Chapter 8
Object Detection

Jianxin Wu and James M. Rehg

8.1 Introduction

Over the past twenty years, data-driven methods have become a dominant paradigm for computer vision, with numerous practical successes. In difficult computer vision tasks, such as the detection of object categories (for example, the detection of faces of various gender, age, race, and pose, under various illumination and background conditions), researchers generally learn a classifier that can distinguish an image patch that contains the object of interest from all other image patches. Ensemble learning methods have been very successful in learning classifiers for object detection.

The task of object detection, however, poses new challenges for ensemble learning, which we will discuss in detail in Sect. 8.2. We summarize these challenges into three aspects: scale, speed, and asymmetry.

Various research contributions have been made to overcome these difficulties. In this chapter, we mainly focus on those methods that use the cascade classifier structure together with ensemble learning methods (e.g., AdaBoost). The cascade classifier structure for object detection was first proposed by Viola and Jones [41], who presented the first face detection system that could both run in real-time and achieve high detection accuracy. We will describe this work in Sect. 8.3, with ensemble learning methods being one of the key components in this system.
Various research efforts have been devoted to improve the learning speed of a cascade, which took several weeks in the original version of [41]. In Sect. 8.4, we present several methods that improve the training time by several orders of magnitudes, including a faster implementation of AdaBoost for cascade classifiers [45], an approximate weak classifier training method that is used to form the strong AdaBoost classifier [28], and Forward Feature Selection, an alternative to the AdaBoost learning method [45].

In Sect. 8.5, we present methods that specifically deal with the difficulties associated with the asymmetric learning problem inside a cascade. Two methods are described in detail: the asymmetric AdaBoost method from [40], and the Linear Asymmetric Classifier (LAC) method from [45].

We then move beyond the detection of upright and frontal faces into broader object detection domains in Sect. 8.6. We will, however, still pay attention to methods in the cascade and/or ensemble learning framework. Profile faces and rotated faces are also effectively detected using these techniques [17]. In addition, two methods for pedestrian detection are described: one for detection in still images [54], and the other for detection in low resolution surveillance videos that incorporates motion information [42]. Finally, we show that detection is not only useful for its own sake. It can, for example, be the cornerstone for a visual tracking system (i.e., tracking-by-detection).

Many other methods enable object detection and related tasks by applying novel ensemble learning algorithms, which will be given in Sect. 8.8 as bibliographical notes, after some discussions in Sect. 8.7.

### 8.2 Brute-Force Object Detection: Challenges

#### 8.2.1 The Brute-Force Search Strategy

When the object of interest has a fixed aspect ratio (e.g., in frontal face detection the height divided by width of all faces are roughly the same), the brute-force search strategy is the most widely used method for object detection.

The first step in the brute-force search approach is to train a classifier that can distinguish between the object of interest and all other image patches. A training dataset is constructed, which consists of positive examples (image patches of the target object) and negative examples (representative image patches from the background and all other objects). Since the object of interest has a fixed aspect ratio, image patches in the training set are normalized to the same size. The next step is then to train a classifier using such a training dataset. In this learning phase, ensemble learning is the most popular choice [41], although other classifiers (e.g., the Support Vector Machine) have also been applied [25].

During the detection phase, we scan all the image patches of a fixed size given by the training patches, by enumerating all possible locations within a testing image.
We usually start from the top-left corner, and apply the learned classifier to determine whether an object of interest exists in this position or not. We then move the scanning window and apply the classifier from left to right, and from top to bottom. That is, the classifier is applied to a regular grid of positions overlaid on the testing image. The step size of the grid can be varied in order to trade off between detection accuracy and running speed.

In order to detect an object bigger or smaller than the training patches, two strategies can be used. One approach is to resize the test images multiple times, so that the true object size equals the detector window size in one of the resized images (e.g., in [45]). It is also viable to resize the detector. For example, in [41] detectors for faces of different sizes are trained.

Postprocessing (or nonmaximum suppression) is the last step in brute-force object detection systems. During the search process, the classifier usually fires at multiple positions and scales around a true object of interest. Thus an object can have more than ten detected positions surrounding it. The postprocessing step merges nearby detections into a single output. In addition, if only a few (e.g., one or two) detections are found around a position, we may want to ignore such a detection. The postprocessing step thus can reduce the number of false detections (false positives), at the cost of potentially missing some target objects.

### 8.2.2 Challenges in Learning the Classifier

Properties that are particular to the learning task in object detection make the classifier learning step very challenging. We summarize such challenges into three aspects: scale, speed, and asymmetry. We will describe relevant machine learning solutions to these challenges in the following sections of this chapter.

#### 8.2.2.1 The Scale Challenge

The number of training examples needed for a learning task grows with the complexity of the problem. Object class detection is a task with high complexity: we need to detect the object under many variations such as pose, illumination, deformation, etc. As a consequence, the training set needs to be quite large, ranging from tens of thousands [25] to several billion [45]. Special considerations are needed for dealing with these large training sets.

#### 8.2.2.2 The Speed Challenge

A huge training set naturally leads to a long training time. For example, the original training method in [41] takes weeks for training a cascade classifier for face detection. It is thus necessary to greatly reduce the training time. However, the
testing speed is more important for object detection. Video rate (or even faster) object detection speed has attracted the attention of researchers and substantial progress has been made. Nowadays, real-time face detection is a must-have feature for even entry-level digital cameras.

### 8.2.2.3 The Asymmetry Challenge

Object detection involves a highly asymmetric learning task. The asymmetry property greatly contributes to the difficulty of classifier learning. Three asymmetries are summarized in [45]:

1. **Uneven class priors.** It is known in machine learning that problems with imbalanced training sets lead to poor accuracy in the minority class (the class with relatively few training examples) [16]. In object detection, the scanning grid generates millions of image patches, but few are the objects of interest. The negative class easily generates a huge set of training examples, but the object class usually has a limited set of training examples. We need to carefully deal with this asymmetry.

2. **Goal asymmetry.** Even a single false detection (false positive) in one testing image may be annoying in object detection and its applications. On one hand, since we are scanning millions of image patches in a medium-sized testing image, we are in essence requiring the classifier to have an extremely low false positive rate (e.g., $10^{-7}$). On the other hand, a high detection rate (e.g., 5% false negative/miss rate) is also required. The combination of these requirements and the big difference among these goal numbers make learning in object detection more challenging.

3. **Unequal complexity within the positive and negative classes.** As aforementioned, the positive (object) class is complex because of possible variations. However, the negative class is much more complex because this class contains everything else in the world except the object of interest. It is suggested in [45] that

   It is not hard to distinguish faces from cars. However, it is much harder to distinguish faces from all other objects.

### 8.3 The Cascade Face Detector

The cascade structured face detector by Viola and Jones [41] is the first system that achieves accurate frontal face detection in real time, with the help of three components: an image feature that can be computed quickly, an efficient classifier structure, and a novel application of the ensemble learning method (discrete AdaBoost in this case).
8.3.1 The Cascade Classifier Structure

The cascade classifier structure is mainly designed for high testing speed. We have discussed in Sect. 8.2 that a complex classifier is needed for object detection tasks, which also means that testing will be slow. This difficulty is alleviated by the cascade classifier structure [41]. A cascade consists of a sequence of classifiers with binary node classifiers $H_1, H_2, \ldots, H_n$, as illustrated in Fig. 8.1. An image patch is classified as the object of interest if it can pass tests in all the nodes. Since most background patches in a test image are filtered away by the early nodes, only a few image patches will fire all node classifiers (most of which will contain the object of interest). A cascade thus has very fast detection speed, as a consequence of the rarity of target objects.

Let us assume that the errors made by node classifiers are independent of each other. Furthermore, let us assume the cascade has 20 nodes and the node classifiers (refer to Fig. 8.1) have high detection rate $d_i = 99.9\%$ and false positive rate $f_i = 50\%$ for all $i$. The cascade classifier will be able to detect $\prod_{i=1}^{20} d_i = 98\%$ of the objects, and with a false positive rate $\prod_{i=1}^{20} f_i = 10^{-6}$. The cascade classifier structure is shown in Algorithm 1.
Note that only a limited number of negative training examples are used for training a node classifier. After $T$ node classifiers have been trained, the current cascade is applied to a huge set of negative image patches to find those “difficult” examples that are wrongly classified by all the existing nodes. These examples are added to the negative training set for training node classifier $T+1$. The database of bootstrapping negative examples $\mathcal{D}$ does not need to be explicitly stored. It can be generated online by applying the current cascade to images that do not contain any object of interest.

### 8.3.2 The Haar-like Features and the Integral Image

Another factor that makes the Viola–Jones face detector real-time is a set of simple Haar-like visual features. We illustrate one example Haar-like feature in Fig. 8.2. A Haar-like feature corresponds to a mask that has the same size as the training image patches. Elements in the mask can only take three different values: white pixels in the mask corresponding to the value $+1$, the black pixels with the value $-1$, and the gray pixels with the value $0$. One Haar-like feature will have a feature value that equals the dot product between the mask and an image patch. Four types of Haar-like features are proposed in [41].

The Haar-like features can be computed in a constant number of machine instructions, which make evaluation of the node classifiers $H_i$ very fast. As illustrated in Fig. 8.2, the feature value can be quickly computed if we can efficiently compute the sum of pixel values inside each rectangle with the same color (white or black).

Given any image $I(x, y)$, an integral image [41] is defined as an image $I'$ with the same size as $I$. The values in $I'$ are defined as:

$$I'(x, y) = \sum_{1 \leq i \leq x, 1 \leq j \leq y} I(x, y).$$  \hspace{1cm} (8.1)

Given $I'$, the sum of pixel values in the rectangle $ABCD$ (Fig. 8.2) is simply $I'(A) + I'(D) - I'(B) - I'(C)$. Thus, a constant number of machine instructions are needed to compute the sum of pixel values in a rectangle. The example feature
in Fig. 8.2 requires the sum inside four rectangles, which means that it can be computed in constant time with the help of $I'$. 

The integral image $I'$ for a testing image $I$ also needs to be computed in the testing phase. Fortunately, (8.1) can be computed very efficiently by two recurrence relationships [41]:

$$s(x, y) = s(x, y - 1) + I(x, y)$$

(8.2)

$$I'(x, y) = I'(x - 1, y) + s(x, y).$$

(8.3)

in which $s(x, y)$ is the sum of pixel values in the $x$th row till the $y$th column. Border values for $s$ and $I'$ can be initialized as $s(x, 0) = 0$ and $I'(0, y) = 0$. In short, only two summations are required to compute the integral image at every pixel position.

### 8.3.3 AdaBoost Feature Selection and Classification

It is obvious that one single Haar-like feature has weak discrimination capability, far below the requirement for object detection. It is inevitable then to choose (i.e., feature selection) and combine (i.e., classifier learning) a number of these weak features to form a strong classifier that will act as a node classifier $H_i$.

Feature selection, however, is itself a challenging task. Viola and Jones used training images with size $24 \times 24$, which leads to 45,396 Haar-like features [41]. The large number of features makes feature selection difficult in terms of both speed and accuracy considerations. In [41], Viola and Jones proposed a creative idea to use the discrete AdaBoost algorithm [31] to integrate feature selection seamlessly into the classifier learning process. Their method is shown in Algorithm 2.

The key idea in Algorithm 2 is to turn every Haar-like feature into a weak classifier. AdaBoost is then applied to select and combine from the pool of weak classifiers to form a strong classifier.\(^1\)

When equipped with a threshold value, a Haar-like feature can be easily turned into a decision stump (i.e., a decision tree with only one level). Formally, a weak classifier consists of a Haar-like feature (with corresponding mask $m_j$), a threshold $\theta_j$, and a parity $p_j$. The resulting weak classifier $h_j$ then classifies an example $x$ as:

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

(8.4)

\(^1\)After training the AdaBoost classifier (i.e., a node classifier in the cascade), one can adjust the threshold $\theta$ to meet the learning goal of a node classifier (e.g., a fixed detection rate or a fixed false positive rate.)
Algorithm 2 *NodeLearning* using AdaBoost (modified from Table 1 in [41])

1: {Input: a set of positive examples \( P \), and a set of negative examples \( N \).}

2: Initialize weights \( w_{1,i} = \frac{1}{2m}, \frac{1}{2l} \) for \( y_i = -1, 1 \) respectively, where \( m \) and \( l \) are the number of negative and positive examples respectively.

3: for \( i = 1 \) to \( T \) do

4: Normalize the weights \( w_i \) so that \( w_i \) is a probability distribution.

5: For each feature \( j \), train a weak classifier \( h_j \). The weighted error is \( \sum_i w_{i,i} | h_j(x_i) - y_i | \).

6: Choose the classifier, \( h_t \), to be the weak classifier with the lowest weighted error \( e_t \).

7: Update the weights as \( w_{i+1,i} = w_{i,i} \beta^{-e_i} \) in which \( \beta = \frac{e_i}{1-e_i} \), \( e_i = 0 \) if \( x_i \) is classified correctly by \( h_t \), and \( e_i = 1 \) if otherwise.

8: \( \alpha_t = \log \frac{1}{\beta_t} \).

9: end for

10: Output: a node classifier

\[
H(x) = \begin{cases} 
1 & \text{if } \sum_{i=1}^{T} \alpha_i h_j(x) \geq \theta, \\
-1 & \text{otherwise}
\end{cases}
\]

in which \( \theta \) is an adjustable parameter to control the trade off between detection rate and false positive rate of \( H \).

The parity \( p_j \) can take values in \( \{+1, -1\} \). This parameter determines which side of the threshold \( \theta_j \) should be classified as positive. We will describe an efficient method to learn the optimal value for \( p_j \) and \( \theta_j \) in Sect. 8.4.

Although a single Haar-like feature usually has a high error rate (e.g., between 40% and 50% in the face detection task [41]), the discrete AdaBoost algorithm in Algorithm 2 can boost multiple weak classifier into a strong one. Moreover, a cascade of these trained node classifiers can detect faces accurately in real time. The system in [41] detects faces at the speed of 15 frames per second on a slow 700 MHz Pentium III computer. It can detect 89.8% of all faces with only 0.5 false detections per testing image, when evaluated on the benchmark MIT+CMU frontal face detection dataset [30].

### 8.4 Improving Training Speed of a Cascade

It is reported in [41] that training a complete cascade requires weeks to finish. Thus, it is very important to improve the training speed. In this section, we only consider the *NodeLearning* part.
Algorithm 3 Training a weak classifier (modified from Algorithm 3 in [45])

1: {Input: training examples with labels \(\{x_i, y_i\}_{i=1}^{N}\) with weights \(\{w_i\}_{i=1}^{N}\), and a Haar-like feature with corresponding mask \(m\).}
2: Find feature values, \(v_1, \ldots, v_N\), where \(v_i = x_i^T m\).
3: Sort the feature values as \(v_{i_1}, \ldots, v_{i_N}\) where \((i_1, \ldots, i_N)\) is a permutation of \((1, \ldots, N)\), and satisfies that \(v_{i_1} \leq \cdots \leq v_{i_N}\).
4: \(\varepsilon \leftarrow \sum_{y_i = -1} w_{i,j}\).
5: for \(k = 1\) to \(N\) do
6: \(\text{if } y_{i_k} = -1\) then
7: \(\varepsilon \leftarrow \varepsilon - w_{i_k,j}, \varepsilon_{i,j} \leftarrow \varepsilon\).
8: else
9: \(\varepsilon \leftarrow \varepsilon + w_{i_k,j}, \varepsilon_{i,j} \leftarrow \varepsilon\).
10: end if
11: end for
12: \(k = \arg \min_{1 \leq i \leq N} \varepsilon_{i,j}, \tau = x_{i_k}^T m\).
13: Output: a weak classifier \(h(x) = \text{sgn}(x^T m - \tau)\).

8.4.1 Exact Weak Classifier Learning

Let us denote the number of iterations in Algorithm 2 as \(T\), the number of Haar-like features as \(M\), and the number of training examples as \(N\). The first step to accelerate is the line 5 of Algorithm 2. This line trains a weak classifier from a Haar-like feature, and will be called upon \(MT\) times in Algorithm 2. Training weak classifiers needs to be done very efficient because \(MT\) is on the order of millions. Algorithm 3 gives a method for accelerating the training process, which is taken from [45].

Suppose there are \(N\) training examples, and these examples have feature values \(v_1, \ldots, v_N\) for a Haar-like feature. Algorithm 3 first sorts the feature values into \(v_{i_1}, \ldots, v_{i_N}\), where \((i_1, \ldots, i_N)\) is a permutation of \((1, \ldots, N)\), and \(v_{i_1} \leq \cdots \leq v_{i_N}\). If \(v_{i_k} \leq \theta_1, \theta_2 \leq v_{i,k+1}\) is true for some integer \(k\) and two different thresholds \(\theta_1\) and \(\theta_2\), setting the threshold of this Haar-like feature to \(\tau = \theta_1\) will have the same accuracy on the training set as that of \(\tau = \theta_2\). Thus, we only need to check \(N + 1\) possible values for finding the optimal \(\tau\). In addition, a sequential update can compute the weighted error rate of different threshold values in \(O(N)\) steps. Note that Algorithm 3 only checks the parity +1. It is easy to find the optimal weak classifier for both parity values using the idea of Algorithm 3. The complexity of Algorithm 3 is then \(O(N \log N)\), dominated by the complexity of sorting feature values to get the permutation. Consequently, the complexity of Algorithm 2 is \(O(NMT \log N)\).

However, one does not need to recompute the permutation \((i_1, \ldots, i_N)\) at every iteration inside Algorithm 2 [45]. In AdaBoost learning, weak classifiers need to be re-trained at every iteration because of the updated weights \(w_{i,j}\). However, the permutations remain constant throughout Algorithm 2, because they do not depend on \(w_{i,j}\). By creating a table to precompute and store the permutations for all
Haar-like features ($O(NM \log N)$), training a weak classifier becomes $O(N)$, and the entire AdaBoost complexity can be reduced to $O(NM(T + \log N))$. In practice this space-for-time strategy leads to two orders of magnitudes speedup.

### 8.4.2 Approximate Weak Classifier Learning

Algorithm 3 seeks to find the optimal threshold that achieves minimum weighted error on the training set. However, the power of AdaBoost resides with the combination of multiple weak classifiers. Theoretically, we only need to guarantee that in each iteration the selected weak classifier $h_t$ has a weighted error rate that is smaller than 0.5. Faster algorithm can be achieved if some approximations are allowed in the weak classifier training step.

One such approximation algorithm was proposed by Pham and Cham [28]. Instead of directly using the raw training examples $x_i$ and their associated weights $w_{t,i}$, Pham and Cham used statistics of the training set to find the weak classifiers’ parameters. Specifically, they assume that feature values for the positive examples follow a normal distribution $N(\mu_+, \sigma_+^2)$. $\mu_+$ is the average weighted feature value for all positive training examples, given a specific Haar-like feature with mask $m$:

$$\mu_+ = \sum_{y_i=+1} w_{t,i} m^T x_i, \quad (8.5)$$

in which $m$ is the mask corresponding to the Haar-like feature; $\sigma_+$ is the standard deviation of the feature value for positive training examples. Similarly, negative examples’ feature values are also assumed to follow $N(\mu_-, \sigma_-^2)$. Closed-form and efficient solution exists for finding the optimal separating plane for two one-dimensional normal distributions [11]. We only need to efficiently compute the values $(\mu_+, \sigma_+, \mu_-, \sigma_-)$ when the weights $w_{t,i}$ are updated, in which the integral image once again helps.

An image $x$ and its integral image $x'$ are linked together by (8.1), which is obviously a linear transformation. Thus there exists a matrix $B$ such that $x' = Bx$. The matrix $B$ is constant and invertible, and encodes the linear transformation between $x$ and $x'$. The example $x$ can be expressed as $x = B^{-1}x'$. One can in turn use the integral image to compute $\mu_+$ as [28]:

$$\mu_+ = \sum_{y_i=+1} w_{t,i} m^T B^{-1} x_i' = m^T B^{-1} \left( \sum_{y_i=+1} w_{t,i} x_i' \right). \quad (8.6)$$

Two facts make (8.6) easy to compute. First, the term $\sum_{y_i=+1} w_{t,i} x_i'$ can be precomputed and stored whenever $w_{t,i}$ are updated. Second, the vector $m^T B^{-1}$ is a sparse vector which usually contains less than 10 nonzero entries.
Note that \( \sum_{y_i = +1} w_{t,i} x'_i \) is the weighted average of \( x' \) (integral version of training patches). Let \( \Sigma_{x'} \) be the (weighted) covariance matrix of \( x' \), then we can compute \( \sigma^2_+ \) as\(^2\):

\[
\sigma^2_+ = (m^T B^{-1}) \Sigma_{x'} (m^T B^{-1})^T.
\]

(8.7)

The values \( \mu_- \) and \( \sigma_- \) can be computed using the same trick.

This method has two attractive characteristics. First, the complexity of training a weak classifier for a given feature is independent of \( N \), the number of training examples. When \( N \) is big (which is usually the case in object detection), this approximation method is more efficient than Algorithm 3. And its speed advantage increases when \( N \) gets bigger. Second, unlike Algorithm 3, which requires additional storage for saving the permutation vectors, this approximation method has a smaller memory footprint. As a direct consequence, more Haar-like features can be used in the training process (and the use of more Haar-like features usually implies higher accuracies).

Empirically, in [28] Pham and Cham reported the training of a cascade classifier using 295,920 Haar-like features (with more feature types than those appearing in [41]) for face detection. The training process finished in 5 h and 30 min. Using Algorithm 3 with 40,000 features, it took 13 h and 20 min to train a cascade, using the same training set and running on the same hardware. With the same number of Haar-like features, both methods achieve very similar detection accuracies, which means that the approximate weak classifier training part does not hinder the final classifier’s accuracy. Having the ability to deal with more features, the approximation method reduces the number of false detections at a given detection recall rate in comparison to Algorithm 3.

### 8.4.3 FFS: Alternative Feature Selection

The AdaBoost-based Algorithm 2 combines the selection of discriminative Haar-like features and the learning of a node classifier into an integrated framework. However, Wu et al. showed that these two components are not necessarily tied together. Alternative feature selection and node classifier learning methods can be applied sequentially, with the benefit of reduced training time and better detection performance.

Forward feature selection (FFS) [44] is a frequently used greedy feature selection method. It can be used effectively to select Haar-like features [45], whose algorithmic details are presented in Algorithm 4.

The first step of Algorithm 4 is to train the weak classifiers for all Haar-like features \( O(NM \log N) \) using Algorithm 3). The same space-for-time strategy is

\(^2\)Special care is required for computing \( \Sigma_{x'} \) efficiently. However, we omit these details. The readers may refer to Sect. 3.2 of [28] for more information.
Algorithm 4 FFS feature selection (modified from Algorithm 2 in [45])

1: {Input: a training set \{x_i, y_i\}_{i=1}^N, a set of Haar-like features \{h_i\}_{i=1}^M, where \(N\) and \(M\) are the number of training examples and Haar-like features, respectively.}

2: for \(i = 1\) to \(M\) do
3: Choose appropriate parity and threshold for a Haar-like feature \(h_i\), such that \(h_i\) has smallest error on the training set using Algorithm 3.
4: end for
5: \(V_{ij}\) such that \(V_{ij} = h_i(x_j), 1 \leq i \leq M, 1 \leq j \leq N\).
6: \(S \leftarrow \emptyset, v \leftarrow \theta_{1 \times N}, \) where \(\theta_{1 \times N}\) is a row vector filled by zeros.
7: for \(t = 1\) to \(T\) do
8: for \(i = 1\) to \(M\) do
9: \(S' \leftarrow S \cup h_i, v' \leftarrow v + V_{i,:}, \) where \(V_{i,:}\) is the \(i\)th row of \(V\).
10: \(\{H'(x) = \text{sgn}(\sum_{h \in S'} h(x) - \theta)\}\) is the classifier associated with \(S'\), and we can compute its value using \(H'(x) = \text{sgn}(v' - \theta)\).
11: \(\epsilon_i\) the error rate of \(H'\) with the chosen \(\theta\) value.
12: \(k \leftarrow \arg \min_{1 \leq i \leq M} \epsilon_i\).
13: end for
14: \(S \leftarrow S \cup h_k, v \leftarrow v + V_{k,:}\).
15: end for
16: Output: a node classifier 
\[H(x) = \text{sgn}\left(\sum_{h \in S} h(x) - \theta\right)\].

used: precompute and save classification results of all such classifiers into a table \(V\).

One noticeable difference between Algorithms 2 and 4 is that in FFS these weak classifiers are trained without using the weights. Thus, there is no need to update these weak classifiers.

Both Algorithms 2 and 4 are “wrapper” feature selection methods, in the sense that the effectiveness of a selected subset of features is evaluated by a classifier trained from such a subset. Instead of using weighted voting as that in AdaBoost, FFS uses a simple vote strategy: every weak classifier has the same weight. In every iteration, the FFS algorithm examines every Haar-like feature, temporarily adds it to the selected feature subset, and evaluates the classification accuracy of the updated set of features. The Haar-like feature that leads to the minimum classification error rate is chosen and permanently added to the selected feature subset.

However, although both algorithms are greedy in nature, they solve different optimization objectives. In AdaBoost, a node classifier \(H(x)\) is implicitly minimizing the cost function
\[\sum_{i=1}^N \exp(-y_i H(x_i)).\] (8.8)

While in FFS we are explicitly finding the feature that leads to a node classifier with the smallest error rate in the training set.

The classification result of a subset of features can be updated very efficiently using the stored table \(V\). In order to find the optimal threshold value, the trick in
Algorithm 3 is also applicable. Note that at iteration $t$, the number of votes are always integers within the range $[0, t]$, which means that we only need to consider $\theta$ in this range. Two histograms (one for positive examples and one for negative examples) can be built: the cell $i$ contains the number of positive or negative examples that have $i$ votes, respectively. Using both histograms, the error rate at $\theta = 0, \ldots, t$ can be sequentially updated efficiently.

The complexity of Algorithm 4 is $O(NM(T + \log N))$, same as that of Algorithm 2. However, in practice FFS has a smaller constant factor than AdaBoost. It was reported in [45] that the fast AdaBoost implementation usually uses 2.5–3.5 times of the training time of FFS, and the original AdaBoost implementation in [41] needs 50–150 times of that of FFS.

With faster training speed, FFS trains face detectors that have similar results as AdaBoost in terms of both accuracy and testing speed. FFS can be used as an alternative method for AdaBoost in training node classifiers in a cascade object detection system.

8.5 Asymmetric Learning in Cascades

8.5.1 Goal Asymmetry and Asymmetric AdaBoost

As was discussed earlier in Sect. 8.2, at least three major challenges exist in object detection: scale, speed, and asymmetry. Most of these challenges are addressed in the algorithms we have discussed so far. With the cascade classifier structure, testing speed can be improved to video rate or even higher. The bootstrap step in Algorithm 1 can effectively deal with billions of negative training examples. The fast AdaBoost method and the exact weak classifier learning algorithm reduce training time of a complete cascade from weeks to hours, which is further improved by the distribution-based approximation method and the alternative feature selection method (Algorithm 4).

As to the asymmetries, the cascade classifier structure also handles the “uneven class priors” asymmetry. We can choose to use the same number of positive and negative training instances when learning every node classifier. The bootstrap process also implicitly deals with “unequal complexity within the positive and negative classes.” Since at every node we only use a small subset of negative examples, the complexity of node negative training set is limited.

We are left with the “goal asymmetry.” For the complete cascade, we require high detection rate (e.g., 95%) and extremely low false positive rate (e.g., $10^{-7}$ [41]. In a node classifier, we try to achieve the asymmetric node learning goal [45]:

for every node, design a classifier with very high (e.g., 99.9%) detection rate and only moderate (e.g., 50%) false positive rate.

This special requirement demands special learning algorithms. Asymmetric AdaBoost (AsymBoost) was an attempt by Viola and Jones for solving this
The idea of AsymBoost is to emphasize false negatives (i.e., classifying faces as nonfaces) more than false positives (i.e., classifying nonfaces as faces). Both false positive and false negative have the same loss in Algorithm 2. This symmetric loss is replaced by an asymmetric loss function (assuming false negatives are $k$ times more important than false positives):

$$A_{\text{Loss}(i)} = \begin{cases} \sqrt{k} & \text{if } y_i = +1, \text{ and } H(x_i) = -1, \\ \frac{1}{\sqrt{k}} & \text{if } y_i = -1, \text{ and } H(x_i) = +1, \\ 0 & \text{otherwise} \end{cases}$$

(8.9)

This new loss function can be easily incorporated into Algorithm 2, by pre-weighting training examples using $\exp\left(\frac{y_i \log \sqrt{k}}{C}\right)$. This strategy, however, is unsuccessful because AdaBoost will quickly absorb this artificial difference in initialization [40]. Instead, Viola and Jones amortize this asymmetric cost into every iteration of AdaBoost. In a node classifier with $T$ AdaBoost iterations, $\exp\left(\frac{1}{T} y_i \log \sqrt{k}\right)$ is multiplied to the example weights $w_{t,i}$ for $t = 1, 2, \ldots, T$, followed by a normalization procedure to make the new weights a distribution.

AsymBoost enforces a higher cost for missing faces ($k > 1$) than false detections in node classifiers. When comparing complete cascades, a cascade trained using AsymBoost usually achieves 1–2% higher detection rate with the same number of false detections on the MIT+CMU benchmark face detection dataset [40].

### 8.5.2 Linear Asymmetric Classifier

Another attempt to address the goal asymmetry is LAC by Wu et al. [45]. LAC trains node classifiers to directly optimize the asymmetric node learning goal: very high (e.g., 99.9%) detection rate and only moderate (e.g., 50%) false positive rate.

#### 8.5.2.1 LAC Formulation

LAC does not perform feature selection. Instead, it assumes that a subset of discriminative features have been selected by other methods (e.g., AdaBoost, FFS, AsymBoost, or any other method). Furthermore, LAC assumes that weak classifiers corresponding to these selected features have also been trained. Given an example $x$, the classification results of weak classifiers are the input to LAC. In other words, the input to LAC are binary vector ($+1$ or $-1$) in $\mathbb{R}^d$ if $d$ Haar-like features are selected.

For simplicity in the presentation, we use $x$ to represent this binary vector for the same training example $x$ too. These two different meanings should be easily
distinguishable from the context. In this section, we use \( x \) and \( y \) to denote positive and negative training examples, respectively. The class labels (+1 for \( x \) and -1 for \( y \)) are implied in the symbols and thus omitted. An example is denoted as \( z \) if its label is unknown, following the notation of [45].

LAC expresses the asymmetric node learning goal and tries to directly optimize this goal for a linear classifier:

\[
\begin{align*}
\max_{a \neq 0, b} & \Pr_{x \sim (\tilde{x}, \Sigma_x)} \{ a^T x \geq b \} \\
\text{s.t.} & \Pr_{y \sim (\tilde{y}, \Sigma_y)} \{ a^T y \leq b \} = \beta.
\end{align*}
\]

(8.10)

Only the first- and second-order moments are used in the formulation of LAC: \( x \sim (\tilde{x}, \Sigma_x) \) denotes that \( x \) is drawn from a distribution with mean \( \tilde{x} \) and covariance matrix \( \Sigma_x \). The distribution of \( x \), however, is not necessarily Gaussian. Similarly, negative examples are modeled by \( \tilde{y} \) and \( \Sigma_y \). LAC only considers linear classifiers \( H = (a, b) \):

\[
H(z) = \begin{cases} 
+1 & \text{if } a^T z \geq b \\
-1 & \text{if } a^T z < b.
\end{cases}
\]

(8.11)

The constraint in (8.10) fixes the false positive rate to \( \beta \) (and \( \beta = 0.5 \) when learning a node classifier). The objective in (8.10) is to maximize the detection rate. Thus, (8.10) is a literal translation of the asymmetric node learning goal, under the distribution assumption of training examples.

8.5.2.2 LAC Solution

Let \( x_a \) denote the standardized version of \( a^T x \) (\( x \) projected onto the direction of \( a \)), i.e.,

\[
x_a = \frac{a^T (x - \tilde{x})}{\sqrt{a^T \Sigma_x a}}.
\]

(8.12)

\( y_a \) can be defined similarly for negative examples. Equation (8.10) is converted to an unconstrained optimization problem as:

\[
\min_{a \neq 0} \Psi_{x,a} \left( \frac{a^T (\tilde{y} - \tilde{x}) + \Psi_{y,a}^{-1}(\beta) \sqrt{a^T \Sigma_y a}}{\sqrt{a^T \Sigma_x a}} \right),
\]

(8.13)

in which \( \Psi_{x,a} \) (\( \Psi_{y,a} \)) denotes the cumulative distribution function (c.d.f.) of \( x_a \) (\( y_a \)), and \( \Psi_{y,a}^{-1} \) is the inverse function of \( \Psi_{y,a} \). This is, however, a difficult optimization problem because we do not know the properties of \( \Psi_{x,a} \) and \( \Psi_{y,a}^{-1} \).

Two assumptions are made in [45] to simplify (8.13). First, \( a^T x \) is assumed to follow a scalar normal distribution. Second, the median value of the distribution \( y_a \) is close to its mean (so that we have \( \Psi_{y,a}^{-1}(\beta) \approx 0 \) when \( \beta = 0.5 \)). These assumptions
align well with the reality in object detection, as shown in Fig. 8.3. \( \mathbf{a}^T \mathbf{x} \) fits closely to a normal distribution in the normal probability plot, a visual method to test the normality of a distribution. \( \mathbf{a}^T \mathbf{y} \) fits almost perfectly to a normal distribution, which implies that its mean and median are indeed the same.

Under these assumptions, for \( \beta = 0.5 \) (8.13) can be further approximated by

\[
\max_{\mathbf{a} \neq 0} \frac{\mathbf{a}^T (\bar{x} - \bar{y})}{\sqrt{\mathbf{a}^T \Sigma \mathbf{a}}} \tag{8.14}
\]

which has closed-form solutions:

\[
\mathbf{a}^* = \Sigma_x^{-1} (\bar{x} - \bar{y}) \tag{8.15}
\]
\[
\mathbf{b}^* = \mathbf{a}^T \bar{y} \tag{8.16}
\]

When \( \Sigma_x \) is positive semi-definite, \( \Sigma_x + \lambda I \) can be used to replace \( \Sigma_x \), where \( \lambda \) is a small positive number. A summary of applying LAC to train a node classifier in object detection is shown in Algorithm 5.

**Algorithm 5** LAC as *NodeLearning* (modified from Algorithm 4 in [45])

1: \{Input: a training set composed of positive examples \( \{\mathbf{x}_i\}_{i=1}^{n_x} \) and negative examples \( \{\mathbf{y}_i\}_{i=1}^{n_y} \), a set of Haar-like features, and a feature selection method \( \mathcal{F} \).\}

2: Select \( T \) weak classifiers \( \mathbf{h} = (h_1, h_2, \ldots, h_T) \) using \( \mathcal{F} \), where \( h_i(\mathbf{z}) = \text{sgn}(\mathbf{z}^T \mathbf{m}_i - \tau_i) \).

3: For each training example, build a feature vector \( \mathbf{h}(\mathbf{z}) = (h_1(\mathbf{z}), h_2(\mathbf{z}), \ldots, h_T(\mathbf{z})) \).

4:

\[
\bar{x} = \frac{\sum_{i=1}^{n_x} h(x_i)}{n_x}, \quad \bar{y} = \frac{\sum_{i=1}^{n_y} h(y_i)}{n_y},
\]
\[
\Sigma_x = \frac{\sum_{i=1}^{n_x} (h(x_i) - \bar{x})(h(x_i) - \bar{x})^T}{n_x},
\]
\[
\Sigma_y = \frac{\sum_{i=1}^{n_y} (h(y_i) - \bar{y})(h(y_i) - \bar{y})^T}{n_y}.
\]

5:

\[
\mathbf{a} = \Sigma_x^{-1} (\bar{x} - \bar{y}), \quad \mathbf{b} = \mathbf{a}^T \bar{y}.
\]

6: Output: a node classifier

\[
H(\mathbf{z}) = \text{sgn} \left( \sum_{i=1}^{T} \mathbf{a}_i \mathbf{h}_i(\mathbf{z}) - \mathbf{b} \right) = \text{sgn} (\mathbf{a}^T \mathbf{h}(\mathbf{z}) - \mathbf{b}) .
\]
Fig. 8.3 Normality test for $\mathbf{a}^T \mathbf{x}$ and $\mathbf{a}^T \mathbf{y}$. $\mathbf{a}$ is drawn from the uniform distribution $[0, 1]^T$. Part (a) shows overlapped results for 10 different $\mathbf{a}$’s. From Wu et al. [45], © 2008 IEEE, with permission. (a) $\mathbf{a}^T \mathbf{x}$. (b) $\mathbf{a}^T \mathbf{y}$
In fact, the simplified LAC solution is very similar to FDA (Fisher Discriminant Analysis). FDA can also be used in place of LAC in Algorithm 5, using the following equations:

$$a^* = \left(\Sigma_x + \Sigma_y\right)^{-1}(\bar{x} - \bar{y}),$$

$$b^* = a^T\bar{y}.\quad (8.17)$$

Empirical results from [45] show that both LAC and FDA can effectively deal with the asymmetric node learning goal in the node classifiers. On the MIT-CMU benchmark face detection dataset, both AdaBoost+LAC and AdaBoost+FDA have higher detection rates than that of AsymBoost, when the number of false detections are the same.

8.6 Beyond Frontal Faces

So far we have used frontal face detection as the example application to introduce various methods. Ensemble learning methods and the cascade classifier structure, of course, are useful not only for detecting frontal faces. In this section, we will briefly introduce the detection of objects beyond frontal faces. We will mainly focus on profile and rotated faces, pedestrians, and tracking.

8.6.1 Faces in Nonfrontal, Nonupright Poses

Many variations exist in the human head pose. The head can have out-of-plane rotations, which generates left and right profile faces. One face image can also be rotated using image processing softwares, which can generate in-plane rotated faces. A common strategy to deal with such additional complexity is to divide face poses into different “views” according to their in-plane and out-of-plane rotation angles. A cascade classifier can be trained to handle a single view.

It is, however, not easy to properly separate different face poses into views. The view structure must cover all possible head poses in consideration, and must also be efficient during testing time. One such multiview structure was proposed by Huang et al. [17], which is shown in Fig. 8.4.

Faces are organized into a tree structure, which can detect faces with out-of-plane rotation angles ranging from $-90^\circ$ to $90^\circ$ (i.e., from left profile to right profile), and all in-plane rotation angles. The root of the tree include all face poses, which are divided into three level 2 nodes according to the out-of-plane rotation: left, frontal, and right. Left (right) profile faces are further divided into two nodes according to the rotation angle. Thus, there are 5 nodes in the level 3 in total. Finally, every level
3 node is divided into three nodes in level 4, corresponding to in-plane rotation angle $-45^\circ$, $0^\circ$, and $45^\circ$, respectively. Every level 4 node then detects faces in one specific pose.

One design choice by Huang et al. is to allow multiple nodes at the same level in the classifier tree to be active simultaneously. For example, it is reasonable to activate both the second and the third node in the level 2 for a right profile face with a small out-of-plane rotation angle. This choice increases the possibility that a face is detected. But, it also requires a new kind of node classifier: the node classifier must be a multiple class classifier that allow multiple labels (i.e., next level nodes) to be predicted simultaneously. The challenges that are laid out in Sect. 8.2 are still to be solved by the new node classifier. A Vector Boosting algorithm was proposed by Huang et al. [17] as the new node classifier.

### 8.6.2 Pedestrians

Pedestrian detection is another area where the cascade classifier structure and ensemble learning have been successful.

---

3In addition to the level 4 nodes shown in Fig. 8.4, Huang et al. rotate their features (called Granule features) by $90^\circ$, $180^\circ$, and $-90^\circ$ for the level 4 nodes. This strategy effectively covers the entire $360^\circ$ range in-plane rotation.
8.6.2.1 Pedestrian Detection in Still Images

A system that is very similar to the face detection system was used by Zhu et al. [54] to detect pedestrians in still images with real-time speed. Pedestrians, however, exhibit different characteristics than faces. The simple Haar-like features (and their corresponding simple decision stump weak classifiers) are no longer discriminative enough for pedestrian detection, even with the help of AdaBoost.

Instead, the HOG (Histogram of Oriented Gradients) feature [10] was used in [54]. Within a $64 \times 128$ image patch which is typical for pedestrians, different HOG features can be extracted from 5,031 rectangles with various sizes. They constitute the features to be used in an AdaBoost algorithm. Similar to the Haar-like features, a weak classifier is trained for every HOG feature. A HOG feature is 36 dimensional, which automatically means that the decision stump weak learner is not applicable any more. The linear SVM learner (i.e., SVM using the dot-product kernel) is used to train weak classifiers.

HOG features and SVM classifiers do not enjoy the same hyper-speed as Haar-like features and decision stumps, in both the training and the testing phases. Two approaches were used to accelerate pedestrian detection in [54]. First, instead of training all 5,031 weak classifiers, 250 ($\approx 5\%$) HOG features are randomly sampled and their corresponding linear SVM are trained. Second, the “integral histogram” data structure is used to compute HOG features. The HOG feature used in [54] is a 36 dimensional histogram, and the integral histogram can compute HOG features efficiently, similar to the integral image data structure for the Haar-like features.

Compared to use Haar-like feature directly for pedestrian detection, HOG features can reduce the number of false positives by several orders of magnitudes at the same detection rate. In terms of testing speed, HOG features take about 2–4 times the time of that of Haar-like features, which can still be managed to be real-time with modern computers.

8.6.2.2 Pedestrian Detection in Videos

The pedestrians in [54] are of size $64 \times 128$, which is usually larger than pedestrian size in many application domains, e.g., surveillance. Viola et al. proposed to use motion information for pedestrian detection in surveillance videos [42].

For two consecutive video frame $I_t$ and $I_{t+1}$, five images can be defined in order to capture motion information at time $t$:

\[
\Delta = \text{abs}(I_t - I_{t+1})
\]

\[
U = \text{abs}(I_t - I_{t+1} \uparrow)
\]

\[
L = \text{abs}(I_t - I_{t+1} \downarrow)
\]

\[
R = \text{abs}(I_t - I_{t+1} \rightarrow)
\]

\[
D = \text{abs}(I_t - I_{t+1} \leftarrow)
\]
Algorithm 6 Ensemble tracking (modified from Algorithm 1 in [2])

1: {Input: video frames $I_1, \ldots, I_n$, and rectangle $r_1$ in $I_1$ that contains the object.}
2: Initialization: Train $T$ weak classifiers and add them to the ensemble.
3: for each subsequent frame $I_j, j > 1$ do
4: Test all pixels in $I_j$ using the current ensemble strong classifier and create a confidence map $L_j$.
5: Run mean shift on the confidence map $L_j$ and report new object rectangle $r_j$.
6: Label pixels inside rectangle $r_j$ as object and all those outside it as background.
7: Keep $K$ best weak classifier.
8: Train new $T - K$ weak classifiers on frame $I_j$ and add them to the ensemble.
9: end for
10: Output: Rectangles $r_2, \ldots, r_n$.

in which the operators $\{\uparrow, \leftarrow, \rightarrow, \downarrow\}$ shift an image by one pixel in the corresponding direction. An extended set of Haar-like features can be applied to one of these five images or the difference between $\Delta$ and one of the other four images to generate motion features. Motion features, together with the appearance features from $I_t$, form the new feature set for pedestrian detection. The same integral image trick, the AdaBoost ensemble learning method, and the cascade classifier structure can all be used in this new context.

8.6.3 Tracking

Tracking can also benefit from ensemble learning techniques, sometimes dubbed “tracking-by-detection.” In a tracking by detection framework, an object detector (e.g., the pedestrian detector) is continuously applied to every frame of a video. Detection results of consecutive frames are then registered across frames to form a reliable tracking result. In this section, we describe the ensemble tracking approach [2] by S. Avidan, which is shown in Algorithm 6.

AdaBoost is used to train an ensemble classifier that distinguish the target object from the background, and the classifier is continuously updated throughout the tracking process. Initially, positive and negative examples are extracted from the user labels ($r_1$ as positive and else as negative) from the first frame $I_1$. The ensemble classifier is then used to classify pixels in the next frame to form a confidence map about the possibility that a pixel belongs to the object or the background. The mean shift mode seeking algorithm [8] is used to find the object from the confidence map. The AdaBoost classifier needs to be updated by adding new weak classifiers using the newly detected object and the background in the new frame.
8.7 Discussions

We have described several object detection methods and applications, centered around the cascade classifier structure and ensemble learning methods (especially AdaBoost) in this chapter. In many applications (for example, face detection, pedestrian detection, and tracking), real-time detection speed, and detection accuracy suitable for practical usage have been achieved.

There are, however, many open questions remain in the object detection task. At least four factors still prevent the methods we discussed above from being applied to detect many other objects: training data, training speed, visual features, and multiclass learning.

- Both face and pedestrian detection require thousands of training image patches from the target object class, which are gathered through the laborious and error-prone human-guided data collection process. A human being need to manually crop the object of interest from the background clutter, and transform the cropped image patch to appropriate size. Similarly, a large set of images that do not contain any object of interest needs to be collected and verified. It is necessary to design new algorithms that only require few positive training images, and do not need negative training images.
- Training a cascade (or ensemble classifier) using a large feature set is time consuming. Although modern algorithms have reduced the training time to a few hours, it is still too long in many applications, e.g., training an object model for image retrieval. In fact, training time is closely related to the training set size. Ultimately we are aiming at an accurate detector that is trained with few examples and within seconds.
- As already seen in this chapter, different feature sets have been used in different tasks [41, 42, 45, 54]. The Haar-like features, although contributing to super fast detection systems, are usually not discriminative enough in the detection of objects beyond frontal faces. It is attractive to obtain a feature set that is capable for detecting many objects, and have an efficient evaluation strategy.
- We have focused on binary classification in this chapter: the object of interest versus the background clutter. However, object detection is a natural multiclass problem because we are usually interested in more than one object. A detector structure that can detect multiple object categories is both desirable and challenging, especially when we are interested in a large number of objects.

8.8 Bibliographical and Historical Remarks

Face detection based on machine learning methods have long been studies, dated back to at least early 1990s. Principal Component Analysis (PCA) was used in early attempts by Turk and Pentland [39], and Moghaddam and Pentland [23]. Sung and Poggio [35] used mixtures of Gaussians to model both faces and nonfaces.
The idea to bootstrap negative examples was also used in [35]. Various other machine learning methods have been used too. Osuna et al. used the Support Vector Machine to face detection in [25]. Yang et al. applied the SNoW learning architecture in [52]. Neural networks were used by Rowley et al. to provide accurate face detection (both frontal and rotated) [30]. Schneiderman and Kanade [32] used Naive Bayes to pool statistics of various image measurements into accurate detectors that detected faces (both frontal and profile) and cars. A survey of early face detection methods can be found in [51].

The cascade detector by Viola and Jones [41] is the first real-time frontal face detector. The cascade classifier structure is a coarse-to-fine search strategy that was previously used by Fleuret and Geman [13], and Amit et al. [1]. Sequential classifier rejections have also been used before, by Baker and Nayar [4], and Elad et al. [12]. The Haar-like features have been used before, by Papageorgiou et al. [27] for object detection. The integral image data structure that accelerates evaluation of Haar-like features was proposed by F. Crow [9]. Both the Haar-like features and the AdaBoost node classifier (which selects features and combines them) were used by Tieu and Viola [36] for image retrieval.

A large number of research efforts focus on improving the cascade framework in various aspects. Training speed of a cascade were greatly enhanced using a fast AdaBoost implementation [45] by Wu et al., using an alternative node learning method [47] by Wu et al., and using an approximate weak classifier training method based on statistical properties by Pham and Cham [28]. A similar idea was used by Avidan and Butman [3].

Features beyond the simple Haar-like features in [41] have been proposed, e.g., in [19, 28, 42, 45]. A type of features called Granule features was proposed by Huang et al. [17]. HOG features were used in [54]. A modified Census Transform feature was used by Froba and Ernst [14]. Local Binary Patterns (LBP) was used by Zhang et al. [53].

Various weak classifiers that are more complex than the decision stump are also helpful. Histogram of feature values was used in [20] by Liu and Shum. Decision trees with more than one internal node was used by Lienhart et al. [19] and Brubaker et al. [7].

Many variants of the boosting algorithm have been used to replace discrete AdaBoost. For example, an empirical study by Lienhart et al. [19] suggested using real AdaBoost and gentle AdaBoost. Boosting variants that deal with asymmetries have also been shown to improve object detection. Viola and Jones used an amortized version of asymmetric AdaBoost [40]. Wu et al. proposed LAC [48]. Cost-sensitive boosting algorithms can also improve the node classifiers [22]. Column generation was used to train boosting classifiers (LACBoost and FisherBoost) by Shen et al. in [33].

Alternative feature selection methods can be used to select features and train node classifiers in a cascade. Forward Feature Selection [47] was used by Wu et al. to form a node classifier. Floating search incorporated into boosting helped eliminate wrong selection, which was shown by Li and Zhang [18]. Sparse linear discriminant analysis was used by Paisitkriangkrai et al. to choose features [26].
Improvements are also made to the cascade structure. Xiao et al. proposed boosting chain to make use of previous trained node classifiers [50]. Cascade with many exits (e.g., one feature is a node) have been shown to improve both detection speed and accuracy by Bourdev and Brandt [5], Pham et al. [29], and Xiao et al. [49].

Choosing optimal operating points in the ROC curve of node classifiers were studied by Sun et al. [34]. A two-point algorithm was proposed by Brubaker et al. [7] to further improve the cascade.

Faces at all poses (including rotation in- and out-of plane) can be detected with improvements to the cascade architectures. Tree structures for detecting multiview faces were proposed by Li and Zhang [18], and Huang et al. [17]. A Vector Boosting algorithm was also proposed in [17].

Cascade and ensemble learning methods achieved a trade off between detection speed and accuracy for pedestrian detection [54]. However, it is worth noting that there exist other machine learning methods that have faster speed [46] or detection accuracy [43]. LAC from [48] was used in pedestrian detection by Mu et al. [24].

Tracking-by-detection is a successful application of object detectors. The Ensemble tracking method by S. Avidan [2] is an example. Tracking-by-detection usually requires online learning. Online boosting was used in [15] by Grabner and Bischof. Boosted particle filter was used by Lu et al. in [21]. A detector confidence particle filter was used by Breitenstein in [6].

Multiclass classification for object detection was studied in [38], using a probabilistic boosting tree. A boosting framework was proposed in [37], which can share features among different object categories.

References