bigger one, keep both.

   (b) Keep the one with higher detection confidence otherwise.

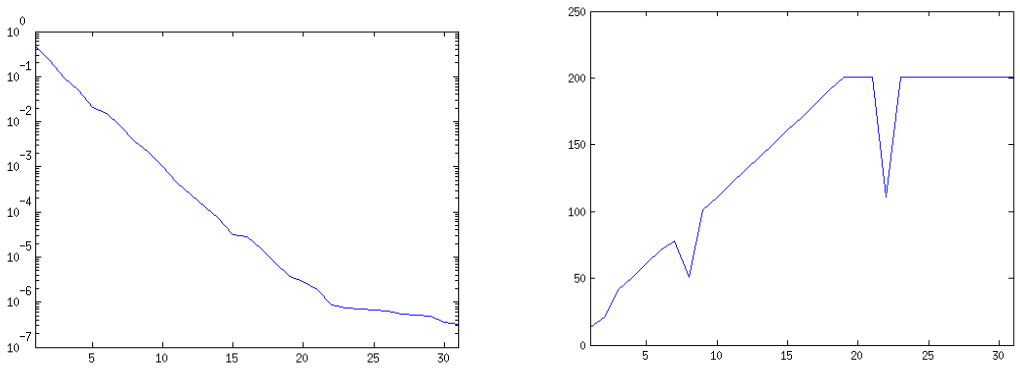
Finally, to make the detector slightly more rotation-invariant, in this implementation, we decided to run Algorithm 7 three times, once on the input image, once on a clockwise rotated image and once on an anti-clockwise rotated image before post-processing all the detected windows (see Algorithm 12). In addition, when available, color also conveys valuable information to help further eliminate false positives. Hence in the current implementation, after the robustness test, an option is offered as to whether color images should be post-processed with this additional step (see Algorithm 13). If so, a detected window is declared positive only if it passes both tests.

Algorithm 11 Post-Processing

Input: a set G windows declared positive on an $M \times N$ grayscale image
Parameter: minimum detection confidence threshold $\mathfrak{r}$
Output: a reduced set of positive windows $\mathcal{P}$
Create an $M \times N$ matrix E filled with zeros.
for each window $w \in G$ **do**
  Take w's upper left corner coordinates (i, j) and its size e and set $E(i, j) \leftarrow e$
end for
Run a connected component algorithm on E.
for each component C formed by $|C|$ detected windows of dimension $e_C \times e_C$ **do**
  if its detection confidence $|C|e_C^{-1} > \mathfrak{r}$ **then**
    send one representing window to $\mathcal{P}$
  end if
end for
Sort the elements in $\mathcal{P}$ in ascending order of window size.
for window $i = 1$ to $|\mathcal{P}|$ **do**
  for window $j = i + 1$ to $|\mathcal{P}|$ **do**
    if window j remains in $\mathcal{P}$ and the center of window i is inside of window j **then**
      if window i has a higher detection confidence than window j **then**
        remove window j from $\mathcal{P}$
      else
        remove window i from $\mathcal{P}$ and break from the inner loop
      end if
    end if
  end for
end for
Return $\mathcal{P}$.

Algorithm 12 Face detection with image rotation

1: **Input:** an $M \times N$ grayscale image I
2: **Parameter:** rotation θ
3: **Output:** a set of detected windows $\mathcal{P}$
4: Rotate the image about its center by θ and $-\theta$ to have I_θ and $I_{-\theta}$
5: Run Algorithm 7 on I, I_θ and $I_{-\theta}$ to obtain three detected window sets $\mathcal{P}$, $\mathcal{P}_\theta$ and $\mathcal{P}_{-\theta}$ respectively
6: **for** each detected window w in $\mathcal{P}_\theta$ **do**
7: Get w's upper left corner's coordinates (i_w, j_w) and its size e_w
8: Rotate (i_w, j_w) about I_θ's center by $-\theta$ to get $(\tilde{i}_w, \tilde{j}_w)$
9: Quantify the new coordinates $\tilde{i}_w \leftarrow \min(\max(0, \mathrm{J}\tilde{i}_w\mathrm{K}), M - 1)$ and $\tilde{j}_w \leftarrow \min(\max(0, \mathrm{J}\tilde{j}_w\mathrm{K}), N - 1)$
10: **if** there is no $e_w \times e_w$ window located at $(\tilde{i}_w, \tilde{j}_w)$ in $\mathcal{P}$ **then**
11: Add it to $\mathcal{P}$
12: **end if**
13: **end for**
14: Replace θ by $-\theta$ and go through lines 6–13 again
15: Return $\mathcal{P}$



(a) (b)

Figure 7: (a) Though occasionally stagnant, the accumulated false positive rate declines pretty fast with the number of layers. (b) As the learning task becomes more difficult as the cascade has more layers, more weak learners per layer are called upon. (In our experiments, the number of weak learners cannot exceed 201 per layer.)

Algorithm 13 Skin Test

Input: an $N \times N$ color image I
Output: return whether I has enough skin like pixels
Set a counter $c \leftarrow 0$
for each pixel in I **do**
  if the intensities of its green and blue channel are lower than that of its red channel **then**
    $c \leftarrow c + 1$
  end if
end for
if $c/N^2 > 0.4$ **then**
  Return true
else
  Return false
end if

A Appendix

This section explains, from a mathematical perspective, how and why Adaboost (Algorithm 6) works. We define the exponential loss

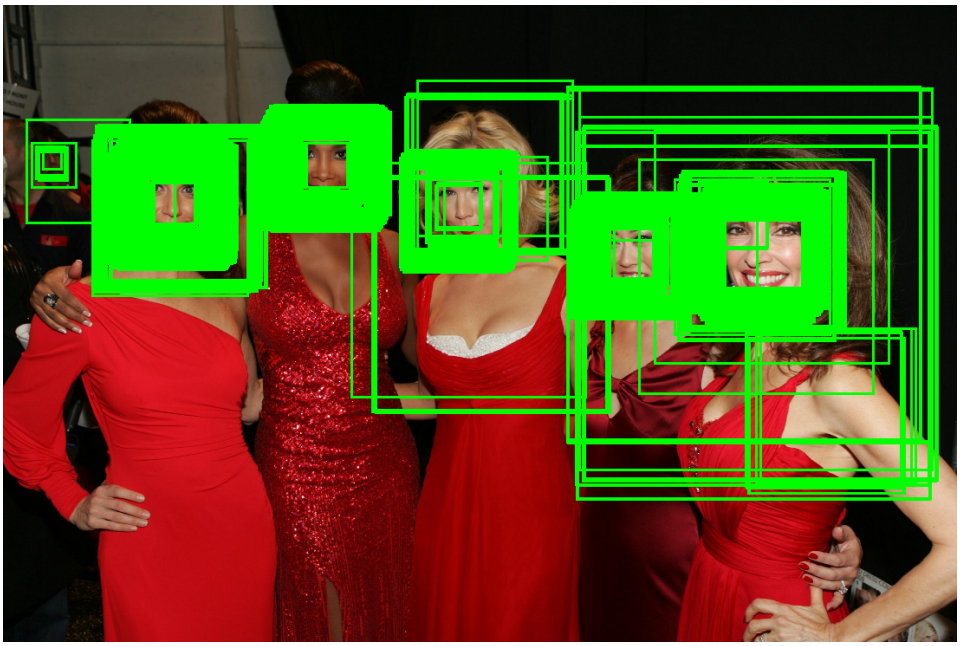
$$\forall (x, y) \in \mathbb{R}^d \times \{-1, 1\}, \quad L(y, f(x)) = \exp(-yf(x)),$$

where the classifier $f : \mathbb{R}^d \mapsto \mathbb{R}$ takes the form of a linear combination of weak classifiers

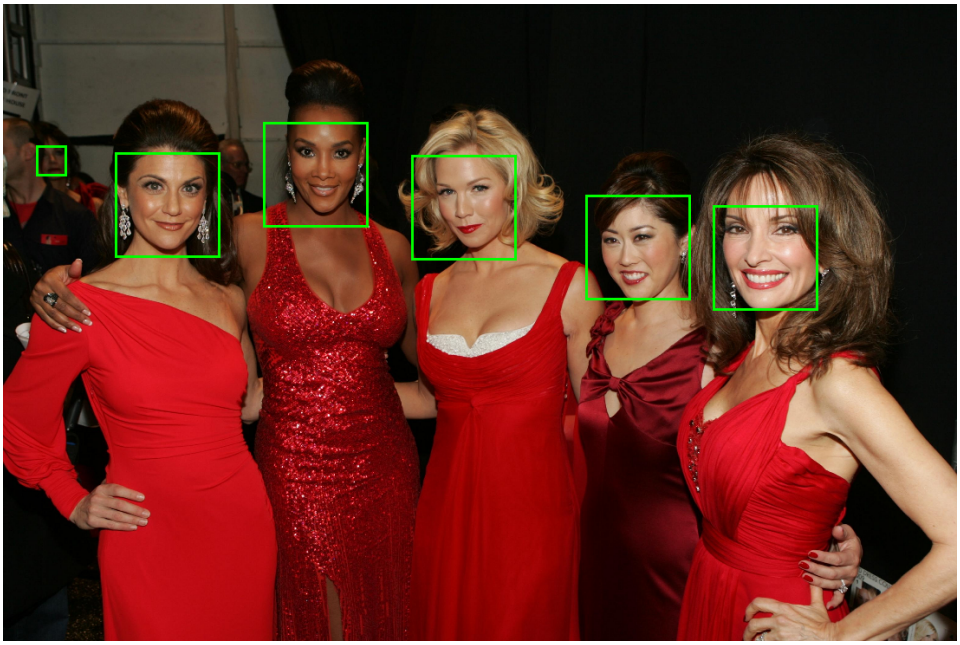
$$f(\cdot) = \sum_{t=1}^{T} \alpha_t h_t(\cdot),$$

with $T \in \mathbb{N}$, $\min_{1 \leq t \leq T} \alpha_t > 0$ and $\forall t,\ h_t(\cdot) \in \{-1, 1\}$. Naturally, the overall objective is

$$\min_{h_t, \alpha_t \geq 0} \sum_{i=1}^{n} w_i(1) L(y_i, \sum_{t=1}^{T} \alpha_t h_t(x_i)), \tag{5}$$



(a)



(b)

Figure 8: The suggested post-processing procedure further eliminates a number of false positives and beautifies the detected result using a 31 layer cascade.

with some initial probabilistic weight $w_i(1)$. A greedy approach is deployed to deduce the optimal classifiers h_t and weights α_t one after another, although there is no guarantee that the objective (5) is minimized. Given $(\alpha_s, h_s)_{1\le s<t}$, let Z_{t+1} be the weighted exponential loss attained by a t-member committee and we seek to minimize it through (h_t, α_t)

$$Z_{t+1} := \min_{h_t,\alpha_t\ge 0} \sum_{i=1}^{n} w_i(1) e^{-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)}$$

$$
\begin{aligned}
&= \min_{h_t,\alpha_t\geq 0} \sum_{i=1}^{n} D_i(t) e^{-\alpha_t y_i h_t(x_i)} \\
&= \min_{h_t,\alpha_t\geq 0} \sum_{i=1}^{n} D_i(t) e^{-\alpha_t} 1_{y_i h_t(x_i)=1} + \sum_{i=1}^{n} D_i(t) e^{\alpha_t} 1_{y_i h_t(x_i)=-1} \\
&= \min_{h_t,\alpha_t\geq 0} e^{-\alpha_t} \sum_{i=1}^{n} D_i(t) + (e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^{n} D_i(t) 1_{y_i h_t(x_i)=-1} \\
&= Z_t \min_{h_t,\alpha_t\geq 0} e^{-\alpha_t} + (e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^{n} \frac{D_i(t)}{Z_t} 1_{y_i h_t(x_i)=-1},
\end{aligned}
$$

Therefore the optimization of Z_{t+1} can be carried out in two stages: first, because of α_t's assumed positivity, we minimize the weighted error using a base learning algorithm, a decision stump for instance

$$
\begin{aligned}
\epsilon_t &:= \min_h Z_t^{-1} \sum_{i=1}^{n} D_i(t) 1_{y_i h(x_i)=-1}, \\
h_t &:= \operatorname*{argmin}_h Z_t^{-1} \sum_{i=1}^{n} D_i(t) 1_{y_i h(x_i)=-1}.
\end{aligned}
$$

In case of multiple minimizers, take h_t to be any of them. Next choose

$$
\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} = \operatorname*{argmin}_{\alpha>0} e^{-\alpha} + (e^{\alpha} - e^{-\alpha})\epsilon_t.
$$

Hence $\epsilon_t < 0.5$ is necessary, which imposes a minimal condition on the training set and the base learning algorithm. Also obtained is

$$
Z_{t+1} = 2Z_t \sqrt{\epsilon_t(1-\epsilon_t)} \leq Z_t,
$$

a recursive relation asserting the decreasing behavior of the exponential risk. Weight change thus depends on whether an observation is misclassified

$$
w_i(t+1) = \frac{D_i(t+1)}{Z_{t+1}} = \frac{D_i(t) e^{-y_i \alpha_t h_t(x_i)}}{2Z_t \sqrt{\epsilon_t(1-\epsilon_t)}} = \frac{w_i(t)}{2} \big(1_{h_t(x_i)=y_i} \frac{1}{1-\epsilon_t} + 1_{h_t(x_i)\neq y_i} \frac{1}{\epsilon_t} \big).
$$

The final boosted classifier is thus a weighted committee

$$
f(\cdot) := \operatorname{sign} \big[\sum_{t=1}^{T} \alpha_t h_t(\cdot) \big].
$$

Acknowledgements

Research partially financed by the European Research Council, advanced grant "Twelve labours", the Office of Naval research under grant N00014-97-1-0839, and by DxO labs.

Image Credits

by The Heart Truth, Flickr[6] CC-BY-SA-2.0.

The USC-SIPI[7] Image Database.

References

[1] Y. Freund and R. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of Computer and System Sciences, 55 (1997), pp. 119–139. http://dx.doi.org/10.1006/jcss.1997.1504.

[2] Y. Freund, R. Schapire, and N. Abe, *A short introduction to boosting*, Journal of Japanese Society For Artificial Intelligence, 14 (1999), pp. 771–780.

[3] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1, Springer Series in Statistics, 2001.

[4] H. Jégou, M. Douze, and C. Schmid, *Hamming embedding and weak geometric consistency for large scale image search*, in European Conference on Computer Vision, vol. I of Lecture Notes in Computer Science, Springer, 2008, pp. 304–317.

[5] A. Olmos, *A biologically inspired algorithm for the recovery of shading and reflectance images.*, Perception, 33 (2004), pp. 1463–1473. http://dx.doi.org/10.1068/p5321.

[6] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review, 65 (1958), pp. 386–408. http://dx.doi.org/10.1037/h0042519.

[7] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, *Boosting the margin: A new explanation for the effectiveness of voting methods*, The Annals of Statistics, 26 (1998), pp. 1651–1686. http://dx.doi.org/10.1214/aos/1024691352.

[8] L. Shapiro and G.C. Stockman, *Computer vision*, 2001. ISBN 0130307963.

[9] G. Tkačik, P. Garrigan, C. Ratliff, G. Milčinski, J. M. Klein, L. H. Seyfarth, P. Sterling, D. H. Brainard, and V. Balasubramanian, *Natural images from the birthplace of the human eye*, Public Library of Science One, 6 (2011), p. e20409.

[10] P. Viola and M. J. Jones, *Robust real-time face detection*, International Journal of Computer Vision, 57 (2004), pp. 137–154. http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb.

[6] http://www.flickr.com/photos/thehearttruth/3281640031/in/photostream/
[7] http://sipi.usc.edu/database/