Chapter 3
Traffic Sign Detection

Abstract To detect and classify objects contained in real images, acquired in unconstrained environments, is a challenging problem in computer vision, which complexity makes unfeasible the design of handcrafted solutions. In this chapter, the object detection problem is introduced, highlighting the main issues and challenges, and providing a basic introduction to the main concepts. Once the problem is formulated, a feature based approach is adopted for traffic sign detection, introducing the basic concepts of the machine learning framework and some bio-inspired features. Learning algorithms are explained in order to obtain good detectors using a rich description of traffic sign instances. Using the context of classical windowing detection strategies, this chapter introduces an evolutionary approach to feature selection which allows building detectors using feature sets with large cardinalities.

Keywords Traffic sign detection · Haar-like features · Integral image · Adaboost detection · Cascade of classifiers · Evolutionary computation

The detection and classification of objects in images that have been acquired in unconstrained environments is a challenging problem, because objects can appear in different poses and lighting conditions, and against different backgrounds and clutter. This variation in object appearance makes the design of handcrafted methods for object detection impossible. Although this problem has been the subject of research since the early days of the computer vision field, it is only recently that researchers have developed generic object recognition systems for a broad class of real world objects. The breakthrough was the use of a machine learning framework that makes use of very large sets of sample images to learn robust models: Given a training set of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and l_i is the category of the object present in $\boldsymbol{\rho}_i$, the aim is to learn a model, $h(\boldsymbol{\rho}_i) = l_i$ that maps images to object categories. In the specific case of traffic sign detection, $l_i \in \{-1, +1\}$, where $+1$ means presence of a traffic sign and -1 otherwise.

The classical methods for visual recognition involve two steps. First, a set of visual features is extracted from the image and the object of interest is represented using

S. Escalera et al., *Traffic-Sign Recognition Systems*, SpringerBriefs in Computer Science,
DOI: 10.1007/978-1-4471-2245-6_3, © Sergio Escalera 2011

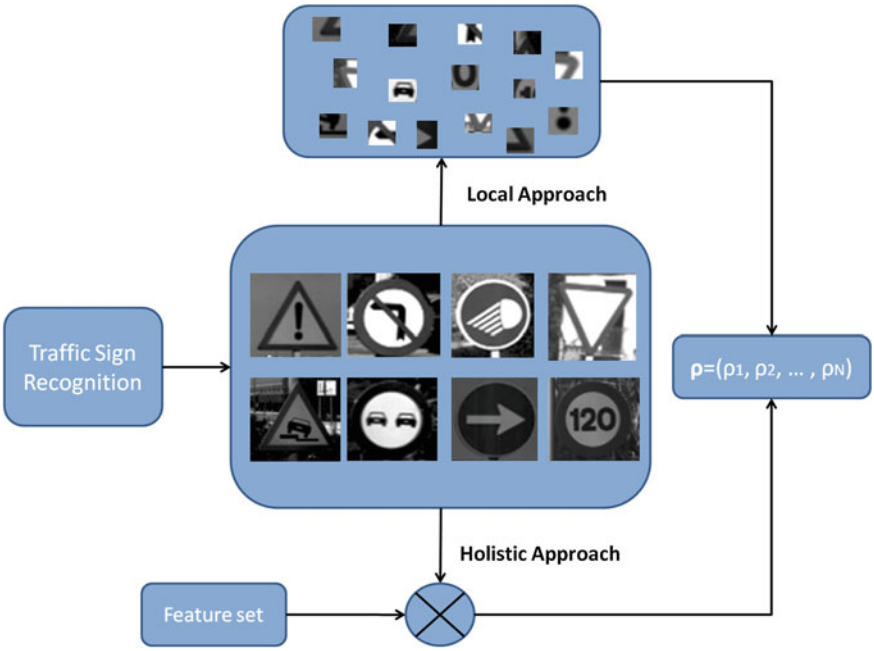


Fig. 3.1 A representation of the two main approaches to image representation

these features. Feature selection plays a crucial role in recognition: it facilitates the identification of aspects that are shared by objects in the same class, despite the variations in their appearance, and it supports discrimination between objects and between classes that can be highly similar. In the second step a classification rule is learned from the chosen feature representation in order to recognize different instances of the object. Depending on the features extracted, a variety of classification methodologies have been proposed in the literature.

As regards the first step, there are two main approaches to deal with the feature extraction problem (see Fig. 3.1):

Holistic methods. use the whole object image, which corresponds to a window of the image where the object has to be detected, in order to define and extract a set of features that will represent a global view of the object. These systems are typically based on defining a template matching strategy by comparing image windows to different m "templates" and recording the similarity measures in a vector. Templates can be learned from data (e.g. using Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Non Negative Matrix Factorization (NDA) or some form of artificial neural net) or can be defined *a priori* (e.g. using a fixed wavelet dictionary or Gabor filter responses). Thus, an image $\boldsymbol{\rho}_i$ can be considered to be a vector $(\rho_{i,1}, \ldots, \rho_{i,k})$ of k scalar values corresponding to k similarity measures.

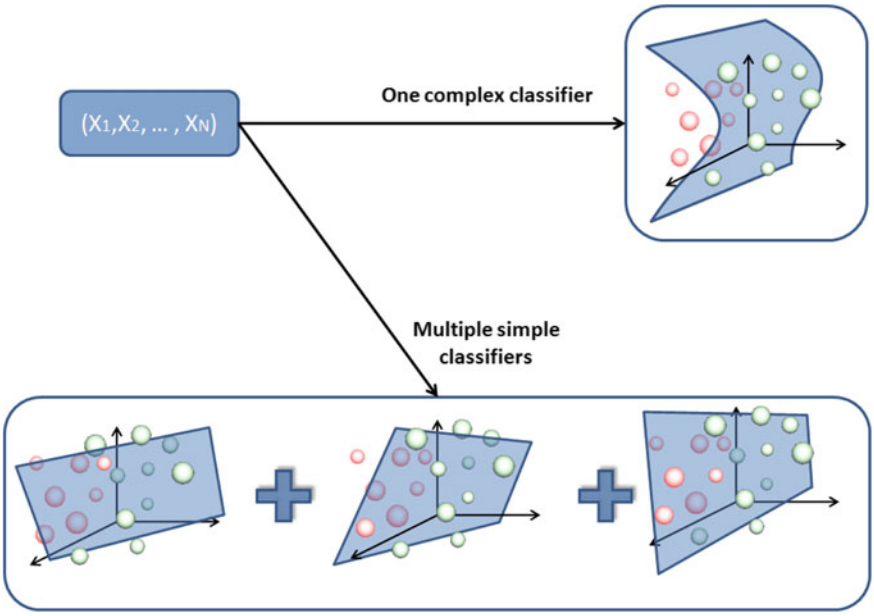


Fig. 3.2 The two main classification approaches

Local methods. model an object as a collection of local visual features or "patches". Thus, an image $\boldsymbol{\rho}_i$ can be considered to be a vector $(\boldsymbol{\rho}_{i,1}, \ldots, \boldsymbol{\rho}_{i,k})$ of k patches. Each patch $\boldsymbol{\rho}_{i,j}$ has a feature-vector representation $F(\boldsymbol{\rho}_{i,j}) \in \Re^d$; this vector might represent various features of the appearance of a patch, as well as features of its relative location and scale. We can choose from a wide variety of features, such as UllmanĠs fragments-based representation approach [1], the gradient orientation-based SIFT [2], or some forms of geometric invariant descriptors.

As regards the second step, there are two main approaches for defining the classification rule for object representation $\boldsymbol{\rho}_i$ (see Fig. 3.2): we can use a powerful k-dimensional classifier $h(\boldsymbol{\rho}_i)$ to learn a rule for assigning the object category, or we can use a classification rule $H(\boldsymbol{\rho}_i)$ based on the combination of the classification results of several "simple" classifiers $h_n(\boldsymbol{\rho}_i)$. In the first case we can use the Nearest Neighbor classifier, Support/Relevance Vector Machines, neural networks, etc. In the second case, the most successful approaches have been based on different versions of the AdaBoost algorithm [3], which are based on a weighted combination of "weak" classifiers $H(\boldsymbol{\rho}_i) = \sum_t \alpha_t h_t(\boldsymbol{\rho}_i)$.

This chapter presents an overview of the general problem of object recognition, and introduces the basic concepts. After presenting the general scheme, we show some classical features for describing the traffic sign images, and explain a learning method for building a traffic sign detector. Finally, we introduce an evolutionary version of the learning method, which allows us to work with a larger set of features.

3.1 Object Recognition

Object recognition is one of the most important, but least understood, aspects of visual perception. For many biological vision systems, the recognition and classification of objects is a spontaneous, natural activity. Young children can immediately and effortlessly recognize a large variety of objects [4].

In contrast, the recognition of common objects remains way beyond the capabilities of artificial systems, or of any recognition model proposed so far. The brain generalizes spontaneously from visual examples without the need for explicit rules and laborious instruction, and is able to recognize these objects under a huge variety of viewing conditions. In contrast, computers must be programmed to recognize specific objects with fixed, well-defined shapes. It is far more difficult for an object recognition system to capture the essence of a dog, a house or a tree, the kind of classification that is natural and immediate for the human visual system.

It is not easy to define the term *object recognition* in a simple, precise and uncontroversial manner. What do we mean, exactly, when we say that we *recognize an object*? The simplest answer might be: "naming an object in sight." This answer is not entirely unambiguous, because in recognizing an object, we sometimes wish to identify an individual object or a specific *token* (such as *my car*), while in other cases recognition means identifying the object as a member of a certain class, or a type (*a truck*). We will name the first case *identification*, in which we want to identify an individual object, and the second one *classification*, in which we only want to establish the fact of belonging to a certain class.

Furthermore, an object may belong to a number of classes or categories simultaneously (e.g. my cat, a Siamese cat, a cat, an animal). In the same way, in an image we can find multiple objects, each one of which can be composed of differentiated parts. The finality of the recognition system will define the level of classification we require. In the specific case of traffic signs, we can find a classification hierarchy. For instance, a traffic sign which limits the speed to 80 km/h may be just a traffic sign, a prohibition sign, a speed limit sign, an 80 km/h limitation sign or the 80 km/h limitation sign in a certain real world position.

Object recognition is a problem that seems easy enough to overcome if we have a sufficiently large and efficient memory system. In performing recognition, we are trying to determine whether the image we are currently seeing corresponds to an object we have seen in the past. It might be possible, therefore, to approach object recognition by storing a sufficient number of different views associated with each object, and then comparing the image of the currently viewed object with all the views stored in memory [5]. Several mechanisms, known as associative memories, have been proposed for implementing this *direct* approach to recognition. A major problem with this approach is that it relies on a simple and restricted notion of similarity in order to measure the distance between the input image and each of the images previously stored in the memory. As shown in [4], the use of a simple image comparison is insufficient by itself to cope with the large variations between different images of a given object. Summarizing, for the general problem of visual

object recognition this direct approach is insufficient, for two reasons: first, the space of all possible views of all the objects to be recognized is likely to be prohibitively large, and second (and more fundamentally) the image to be recognized will often not be sufficiently similar to any image seen in the past. The differences in this second case can be produced by several factors: Viewing position, photometric effects, object setting, or changing shape.

Viewing position. Three-dimensional objects can be viewed from a variety of viewing positions (directions and distances), and these different views can give rise to widely different images.

Photometric effects. These include the positions and distribution of light sources in the scene, their wavelengths, the effects of mutual illumination by other objects, and the distribution of shadows and specularities.

Object setting. In natural scenes, objects are rarely seen in isolation: they are usually seen against some background, next to, or partially occluded by, other objects.

Changing shape. Many objects, such as the human body, can maintain their identity while changing their three-dimensional shape. Changing objects, such as a pair of scissors, can sometimes be composed of rigid sub-parts. Other objects may undergo non-rigid distortions: for example, faces undergoing complex transformations due to facial expressions or hand movements.

A large variety of methods have been proposed for the task of visual object recognition, some of them as models of human vision, others as possible schemes for machine vision. Some of the proposed schemes are general in nature, while others were developed for specific application domains (see [6, 7] for reviews). In [4], Ullman classifies the basic approaches to recognition in three main classes, based on their approach to the regularity problem:

Invariant properties methods. Theories in this class assume that certain simple properties remain invariant under the transformations that an object is allowed to make.

Parts decomposition methods. Theories in this class rely on the decomposition of objects into parts. This leads to the notations of symbolic structural descriptions, feature hierarchies and syntactic pattern recognition.

Alignment methods. The main idea for theories in this class is to compensate for the transformations separating the viewed object and the corresponding stored model, and then to compare them.

This classification is a taxonomy of the underlying ideas, not of existing schemes. That is, a given scheme is not required to belong strictly to one of these classes, but may employ one or more of these ideas. A successful recognition scheme may in fact benefit from incorporating key ideas from all three classes.

3.1.1 Object Detection

Although theoretical object recognition defines the general framework for the problem faced in this chapter, we need to introduce more accurate and practical definitions

to the specific case of study, i.e. the traffic sign detection problem. We use the term *object detection* to describe a specific kind of *classification*, where we have only two classes: the class *object* and the class *no object*. In this case, the class *object* corresponds to a traffic sign, while the class *no object* means that it does not contain a traffic sign.

Given an input image, the goal of object detection is to return the instances of the class *object* in this image. Therefore, object detection is often posed as a search and classification problem: a search strategy generates potential image regions and a classifier determines whether or not they are an object. The standard approach is brute-force search, in which the image is scanned in raster order using a $n \times m$ pixels window over multiple image scales. For each scale and position the window is classified. When the brute-force search strategy is used, object detection is a *rare event detection* problem, in the sense that among the millions of image regions, only a few are objects.

When facing *rare event detection* problems, we are restricted to using a fast classification strategy in order to discard those millions of windows that do not contain any object. This fact must be considered in the three main aspects of the classifier: How objects are described, how objects are modeled, and how image regions are analyzed.

3.2 Features

There is no universal definition of what constitutes a feature, and the exact definition often depends on the problem or the type of application. In object recognition and pattern recognition in general, the objects or phenomena being observed must be described in a computationally plausible representation. In this context, features are the individual measurable heuristic properties of the object or phenomena of interest, and are restricted by the perception of the world available, such as sensors or transaction databases. Building proper representations has become an important issue in pattern recognition [8].

In the case of computer vision, the starting point is always a matrix with numerical values which are interpreted as an image. Depending on the application, this image can represent distances from a sensor, the spectrogram of a sound or more frequently the intensity value on a light sensor matrix. Although this numerical matrix is in fact a set of features that represent the object, in general, this initial representation of the object is not enough to perform classification tasks, and more complex features must be computed. We can define a feature as any function $f : I \mapsto R$ that takes an image I as input and returns a real value as output. In general, we never use a feature alone; instead, we use groups or families of features which are referred to as a feature set. While different areas of pattern recognition obviously have different features, once the features are decided they are classified by a much smaller set of algorithms. These include nearest neighbor classification in multiple dimensions, neural networks, or statistical techniques.

The description of an object by means of a set of features is known as feature extraction. Although the nature of our objects and application will suggest the features we can use, in the literature different levels of features are used for different purposes. In 1969, Levine [9] classified the features into two main groups: Microanalysis and macroanalysis. The main idea is that previous to the description of an image, we need to identify the most informative regions of that image, using a set of generalist features such as edges, lines, and so on. This initial process is called the microanalysis, while the description of the informative regions using a more problem-dependant features is called the macroanalysis. Apart from nomenclature changes (i.e. microanalysis is commonly referred to as feature detection), this division is still accepted. It is important to state that this is a fuzzy division, and some features can be classified in either of the groups depending on the application.

It seems clear, both from biological and computational evidence, that some form of data compression occurs at a very early stage in image processing. Moreover, a great deal of physiological evidence suggests that one form of this compression involves finding edges and other discriminative features in images. Edges often occur at points where there is a large variation in the luminance values in an image, and consequently they often indicate the edges, or occluding boundaries, of the objects in a scene. However, large luminance changes can also correspond to surface markings on objects. Points of tangent discontinuity in the luminance signal (rather than simple discontinuity) can also signal an object boundary in the scene.

So the first problem encountered with modeling this biological process is that of defining, precisely, what an edge might be. The usual approach is to simply define edges as step discontinuities in the image signal. Often, then, the way to localize these discontinuities is to find local maxima in the derivative of the signal, or zero-crossings in the second derivative of the signal. This idea was first suggested to the AI community, both biologically and computationally, by Marr [10], and later developed by Marr and Hildreth [11], Canny [12], and many others [13, 14].

Feature detection usually refers to the computation of points of interest where we compute the features. A point of interest is a point in the image where the local image structure around it is rich in terms of local information contents, such that the use of points of interest simplifies further processing in the vision system. In general, the points of interest are required to be invariant under local and global perturbations in the image domain, including deformations such as those arising from perspective transformations (sometimes reduced to affine transformations, scale changes, rotations and/or translations) as well as illumination/brightness variations, such that the points of interest can be computed reliably and with a high degree of reproducibility.

Once the informative regions of an object are selected, there is a long list of possible descriptors to characterize them. Most of them are based on illumination changes or on gradient information. This stage is commonly known as feature extraction. As we will see later, feature detection is sometimes mixed within the feature extraction process, and we cannot differentiate between the two processes.

3.2.1 Haar-Like Features

One of the most successful a priori image features, at least for a broad class of visual objects, is the Haar-like feature. Haar-like features, which are related to the discrete wavelet decomposition (DWT), were originally proposed in the framework of object detection by Viola and Jones [15] in their face detection approach.

The foundations of the DWT go back to 1976 with the studies of Croiser et al. [16] and Crochiere et al. [17], where a technique was devised in order to decompose discrete time signals. They called their analysis scheme sub-band coding, which was later improved by Vetterli and Le Gall [18] who removed the redundancy existing in the pyramidal coding scheme.

The two-dimensional Haar decomposition of a square image with n^2 pixels consists of n^2 wavelet coefficients, each of which corresponds to a distinct Haar wavelet. The first of these wavelets is the mean pixel intensity value of the whole image, and the rest of the wavelets are computed as the difference in mean intensity values of horizontally, vertically, or diagonally adjacent squares. In Fig. 3.3, a sample of a 2D decomposition of a Traffic Sign image is presented. Note that Haar wavelets are sensitive to edges in the image.

Although the Haar wavelet is widely used in the image processing and image compression fields, Haar-like features, which are a simplified version, are often used in pattern recognition. Since their appearance in the work of Viola and Jones [15] and the posterior extension in [19], Haar-like features have become a standard image feature for the detection of a broad class of visual objects. These features are based on a set of predefined patterns (see Fig. 3.4), which are defined in order to detect concrete structures in an image. Haar-like features are an extension of the Haar wavelet definition to all possible adjacent rectangles in the image.

3.2.2 Dissociated Dipoles

Although the representation of image structure using differential operators that compare adjacent image regions is well suited to encoding local relationships, these operators have significant drawbacks. One of the major problems arises when we try to compare distant regions of the image. In fact, any comparison of small regions across long distances is quite difficult, since an operator large enough to span the relevant distance must trade resolution for size. Alternatively, comparing distant regions by propagating information via a chain of small sized operators increases the susceptibility to the noise contributed by each of the intermediate elements.

In [20], Balas and Sinha introduce the *Dissociated Dipoles* or *Sticks operator*, for encoding non-local image relationships. These features aim to allow comparison of small images across long distances. As in the case of Haar-like features, *Dissociated Dipoles* are based on a classical wavelet image description technique, the Gabor wavelets, which have been widely applied on iris and fingerprint recognition problems.

Fig. 3.3 Second level decomposition of a Stop Sign Image using Haar Wavelet

While Gabor-like operators provide a simple means of representing image structure, the local image processing they embody limits a recognition system in significant ways [20]:

> Edge-based representations may fail to adapt to small changes in an image brought on by changes in object geometry or position. This particular weakness stems from more general problems with edge-based algorithms, namely that most natural images contain relatively few high-frequency (edge-like) components. Consequently, edge maps implicitly ignore most of the content of an image, and can suffer dramatically from subtle transformations that perturb edge locations while leaving large portions of the image untouched. Simple analysis also strains the capabilities of a Gabor-based representation scheme due to the conflation of the size of an operator's lobes with the distance spanned by this operator. In fact, any comparison of small regions across large distances proves quite difficult, since an operator large enough to span the relevant distance must trade resolution for size. Alternatively, comparing distant regions by propagating information via a chain of small sized operators increases susceptibility to the noise contributed by each of the intermediate elements.

Balas and Shina [20] state that the primary source of the shortcomings of the conventional differential operators is the confusion of the inter-lobe distance with

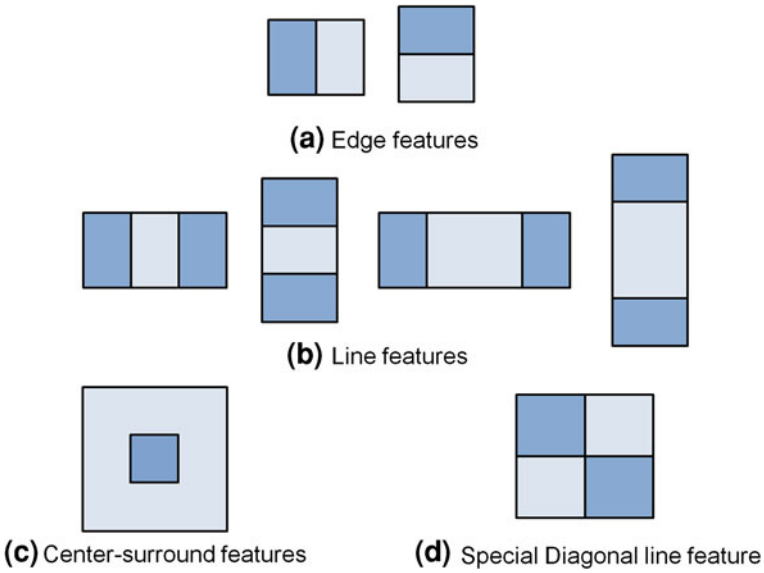


Fig. 3.4 Extended Haar-like feature patterns [19]

lobe-size. Therefore, they de-couple the lobe-size and inter-lobe distance parameters, allowing the operator to compare small regions separated by long distances. The result is the *Dissociated Dipole* or *Sticks* operator as a tool for performing non-local image comparisons.

Like a simple edge-finder, a *Dissociated Dipole* is a differential operator consisting of an excitatory and an inhibitory lobe, and may be used in any orientation or at any scale. However, unlike a conventional edge detector, the correlation of inter-lobe distance and lobe size has been removed; therefore, they allow an arbitrary separation between these two lobes. Formally, the basic form of a *Dissociated Dipole* operator comprises a pair of Gaussian lobes, each with standard deviation σ and a spatial separation of δ.

As in the case of Haar-like features, *Dissociated Dipole* has a simplified and computationally more feasible representation, introduced by Balas and Sinha in [21], in which the lobes were approximated using rectangles (see Fig. 3.5). The discrimination power of these simplified features was studied from the point of view of recognition. Note that some patterns present on Haar-like features shown in Fig. 3.4 can also be simulated with this feature set.

3.3 Detection Methods

Once phenomena or objects of interest have been described using the desired features, the main goal is to hypothesize their class by choosing the model that best corresponds to each sensed pattern.

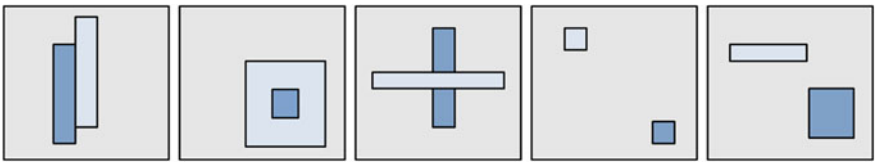


Fig. 3.5 Examples of bilobed differential operators of the sort employed in [21]

An intriguing problem that has remained unsolved since the beginnings of pattern classification is the relationship between the structure and type of data and the performance of the different classification methodologies. In other words, it is difficult to a priori know which approach is the best for use in a given problem. Although some studies such as Van der Walt and Barnard [22] investigated very specific artificial data sets to determine conditions under which certain classifiers perform better and worse than others, the classical methodology consists of testing the performance of a set of preselected approaches and choosing the best one. This previous selection of methodologies is usually subject to problem restrictions such as dimensionality or time/complexity constraints. However, determining a suitable classifier for a given problem is still more an art than a science.

Although a wide range of classification functions exists, the first approach to a classification problem should be a linear classifier in which the classification decision is based on the value of the linear combination of the features. A linear classifier can be written as $l = h'(\boldsymbol{\rho}, \mathbf{w}) = g(\mathbf{w} \cdot \boldsymbol{\rho} + b)$, where $\mathbf{w}$ is a real vector of weights and g is a function that converts the dot product of the two vectors into the desired output. Often g is a simple function that maps all values above a certain threshold to the first class and all other values to the second class. A more complex g might reflect the probability that an item belongs to a certain class l.

When working in a binary classification problem, one can visualize the operation of a linear classifier as splitting a high-dimensional input space with a hyperplane, where all points on one side of the hyperplane are classified as *positive*, while the others are classified as *negative*. These linear classifiers are usually referred to as *decision stumps*, and are often used in situations where the speed of classification is an issue, since they tend to be the fastest classifiers, especially when $\boldsymbol{\rho}$ is sparse or has a large dimensionality. However, decision trees may be faster.

In the literature, we find two broad classes of methods for determining the parameters of a linear classifier, that is, the values of $\mathbf{w}$:

Generative models. Approaches that estimate $\mathbf{w}$ by modeling conditional density functions $P(\boldsymbol{\rho}|\text{class})$. Examples of such algorithms include:

Linear discriminant analysis. (or Fisher's linear discriminant) (LDA), where Gaussian conditional density models are assumed. In spite of its name, it does not belong to the class of discriminative models in this taxonomy. However, its name makes sense when we compare LDA to the other main linear dimensionality reduction algorithm, such as Principal Components Analysis (PCA).

Naive Bayes classifier. Which assumes independent binomial conditional density models.

Discriminative models. The discriminative models which attempt to maximize the quality of the output on a training set. Additional terms in the training cost function can easily perform regularization of the final model. Examples of discriminative training of linear classifiers include

Logistic regression. A maximum likelihood estimation of $\mathbf{w}$ is performed assuming that the observed training set was generated by a binomial model that depends on the output of the classifier. In [23], Mitchell shows an interesting relationship between logistic regression and Naive Bayes Classifiers: The parametric form of $P(L|\rho)$ used by Logistic Regression is precisely the form implied by the assumptions of a Gaussian Naive Bayes classifier. Therefore, we can see Logistic Regression as a closely related alternative to GNB, though the two can produce different results in many cases.

Perceptron. An algorithm that attempts to fix all errors encountered in the training set. This is one of the most simple algorithms. Although the findings of Minsky and Papert [24] caused a significant decline in interest and funding of neural network research, and thus in the use of Perceptron, it is studied as one of the first classifiers.

Support vector machine. An algorithm that maximizes the margin between the decision hyperplane and the examples in the training set. SVM is considered one of the most important methods in the recent start-of-the-art in classification.

All the linear classifier algorithms listed above can be converted into non-linear algorithms operating in a different input space $\varphi(\boldsymbol{\rho})$, using the kernel trick.

Discriminative training often yields higher accuracy than modeling the conditional density functions. However, when training data is limited or when handling missing data, generative approaches may be preferable [25].

Recently, a generation of hybrid methods has been developed, combining generative models and discriminative learning in order to benefit from the two approaches. Examples of these methods can be found in [26–30].

After this general overview of detection methods, in what follows we present a general theoretical framework to machine learning to explain the AdaBoost learning algorithm, which will be used in order to deal with the traffic sign detection problem.

3.3.1 PAC Model of Learning

The *Probably Approximately Correct* (PAC) model of learning is a theoretical framework for studying machine learning. In this section we give only the main definitions for this model, in order to establish the basis for the study of certain learning approaches in the later sections.

Definition 3.1 Let $\boldsymbol{\rho}$ be a set called the *instance space*. We think of $\boldsymbol{\rho}$ as being a set of encodings of instances or objects in the learner's world.

Definition 3.2 A *concept* over $\boldsymbol{\rho}$ is just a subset $c \subseteq \boldsymbol{\rho}$ of the instance space. It can be thought of as the set of all instances that positively exemplify some simple or interesting rule. We can equivalently define a concept to be a Boolean mapping $c : \boldsymbol{\rho} \rightarrow \{-1, +1\}$, with $c(\rho) = +1$ indicating that $\rho \in \boldsymbol{\rho}$ is a positive example of c and $c(\rho) = -1$ indicating that ρ is a negative example. For this reason, $\boldsymbol{\rho}$ is also called the *input space*.

A *concept class C* over $\boldsymbol{\rho}$ is a collection of concepts over $\boldsymbol{\rho}$. Ideally, we are interested in concept classes that are sufficiently expressive for fairly general knowledge representation.

In this model, a learning algorithm will have access to positive and negative examples of an unknown *target concept c*, chosen from a known concept class *C*. The learning algorithm will be judged by its ability to identify a hypothesis concept that can accurately classify instances as positive or negative examples of c. In this model the learning algorithms "know" the target class *C*, in the sense that the designer of the learning algorithm is guaranteed that the target concept will be chosen from *C*, although he/she must design the algorithm to work for any $c \in C$.

Definition 3.3 Let *W* be any fixed probability distribution over the instance space $\boldsymbol{\rho}$. We will refer to *W* as the *target distribution*. If *h* is any concept over $\boldsymbol{\rho}$, then the distribution *W* provides a natural measure of *error* between *h* and the target concept *c*. We can define:

$$error(h) = \mathbf{Pr}_{\rho \in W}\left[c(\rho) \neq h(\rho)\right] \tag{3.1}$$

Definition 3.4 Let $EX(c, W)$ be a procedure (sometimes called an *oracle*) that runs in unit time, and on each call returns a labeled example $\langle \rho, c(\rho) \rangle$, where ρ is given randomly and independently according to *W*. A learning algorithm will have access to this procedure when learning the target concept $c \in C$. Ideally, the learning algorithm will satisfy three properties:

The number of calls to *EX*(*c,W*) is small, in the sense that it is bounded by a fixed polynomial in some parameters (to be specified shortly). The amount of computation performed is small. The algorithm outputs a *hypothesis concept h* such that *error*(*h*) is small.

The number of calls made by a learning algorithm to *EX*(*c,W*) is bounded by the running time of the learning algorithm.

Finally, the PAC model can be defined as follows:

Definition 3.5 Let *C* be a concept class over $\boldsymbol{\rho}$. We say that *C* is *PAC learnable* if there exists an algorithm *L* with the following property: for every concept $c \in C$, for every distribution *W* on $\boldsymbol{\rho}$, and for all $0 < \varepsilon < 1/2$ and $0 < \delta < 1/2$, if *L* is given access to *EX*(*c,W*) and inputs ε (error parameter) and δ (confidence parameter), then with probability at least $1 - \delta$, *L* outputs a hypothesis concept $h \in C$ satisfying $error(h) \leq \varepsilon$. This probability is obtained from the random examples of *EX*(*c,W*), and any internal randomization of *L*.

The hypothesis $h \in C$ of PAC learning algorithm thus has a high probability of being "approximately correct"—hence the name Probably Approximately Correct learning. For more detailed and extended definitions of the PAC model of learning, a good reference is [31].

3.3.2 Boosting

Learning algorithms that output only a single hypothesis suffer from three problems [32]:

Statistical problem. The statistical problem arises when the learning algorithm is searching a space of hypotheses that is too large for the amount of training data available. In such cases, several different hypotheses may all give the same accuracy for the training data, and the learning algorithm must choose one of these to output. There is a risk that the chosen hypothesis will not predict future data points well. When a learning algorithm suffers from this problem, it is said to have high "variance".

Computational problem. The computational problem arises when the learning algorithm cannot guarantee to find the best hypothesis within the hypothesis space. In some types of classifiers, such as neural networks or decision trees, the task of finding the hypothesis that best fits the training data is computationally intractable, and so heuristic methods must be employed. But these heuristics may get stuck in local minima and may fail to find the best hypothesis. When a learning algorithm suffers from this problem, it is said to have high "computational variance".

Representation problem. The representation problem arises when the hypothesis space does not contain any hypotheses that are good approximations to the true function $f(\boldsymbol{\rho})$. When a learning algorithm suffers from this problem, it is said to have high "bias".

All these problems can be smoothed by using a weighted vote of hypotheses. A weighted vote of equal accuracy hypotheses reduces the risk in the case of high variance methods. In the same way, considering the combination of several different local minima reduces the risk of choosing the wrong local minimum to output in methods with high computational variance. Finally, the combination of several hypotheses allows us to form a more accurate approximation to $f(\boldsymbol{\rho})$, improving the methods with high bias. A basic schema for an ensemble of classifiers that perform a weighted vote is shown in Fig. 3.6. Notice that the final hypothesis h is obtained by combining classifier hypotheses h_i.

One of the most widely used methods to learn ensembles of classifiers is the Boosting approach, a general method for improving the accuracy of any given learning algorithm.

Definition 3.6 Given $\varepsilon, \delta > 0$ and access to random examples, an algorithm is a *strong* PAC-learning algorithm if its output probability $1 - \delta$ has an error of at most ε. Further, the running time must be polynomial in $1/\varepsilon$, $1/\delta$ and other relevant parameters (namely, the "size" of the examples received, and the "size" or "complexity" of the target concept).

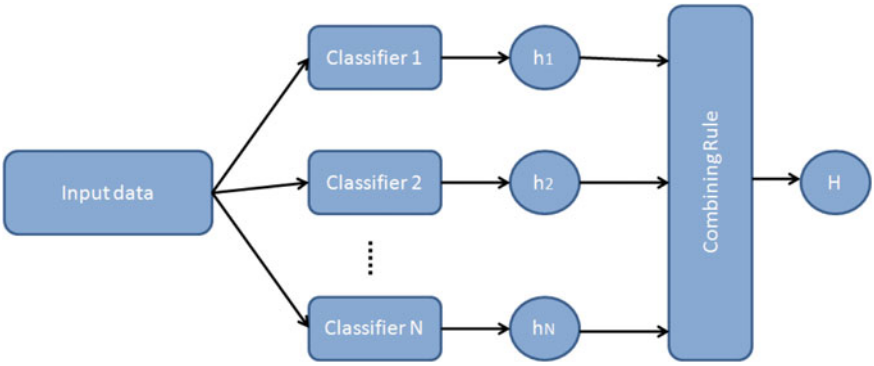


Fig. 3.6 Ensemble of classifiers framework

Definition 3.7 Given $\varepsilon, \delta > 0$ and access to random examples, an algorithm is a *weak* PAC-learning algorithm if its output probability $1 - \delta$ has an error of at most $\varepsilon \geq 1/2 - \gamma$, where $\gamma > 0$ is either a constant, or decreases as $1/p$ where p is a polynomial in the relevant parameters [31, 33]. We will use *WeakLearn* to denote a generic weak learning algorithm.

In [34], Schapire showed that any weak learning algorithm can be efficiently transformed or "boosted" into a strong learning algorithm. Later, in [35] Freund presented the "boost-by-majority" algorithm that is considerably more efficient than Schapire's. Both algorithms work by calling a given weak learning algorithm *WeakLearn* multiple times, each time with a different distribution over $\boldsymbol{\rho}$, and finally combining all the hypotheses generated into a single hypothesis. The intuitive idea is to alter the distribution over the domain $\boldsymbol{\rho}$ in a way that increases the probability of the "harder" parts of the space, thus forcing the weak learner to generate new hypotheses that make fewer mistakes in these parts.

Boost-by-majority algorithm requires that the bias γ of the weak learning algorithm *WeakLearn* be known ahead of time, and this is an important practical deficiency. Not only is this worst-case bias usually unknown in practice, but typically the bias that can be archived by *WeakLearn* will vary considerably from one distribution to the next. Unfortunately, the boost-by-majority algorithm cannot take advantage of hypotheses computed by *WeakLearn* with an error significantly smaller than the presumed worst-case bias of $1/2 - \gamma$.

3.3.3 AdaBoost

In [33], Freund and Schapire introduced the Adaboost algorithm, very nearly as efficient as boost-by-majority. However, unlike boost-by-majority, the accuracy of the final hypothesis produced by Adaboost depends on the accuracy of all the hypotheses

returned by *WeakLearn*, and so is able to more fully exploit the power of the weak learning algorithm [35].

Although boosting has its roots in the PAC model, Adaboost is defined in a more general learning framework, in which the learner receives examples (ρ_i, l_i) chosen randomly according to some fixed but unknown distribution P on $\rho \times L$, where L is a set of possible labels. As usual, the goal is to learn to predict the label l_i given an instance ρ. Adaboost calls a given weak learning algorithm repeatedly in a series of rounds $t = 1, \ldots, T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution in the training example i and in round t is denoted by $W_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set.

The goodness of a weak hypothesis is measured by its *weighted error*

$$\varepsilon_t = Pr_{i \sim W_t}[h_t(\rho_i) \neq l_i] = \sum_{i:h_t(\rho_i) \neq l_i} W_t(i) \tag{3.2}$$

Alternatively, when the weak learner cannot be trained using the weights W_t on the training examples, a subset of the training examples can be sampled according to W_t, and used to train the weak learner.

At each iteration, Adaboost calculates the updating rule β_t. The parameter β_t is chosen as a function of ε_t and is used to update the W_t. The update rule reduces the probability assigned to those examples in which the hypothesis makes a good prediction, and increases the probability of the examples in which the prediction is poor. Furthermore, if h_t is Boolean (with range $\{0, 1\}$), then it can be shown that this update rule exactly removes the advantage of the last hypothesis. That is, the error of h_t on distribution W_{t+1} is exactly $\frac{1}{2}$. The original Adaboost algorithm proposed by Freund and Shcapire [33] is shown in Algorithm 1.

Algorithm 1 The adaptive boosting algorithm [47]

Require: A training set $\boldsymbol{\rho}$ of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and $l_i \in \{-1, +1\}$ is the category of the object present in $\boldsymbol{\rho}_i$, a weak learning algorithm (**WeakLearner**), a distribution W over the m examples, and the maximum number of iterations T.

Initialize the weight vector: $w_i^1 = W(i)\, \forall i = 1, \ldots, m$.

for $t = 1, \ldots, T$ **do**

Set

$$p^t = \frac{w^t}{\sum_{i=1}^{m} w_i^t}$$

Call **WeakLearn**, providing it with the distribution $\mathbf{p}^t$: obtain a hypothesis $h_t : \boldsymbol{\rho} \rightarrow [0, 1]$.

Calculate the error of h_t: $\varepsilon_t = \sum_{i=1}^{N} p_i^t |h_t(\rho_i) - h_i|$
Set $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
Set the new weights vector to be

$$w_i^{t+1} = w_i^t \beta_t^{1-|h_t(\boldsymbol{\rho}_i)-l_i|}$$

end for
Ensure: The final hypothesis:

$$H(\rho) = \begin{cases} 1 & \text{if} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) h_t(\boldsymbol{\rho}) \geq \frac{1}{2} \sum_{t=1}^{T} \log \frac{1}{\beta_t} \\ -1 & \text{otherwise} \end{cases}$$

The Adaboost algorithm can be interpreted as a stagewise estimation procedure for fitting an additive logistic regression model. It optimizes an exponential criterion with the second order equivalent to the binomial log-likelihood criterion. In [36], Friedman proposes a more standard likelihood-based boosting procedure. Because this algorithm will be widely used in this thesis, we introduce a theoretical overview based on Friedman's work.

Viewing the boosting procedures as stagewise algorithms for fitting additive models helps to understand their performance. AdaBoost fits an additive model $F(\boldsymbol{\rho}) = \sum_{t=1}^{T} c_t h_t(\boldsymbol{\rho})$.

We initially focus on the regression problem, where the response l is quantitative, $\boldsymbol{\rho}$ and l have some joint distribution, and we are interested in modeling the mean $E(l|\boldsymbol{\rho}) = H(\boldsymbol{\rho})$. The additive model has the form

$$H(\boldsymbol{\rho}) = \sum_{j=1}^{p} f_j(\rho_j) \tag{3.3}$$

There is a separate function $f_j(\rho_j)$ for each of the p input variables ρ_j. More generally, each component f_j is a function of a small, prespecified subset of the input variables. The *backfitting algorithm* [3, 37] is a convenient modular "Gauss-Seidel" algorithm for fitting additive models. A backfitting update is

$$f_j(\rho_j) \leftarrow E\left[l - \sum_{k \neq j} f_k(\rho_k)|\rho_j\right] \quad \text{for } j = 1, 2, \ldots, p, 1, \ldots \tag{3.4}$$

Any method or algorithm for estimating a function of f_j can be used to obtain an estimate of the conditional expectation in Eq. 3.4. This can include nonparametric smoothing algorithms, such as local regression or smoothing splines. On the right-hand side, all the latest versions of the functions f_k are used in forming the partial residuals. The backfitting cycles are repeated until convergence. Under fairly general conditions, backfitting can be shown to converge to the minimizer of $E(l - F(\boldsymbol{\rho}))^2$ [38].

Now we will consider an additive model whose elements $\{f_t(\boldsymbol{\rho})\}_1^t$ are functions of potentially all of the input features $\boldsymbol{\rho}$. In this context the $f_t(\boldsymbol{\rho})$ are taken to be simple functions characterized by a set of parameters γ and a multiplier β_t,

$$f_t(\boldsymbol{\rho}) = \beta_t b(\boldsymbol{\rho}; \gamma_t) \tag{3.5}$$

The additive model then becomes

$$F_T(\boldsymbol{\rho}) = \sum_{t=1}^{T} \beta_t b(\boldsymbol{\rho}; \gamma_t) \tag{3.6}$$

If least-squares is used as a fitting criterion, one can solve for an optimal set of parameters through a generalized backfitting algorithm with updates

$$\{\beta_t, \gamma_t\} \leftarrow \arg\min_{\beta,\gamma} E\left[l - \sum_{k \neq t} \beta_k b(\boldsymbol{\rho}; \gamma_k) - \beta b(\boldsymbol{\rho}; \gamma)\right]^2 \tag{3.7}$$

for $t = 1, 2, \ldots, T$ in cycles until convergence. Alternatively, one can use a "greedy" forward stepwise approach,

$$\{\beta_t, \gamma_t\} \leftarrow \arg\min_{\beta,\gamma} E\left[l - F_{t-1}(\boldsymbol{\rho}) - \beta b(\boldsymbol{\rho}; \gamma)\right]^2 \tag{3.8}$$

for $t = 1, 2, \ldots, T$ where $\{\beta_k, \gamma_k\}_i^{t-1}$ are fixed at their corresponding solution values at earlier iterations.

In boosting jargon, $f(\boldsymbol{\rho}) = \beta b(\boldsymbol{\rho}; \gamma)$ would be called a *weak learner* and $F_T(\boldsymbol{\rho})$ (Eq. 3.6) the *committee*. At this point note that the backfitting procedure, independently of the version we use (the general or the greedy) only requires an algorithm for fitting a *single* weak learner Eq. 3.5 to data. This base algorithm is simply applied repeatedly to modified versions of the original data

$$h_t \leftarrow l - \sum_{k \neq t} f_k(\boldsymbol{\rho}) \tag{3.9}$$

In the forward stepwise procedure (Eq. 3.8), the modified output h_t at the tth iteration depends only on its value h_{t-1} and the solution $h_{t-1}(\boldsymbol{\rho})$ at the previous iteration,

$$l_m = l_{m-1} - f_{m-1}(\boldsymbol{\rho}) \tag{3.10}$$

At each step t, the previous output value l_{t-1} is modified so that the previous model f_{t-1} has no explanatory power on the new outputs l_t. One can therefore view this as a procedure for boosting a weak learner $f(\boldsymbol{\rho}) = \beta b(\boldsymbol{\rho}; \gamma)$ to form a powerful committee $F_M(\boldsymbol{\rho})$.

Now, consider minimizing the criterion:

$$J(F) = E\left(e^{-yF(\boldsymbol{\rho})}\right) \tag{3.11}$$

for estimation of $F(\boldsymbol{\rho})$. Here E represents expectation: depending on the context, this may be a population expectation (with respect to a probability distribution) or else a sample average. E_w indicates a weighted expectation. Lemma 1 shows that the function $F(\boldsymbol{\rho})$ that minimizes $J(F)$ is the symmetric logistic transform of $P(l = 1|\boldsymbol{\rho})$. (The proof can be found in [36]).

Lemma 1 $E(e^{-yF(\boldsymbol{\rho})})$ *is minimized* at

$$F(\boldsymbol{\rho}) = \frac{1}{2}\log\frac{P(l = 1|\boldsymbol{\rho})}{P(l = -1|\boldsymbol{\rho})} \tag{3.12}$$

Hence

$$P(l = 1|\boldsymbol{\rho}) = \frac{e^{F}(\boldsymbol{\rho})}{e^{-F(\boldsymbol{\rho})} + e^{F(\boldsymbol{\rho})}} \tag{3.13}$$

$$P(l = -1|\boldsymbol{\rho}) = \frac{e^{-F(\boldsymbol{\rho})}}{e^{-F(\boldsymbol{\rho})} + e^{F(\boldsymbol{\rho})}} \tag{3.14}$$

Corollary 2 *If E is replaced by averages over regions of $\boldsymbol{\rho}$ where $F(\boldsymbol{\rho})$ is constant (as in the terminal node of a decision tree), the same result applies to the sample proportions*

Result 3 The Discrete Adaboost algorithm builds an additive logistic regression model via Newton-like updates for minimizing $E(e^{-yF(\boldsymbol{\rho})})$.

3.3.3.1 Adaboost Variants

Since Freund and Schapire presented their AdaBoost algorithm, many other versions of this algorithm have been developed. All these versions differ in the manner in which they modify the weights and construct the final hypothesis. In the following sections we show the algorithms for some of these versions. Like the original AdaBoost algorithm, all their versions can be interpreted from a statistical point of view [36]. In what follows, the key points and algorithms of some different versions are introduced. Notice that the versions differ slightly in the way they update the weights and in the output formulation.

Real Adaboost. The *Real AdaBoost* algorithm builds an additive logistic regression by stagewise and approximate optimization of $J(F) = E\left[e^{-lF(\boldsymbol{\rho})}\right]$. The Real Adaboost can be viewed as an Adaboost with Confidence Weighted Predictions, in the sense that it uses class probability estimates $p_t(\boldsymbol{\rho})$ to construct real-valued contributions $f_t(\boldsymbol{\rho})$. The algorithm is shown in Algorithm 2.

Algorithm 2 The Real AdaBoost Algorithm

Require: A training set $\boldsymbol{\rho}$ of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and $l_i \in \{-1, +1\}$ is the category of the object present in $\boldsymbol{\rho}_i$, a weak learning algorithm (**WeakLearner**) and the maximum number of iterations T.
Initialize the weight vector $w_i = \frac{1}{m}, i = 1, 2, \ldots, m$
for $t = 1, \ldots, T$ **do**
Use the WeakLearn in order to fit the classifier to obtain a class probability estimate $p_t(\boldsymbol{\rho}) = \hat{P}_w(l = 1|\boldsymbol{\rho}) \in [-1, 1]$, using weights w_i on the training data
Set $f_t(\boldsymbol{\rho}) \leftarrow \frac{1}{2} \log \frac{p_t(\boldsymbol{\rho})}{1-p_t(\boldsymbol{\rho})} \in \mathbf{R}$
Set $w_i \leftarrow w_i e^{-l_i f_t(\boldsymbol{\rho}_i)}, i = 1, 2, \ldots, m$, and renormalize so that $\sum_i w_i = 1$.
end for
Ensure: The final hypothesis:

$$F_T = sign\left(\sum_{t=1}^{T} f_t(\boldsymbol{\rho})\right)$$

LogitAdaboost. The *LogitAdaboost* algorithm uses Newton steps for fitting an additive symmetric logistic model by maximum likelihood. The algorithm is shown in Algorithm 3.

Algorithm 3 The Logit AdaBoost Algorithm

Require: A training set $\boldsymbol{\rho}$ of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and $l_i \in \{-1, 1\}$ is the category of the object present in $\boldsymbol{\rho}_i$, a weak learning algorithm (**WeakLearner**) and the maximum number of iterations T.
Start with weights $w_i = \frac{1}{m}, i = 1, 2, \ldots, m, F(\boldsymbol{\rho}) = 0$ and probability estimates $p(\boldsymbol{\rho}_i) = \frac{1}{2}$
for $t = 1, \ldots, T$ **do**
Compute the working response

$$z_i = \frac{h_i^* - p(\boldsymbol{\rho}_i)}{p(\boldsymbol{\rho}_i)(1 - p(\boldsymbol{\rho}_i))}$$

where h_i^* represents outcome and is a $-1/+1$ response.
Compute the weights

$$w_i = p(\boldsymbol{\rho}_i)(1 - p(\boldsymbol{\rho}_i)$$

Find the function $f_t(\boldsymbol{\rho})$ by a weighted least-squares regression of z_i to $\boldsymbol{\rho}_i$ using weights w_i.
Update $F(\boldsymbol{\rho}) \leftarrow F(\boldsymbol{\rho}) + \frac{1}{2} f_t(\boldsymbol{\rho})$ and $p(\boldsymbol{\rho}) \leftarrow \frac{e^F(\boldsymbol{\rho})}{e^{F(\boldsymbol{\rho})} + e^{-F(\boldsymbol{\rho})}}$
end for

Ensure: The final hypothesis:

$$F_T = sign[F(\boldsymbol{\rho})] = sign\left[\sum_{t=1}^{T} f_t(\boldsymbol{\rho})\right]$$

Gentle Adaboost. The *Gentle Adaboost* algorithm uses Newton steps for minimizing $E\left[e^{-yF(\boldsymbol{\rho})}\right]$. This is a modified version of the *Real Adaboost*, using Newton stepping rather than exact optimization at each step. The algorithm is shown in Algorithm 4.

Algorithm 4 The Gentle AdaBoost Algorithm

Require: A training set $\boldsymbol{\rho}$ of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and $l_i \in \{-1, 1\}$ is the category of the object present in $\boldsymbol{\rho}_i$, a weak learning algorithm (**WeakLearner**) and the maximum number of iterations T.
Initialize the weight vector $w_i = \frac{1}{m}, i = 1, 2, \ldots, m$
Initialize $F(\boldsymbol{\rho}) = 0$
for $t = 1, \ldots, T$ **do**
Fit the regression function f_t by weighted least-squares of l_i to $\boldsymbol{\rho}_i$ with weights w_i.
Update $F(\boldsymbol{\rho}) \leftarrow F(\boldsymbol{\rho}) + f_t(\boldsymbol{\rho})$
Update $w_i \leftarrow w_i e^{-l_i f_t(\boldsymbol{\rho}_i)}$ and renormalize.
end for
Ensure: The final hypothesis:

$$F_T = sign[F(\boldsymbol{\rho})] = sign\left[\sum_{t=1}^{T} f_t(\boldsymbol{\rho})\right]$$

In addition to the Adaboost variants that differ according to their use of the error values to update the weights, in the literature we can find different modifications in order to improve other aspects of the obtained classifiers. For instance, Mita [39] improves the generalization performance by using weak classifiers that include multiple features simultaneously. Feature co-occurrence makes it possible to classify difficult samples that are misclassified by weak classifiers using a single feature.

3.4 Evolutionary Approach

Rectangular features are rather primitive when compared with alternatives such as steerable filters [40]. Steerable filters, and their relatives, are excellent for the detailed analysis of boundaries, image compression, and texture analysis. In contrast, rectangle features, while sensitive to the presence of edges, bars and other simple image

structures, are quite coarse. Unlike steerable filters, only a few orientations of the rectangle features are possible.

In spite of their simplicity, rectangular features provide a rich image representation which supports effective learning. Their extreme computational efficiency provides ample compensation for their limited flexibility. Rectangular features have the property that a single feature can be evaluated at any scale and location in a few operations.

It is important to note that Haar features constitute an overcomplete dictionary of the image and that there are more than 2^{18} different features for a small image window of 576 pixels (24×24 pixels). This fact imposes a high computational cost on the learning step of the Adaboost algorithm, which involves several rounds of exhaustive searches. From a practical point of view, when using conventional hardware the development of a high performance object detector represents a learning time of the order of several hundred hours.

The work of Viola and Jones [15] was expanded on by Lienhart and Maydt [19], who showed that the use of a larger feature set may improve the convergence and performance of the final classifier. The extension of the feature set was done by adding rotated versions of original Haar-like features, and thus adding a factor to the exponential relation between the size of the feature set and the training time. In order to allow the use of extended feature sets, a redefinition of the classical approach is needed in order to avoid an exhaustive search over the feature space. In what follows, we introduce the evolutionary computation, and redefine the classical approach to take advantage of evolutionary methods that allow us to work with huge feature sets.

3.4.1 Introduction

Darwin hypothesized that living beings adapted and differentiated to varying conditions or niches in their environment through a process of evolution. Although in DarwinĠs times genetics was an unknown field, most of the studies which are based on evolution assume current knowledge on genetics and define algorithms based on chromosome-based encoding and the processes observed in the natural evolution of species. However, the Darwinian processes can be defined in a more general manner, with no assumptions about the implementation of these processes. This more general definition has been widely developed in memetics [41, 42], the field that attempts to cope with evolutionary models of information transmission. These models have been used to represent certain brain functions and social conducts. In [42], Calvin defines the essential processes in any evolutionary model, in order to ensure quality improvement over generations:

1. There must be a pattern involved.
2. The pattern must be copied somehow.
3. Variant patterns must sometimes be produced by chance.
4. The pattern and its variant must compete with one another for occupation of a limited work space.

5. The competition is biased by a multifaceted environment. This is Darwin's natural selection.
6. New variants always preferentially occur around the more successful of the current patterns. This is what Darwin later called an inheritance principle.

In addition to these essential processes, Calvin [42] introduced five other processes which can notably influence the rate of evolutionary change:

1. Stability may occur. Variants happen, but they are either nonviable or backslide easily.
2. Systematic recombination (crossing over, sex) generates many more variants than do copying errors and the far-rarer point mutations.
3. Fluctuating environments, shaping up more complex patterns capable of doing well in several environments.
4. Parcellation typically speeds evolution. It raises the surface-to-volume ratio (or perimeter-to-area ratio) and exposes a higher percentage of the population to the marginal conditions on the margins.
5. Local extinctions speed evolution because they create empty niches.

After introducing Darwinian processes, Calvin [43] defines a *Darwin machine* by analogy to a Turing machine, as a machine that involves an iteration process that yields a high-quality result. However, whereas a Turing machine uses logic, the Darwin machine uses rounds of variation, selection, and inheritance. In its original connotation, a *Darwin machine* is any process that bootstraps quality by utilizing all of the six essential features of a Darwinian process: A pattern is copied with variations, where populations of one variant pattern compete with another population, their relative success being biased by a multifaceted environment so that winners predominate in producing the further variants of the next generation.

The theoretical definition of a *Darwin machine* states the necessary conditions for evolution. There are various possibilities for implementing these processes. The most widely applied implementation aims to copy the natural implementation of those processes, using genetic theories, and is what we refer to as *Genetic Darwin Machine*. Another possibility is to use probabilistic models in order to implement these processes. Although in the literature these methods can be found under a wide set of different names, we group them together under the name of *Probabilistic Darwin Machines*. We first define the basic concepts for both implementations and the following sections analyze in depth some examples of implementation for both families of *Darwin machines*. Figure 3.7 shows a graphic comparison between the theoretical Darwin Machine and the Genetic Darwin Machine.

3.4.1.1 Genetic Darwin Machine

All living organisms are *coded* by means of their genetic material, represented in DNA chains. The DNA contains all the information used in the development and functioning of all known living organisms and some viruses. Within cells, DNA is

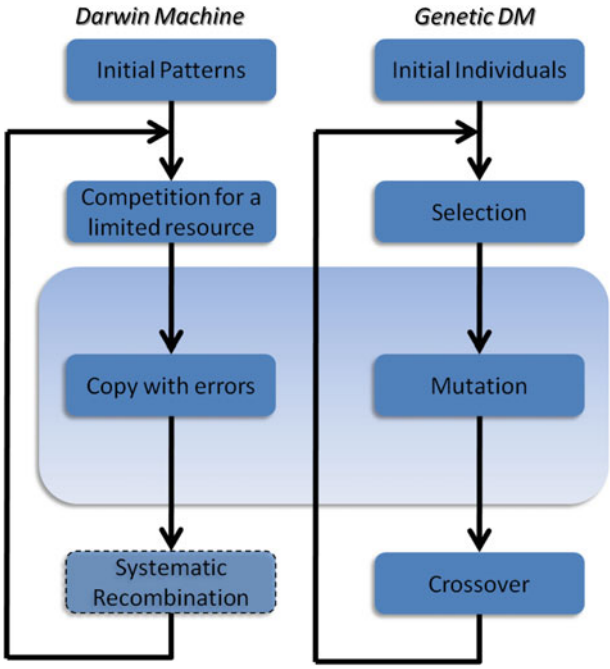


Fig. 3.7 Graphic comparison between Darwin Machines and Genetic Darwin Machines. Although systematic recombination is an optional operation in Darwin Machines, it is usually implemented by Genetic Darwin Machines

organized into structures called chromosomes, and the set of all the chromosomes is called the genome.

The genetic information is used as a recipe to create other structures of the cells such as proteins, which are responsible of most part of the processes of the organism, and therefore, the way in which the genetic information is expressed. Therefore, the capabilities and characteristics of an organism, and thus their ability to adapt to the environment, depend on the information contained in the chromosomes. Evolution concerns the natural processes that allow the perpetuation of useful genetic information and its improvement in order to adapt to changes in the environment.

The basis of any evolutionary algorithm relies on the concept of *Natural Selection*, in which the best individuals can survive and reproduce in order to perpetuate their species. The genetic information of those best individuals is transferred over generations, obtaining better individuals and improving the species. In general, in natural evolution, individuals are classified by their adaptation level: the best individuals are the ones that are most adapted to the environment. Apart from this information pathway through the generations, there are two other sources of evolution:

Crossover. In the reproduction process, two individuals of a population interchange their genetic information. Thus, the offspring have a genetic information that

contains parts of both their parents. When this mixing leads to a better adaptation to the environment, the new individuals will survive and this new genetic information will persist generation after generation. In the opposite case, if that the offspring do not survive, the new information will be discarded.

Mutation. In contrast to crossover, in which existing information is combined, in the case of mutation, new information can be generated. Mutation consists of randomly changing small parts of the genetic information. Analogously to crossover, when these changes imply better adaptation, they will be passed down the generations. Mutation allows evolution to generate new species.

Darwin explains the evolution of species from the inference of these processes (selection and variability), and the subsequent advances in our knowledge of genetics has defined how these aspects are accomplished by all living beings in nature. Biological evidence of the implementation of those concepts in nature is used in order to simulate natural evolution in a computational manner.

One of the first computational approaches to natural evolution was presented by Nils A. Barricelli [44], one of the pioneers in evolutionary computation. His initial experiments comprised a simple simulation of numbers in a grid. The numbers moved in the grid according to local rules that were specified for each number [45]. Barricelli made several observations about the patterns that emerged from such simple rules, which he termed *organisms*. Organisms were defined to be independent if they could reproduce without requiring other organisms of a different pattern, which he termed *another species*. He noted patterns of recombination, including a multiple point operation in which two patterns would collide and the result would be a new self-sustaining pattern with numbers chosen from each of the *parents*. Overall, Barricelli's search for emergent patterns is reminiscent of the search for emergent properties in complex adaptive systems that pervaded artificial life research in the late 1980s and early 1990s.

Beginning in 1957 with the work [46], the Australian quantitative geneticist Alex Fraser published a series of papers addressing different aspects from basic concepts to specific topics such as the simulation of artificial selection of organisms with multiple loci (position of a gene in the genome) controlling a measurable trait [47]. These initial studies established the basis of evolutionary computation. In the early 1960s computer simulation of evolution by biologists became more common and the methods were described in books by Fraser and Burnell [48], Crosby [49], and the Ph.D thesis by Rechenberg [50], which provided a comprehensive overview of the work carried out over a decade. Fraser's simulations included all the essential elements of the most widely used and probably most known *Genetic Darwin Machine*, the *Genetic Algorithms*, but they did not become popular until the publication of applied studies such as Schwefel [51], which used evolutionary computationto solve engineering problems.

Since the popularization of evolutionary algorithms, they have been widely used in several optimization problems, such as scheduling and function minimization. The fundamental approach to optimization is to formulate a single standard of measurement (a cost function) that summarizes the performance or value of a decision and iteratively improves this performance by selecting from among the available

alternatives. The cost function or evaluation function is the measure of adaptation of a certain *organism* (solution) to the environment (problem). An in-depth study of the definition and implementation of *Genetic Algorithms* is presented in Sect. 3.4.4. Note that some strategies commonly used in *Genetic Algorithms*, such as crossover and parallel evolutions are described in the general model as non-essentials.

3.4.2 From Object Detection to Function Optimization

The combination of Genetic Darwin Machines and AdaBoost has been exploited by various authors in recent years. Sedai and Rhee [52] proposed the use of Genetic Algorithms to create subsets of features which adapt better to certain classification tasks and combine specific subsets using AdaBoost. Treptow et al. [53] uses a Genetic Algorithm in order to evolve an extension of the Haar-like features, using AdaBoost to combine the evolved features.

We now explain how the process of learning a detector can be approached as a function optimization problem, using Genetic Darwin Machines for implementation.

Given a training set $\langle(\boldsymbol{\rho}_1, l_1), \ldots, (\boldsymbol{\rho}_m, l_m)\rangle$, where $l_i \in \{-1, +1\}$ is the target value for sample $\boldsymbol{\rho}_i$, the goal of an object detection learning algorithm is to deal with the inference of a strong classifier $H(\boldsymbol{\rho}_i) = l_i$. In the boosting framework, we define a distribution $W = \{w_1, \ldots, w_m\}$ over the training set, where each w_i is the weight associated to the sample $\boldsymbol{\rho}_i$, and $H(\boldsymbol{\rho})$ corresponds to an additive model $H(\boldsymbol{\rho}) = \sum_t \alpha_t h_t(\boldsymbol{\rho})$ where the final decision is a combination of the decisions of several *weak classifiers* $h(\boldsymbol{\rho}) \in \{-1, +1\}$. In contrast to the strong classifier $H(\boldsymbol{\rho})$ where we expect a good performance for any sample $\boldsymbol{\rho}_i$ in the training set, in the case of weak classifier we only expect it to be better than a random decision.

Given $\mathscr{H}$ the set of all possible weak classifiers, $h^{\mathbf{s}} \in \mathscr{H}$ a certain weak classifier defined by parameters $\mathbf{s}$, W the weight distribution of the Adaboost and $\varepsilon(h^{\mathbf{s}}) = Pr_{i\sim W}[h^{\mathbf{s}}(\boldsymbol{\rho}_i) \neq l_i]$ the error function, the regression step consists in finding $\mathbf{s}^*$ that $\varepsilon(h^{\mathbf{s}^*}) \leq \varepsilon(h^{\mathbf{s}}) | \forall h^{\mathbf{s}^*}, h^{\mathbf{s}} \in \mathscr{H}$, where the complexity of finding $\mathbf{s}^*$ depends on the size of $\mathscr{H}$. In Algorithm 5, the evolutionary version for the Discrete Adaboost is shown. Note that the only difference vis-à-vis the classical Discrete AdaBoost is the way in which the weak hypothesis h_t is obtained. The same approach can be used in all the variants of AdaBoost.

Algorithm 5 Evolutionary Discrete Adaboost

Require: A training set $\boldsymbol{\rho}$ of m pairs $(\boldsymbol{\rho}_i, l_i)$, where $\boldsymbol{\rho}_i$ is the ith image and $l_i \in \{-1, +1\}$ is the category of the object present in $\boldsymbol{\rho}_i$, an evolutionary weak learning algorithm (**EvolWeakLearner**) and the maximum number of iterations M.

Initialize the weight vector: $w_i^1 = \frac{1}{m}$ for $i = 1, \ldots, m$.

for $t = 1, \ldots, m$ **do**

Set

$$p^t = \frac{w^t}{\sum_{i=1}^{m} w_i^t}$$

Call the evolutionary method **EvolWeakLearner**, providing it with the distribution $\mathbf{p}^t$. Obtain a hypothesis $h_t{:}\rho \rightarrow \{-1, +1\}$. which minimizes:

$$\varepsilon_t = Pr_{i \sim W_t}[h_t(\rho_i) \neq l_i]$$

Obtain weak hypothesis $h_t(\rho) \mapsto \{-1, +1\}$ with error ε_t
Update:

$$W_{t+1}(i) \leftarrow W_t(i) \times \exp(-l_i \times h_t(\rho_i))$$

Normalize W_{t+1} so

$$\sum_{i=1}^{m} W_{t+1}(i) = 1$$

end for
Ensure: the final hypothesis:

$$H(\rho) = sign\left(\sum_{t=1}^{T} h_t(\rho)\right)$$

Once the evolutionary approach is defined, in order to detect an object we need to define $\mathbf{s}$. In general terms, $\mathbf{s}$ is the set of all the parameters that defines a *weak hypothesis*, and therefore, it is closely related to the features we use to describe the objects and how decisions are made. We can divide $\mathbf{s} = \{s_1, \ldots, s_D\}$ into two different subsets $\mathbf{s}_M = \{s_1, \ldots, s_i\}$ and $\mathbf{s}_F = \{s_{i+1}, \ldots, s_D\}$, which contain the parameters of the decision method and the parameters of the features respectively.

Features parametrization. The first step in object detection is to choose a description method for the images. No restrictions on the type of features are assumed, and so any of the features introduced in Sect. 3.2 or any other descriptors we can imagine can be used to describe objects. In general, certain feature-specific parameters must be defined in order to use each descriptor (i.e. regions of a Haar-like feature or Dissociated Dipole, number of bins in the SIFT and SURF descriptors, etc.). These parameters can be discrete or continuous, predefined or learned, etc. All these parameters are included in $\mathbf{s}_F$. Given an instance for each parameters, we must be able to represent an object, using either a single value (i.e. Haar-like) or a vector (i.e. SIFT).

Classifier parametrization. Once the object is described, a decision must be performed using the descriptor associated to the object. Although there are different approaches for generating hypotheses from descriptors, in general we can consider

threshold based decisions when objects are described by a single value, and some type of distance metric. In the first case, the threshold value and the polarity value must be included as parameters. In the second case, parameters to define the metric or reference points can be added on the parameters. Moreover, once the distance is evaluated, the decision should be generated using a threshold like approach, and thus, the threshold and polarity parameters will also be included as classifier parameters $\mathbf{s}_M$.

3.4.3 Evolutionary Object Detection Approach

At this point, the object detection approach based on a boosting strategy has been redefined as the problem of finding the parameters $\mathbf{s}$ that minimize the weighted error function $\varepsilon(h^{\mathbf{s}})$, that is, an optimization problem. The classical approaches perform an exhaustive search over the parameter space in order to find the best values. Although this can be done when the number of values that parameters can take is reduced, it is impossible in large search spaces.

In the literature we find many different approaches to dealing with optimization problems with large search spaces, most of them based on gradient descent, such as line search methods, normalized steepest methods or the Newton steps method. In these methods, the goal function must be differentiable, and uses the gradient direction to move from a certain solution to a better one. In general, this restriction cannot be guaranteed for the error function, where small changes in the parameters can produce large discontinuities in the error function. In this scenario, the most common optimization methodologies are based on evolutionary computation; in general, the first choice are Genetic Algorithms.

Once a general formulation and its considerations are provided, the first step is to check that using evolutionary strategies we can obtain the same results as in the case of exhaustive methods. Therefore, we need to define a framework in which both approaches can be applied, and compare their learning capabilities. Below we define a parametrization for Haar-like features using decision stumps, analogously to the approach of Viola and Jones.

3.4.3.1 Classifier Implementation

In order to predict the classifier value of a certain object, a decision stump will be used in all the experiments. As we saw in Sect. 3.3, the decision stump is a linear classifier that uses a hyperplane to classify points in a binary problem. The parameters related to a decision stump are hyperplane parameters that codify how the space is divided (see Fig. 3.8) and a polarity value which decides which class is on each side of the hyperplane (see Fig. 3.9).

Given an object descriptor $\mathbf{d} \in R^N$, classifier parameters can be defined as $\mathbf{s}_M = \{s_p \in \{-1, +1\}, \mathbf{s}_h \in R^N\}$, where s_p is the polarity value and $\mathbf{s}_h$ the parameters of the hyperplane.

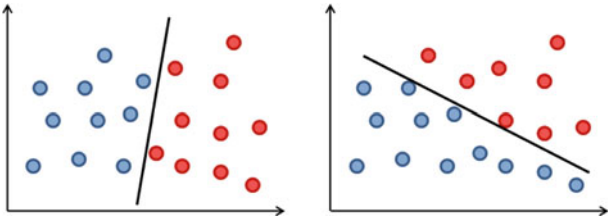


Fig. 3.8 Two different instances of the hyperplane parameters. There are an infinity of possible instances for the hyperplane parameters

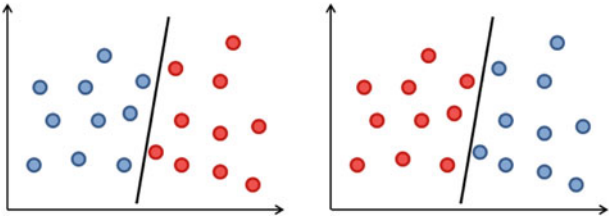


Fig. 3.9 The polarity value codifies which side of the hyperplane corresponds to each class. There are only two possible values for this parameter

3.4.3.2 Features Implementation

Object description will be performed using the Haar-like features used in the work of Viola and Jones (see Sect. 3.2.1). Given the region configuration, a feature is determined with only one of its regions. Figure 3.10 presents the different configurations considered.

Therefore, codifying a Haar-like feature is equivalent to codifying just one rectangle and the configuration. A rectangle can be easily computed by codifying its upper-left corner and its vertical and horizontal sizes (see Fig. 3.11).

Moreover, an additional parameter s_f can be added in order to flip between excitatory and inhibitory regions, obtaining complementary features (excitatory regions become inhibitory and viceversa). Using all the above definitions, the parametrization for a Haar-like feature can be written as $\mathbf{s}_F = \{s_x, s_y, s_w, s_h, s_t, s_f\}$, where $s_x, s_y, s_w, s_h \in N$, $s_t \in \{1, \ldots, 8\}$, and $s_f \in \{-1, 1\}$.

3.4.3.3 Final Model

Once all the parameters involved in the weak hypothesis have been described, we can define the parameter vector $\mathbf{s} = \mathbf{s}_M \cup \mathbf{s}_F = \{s_p, \mathbf{s}_h, s_x, s_y, s_w, s_h, s_t, s_f\}$. Since Haar-like features describe an image with a single value, $\mathbf{s}_h$ has dimension one and corresponds to a threshold value. Although all these parameters must be optimized in order to obtain a good weak hypothesis, not all of them have to be learned using

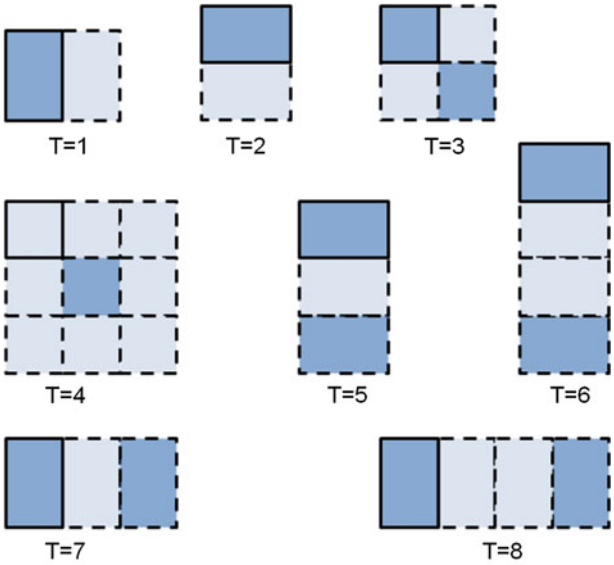
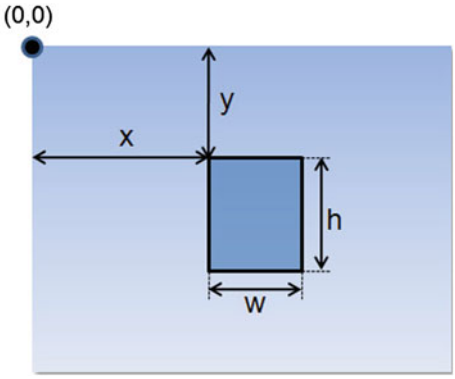


Fig. 3.10 Different Haar-like region configurations. Dashed lines correspond to regions inferred from the given region, which are represented with a continuous line. Darker regions correspond to inhibitory (negative) regions while lighter ones are the excitatory (positive) regions

Fig. 3.11 Parametrization of a region in an image



an evolutionary approach: some of them can either be learned using other methods or fixed to a certain value. For instance, the threshold value of the classifier can be exhaustively found once the feature parameters have been learned with an evolutionary approach.

At this point, a simplification respect to the Viola and Jones approach is applied to our method. The simplification is to use threshold value fixed to zero, known in the literature as Òordinal featuresÓ. This term is related to the use of the sign instead of the value of the feature. In [54], a face detection approach is presented using only the

sign of region intensity differences. The authors demonstrate that by removing the magnitude of the difference the model becomes more stable to illumination changes and image degradation. With this approach, we can remove the threshold value from the parameter vector. Moreover, using ordinal features, it is easy to verify that s_p and s_f have the same effect on the classification value, and therefore, only one of them need be learnt.

Finally, using all above considerations, the problem consists in finding the model parameters:

$$\mathbf{s} = \{s_p, s_x, s_y, s_w, s_h, s_t\} \tag{3.15}$$

which minimize the weighted error function of the AdaBoost, under the following constraints:

$$\begin{aligned} & s_p \in \{-1, 1\} \\ & s_x, s_y \geq 0 \\ & s_w, s_h > 0 \\ & s_x + s_w < W \\ & s_y + s_h < H \\ & s_t \in \{1, \ldots, 8\} \end{aligned} \tag{3.16}$$

where W and H corresponds to the width and height of the learning window.

3.4.4 Object Detection Based on Genetic Algorithms

Now we have formulated the problem, in this section we present the implementation of an evolutionary weak learner based on Genetic Algorithms. The first step is to define a chromosome-based representation for problem variables in Eq. 3.15. Once the encoding is discussed, the adaptation of a certain individual restricted to the constraints in Eq. 3.15 must be included in the evaluation function. Finally, we present an experiment to verify if the resulting schema is able to learn.

3.4.4.1 Chromosome Encoding

Although the encoding of a problem in a chromosome-like representation can be performed in multiple ways, the use of binary representation simplifies mutation and cross-over operators, and it is the method recommended in most problems. Since all our variables are integers, their representation do not need to store decimal positions. In addition, the ranges for each variable are known, and so we can adjust the number of bits in order to minimize values outside the valid ranges.

As a general rule, the number of bits needed to represent a parameter x which take values inside a range [MinVal, . . . , MaxVal], can be calculated as:

Table 3.1 Number of bits to represent each parameter

Parameter name	Initial range	Final range	Number of bits
s_p	$\{-1, +1\}$	$[0_{-1}, 1_{+1}]$	1
s_x	$[0, W-1]$	$[0, W-1]$	$\lceil \log_2 W \rceil$
s_y	$[0, H-1]$	$[0, H-1]$	$\lceil \log_2 H \rceil$
s_w	$[0, W-1]$	$[0, W-1]$	$\lceil \log_2 W \rceil$
s_h	$[0, H-1]$	$[0, H-1]$	$\lceil \log_2 H \rceil$
s_t	$[1, 8]$	$[0, 7]$	3

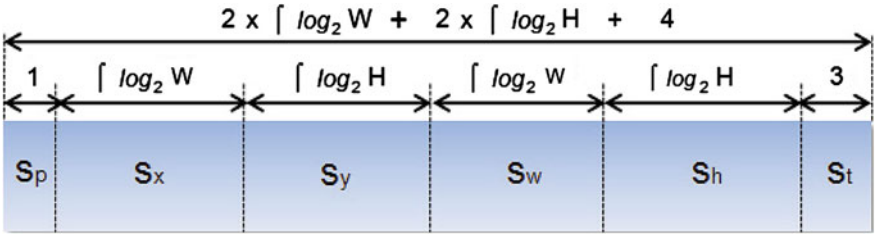


Fig. 3.12 Final chromosome representation for an ordinal Haar-like feature based weak learner. The number of bits is calculated over a learning window of $W \times H$ pixels

$$\text{NumBits} = \lceil \log_2 (\text{MaxValue} - \text{MinValue} + 1) \rceil \tag{3.17}$$

in the case of variables that take values from a set of non-contiguous values, we need to define a range of continuous values and map each value from the original value set to one of the values in the contiguous range defined.

In order to codify the parameter, we need to move it from its original range to a zero based range $\{0, \ldots, \text{MaxVal} - \text{MinVal}\}$ and codify the resulting number in a binary representation. Analogously, when we need to recover the value of a certain parameter, we need to decode the value codified in a binary format to its decimal representation and move this value to its original range.

Using the representation above, we need to differentiate between the parameters which depend on the problem (and for which we therefore need problem information to decode) and those that are independent. Basically, the only problem information we need is the size of the training window, in order to adjust the range of region parameters (see Fig. 3.11). Assuming a training window of $W \times H$ pixels, the ranges and number of bits needed to represent each parameter is shown in Table 3.1.

The final chromosome representation for an individual is shown in Fig. 3.12, where the width of each field in bits and the final number of bits is represented. The last consideration in order to obtain the final representation is how we codify a decimal number to its binary representation. We adopt a codification in terms of Gray codes, which guarantee that similar representations correspond to similar values, although a direct binary representation could be used instead.

3.4.4.2 Evaluation Function

The goal of the evaluation function is to assign an adaptation value to each chromosome. The first step is to decode the chromosome into the original parameter vector, and to evaluate whether those parameters fulfill the constraints or not. When one individual does not fulfill some of the problem constrains, we have two possibilities:

Fixed Error Value. Since we are working on a minimization problem, we can define a value greater than the worst possible value that an individual can achieve. Therefore, in the evolutionary process, the individual will have near null possibilities to be selected for next generations.

Informative Error Value. The idea is similar to the fixed error value approach, but in this case, instead of returning a fixed value, we calculate a value according to how near the individual is to fulfilling the constraints. Therefore, if in some generation of the evolutionary process a large number of bad individuals is present, it is possible to evolve to a good individual. If a fixed value is used, no difference between bad individuals is provided, making evolution to a good individual impossible.

Since in our case the range is adjusted to the possible values, we are more likely to have a large amount of valid individuals, and a fixed error value is used in order to obtain a faster evaluation function.

For valid individuals, the weighted error function is calculated using a training set and the weights associated to each sample. This process is summarized in Algorithm 6. Note that a function $f_{\mathbf{s}}$ must be defined, which calculates the value of the Haar-like feature. This process can be done using an image representation based on integral images [15]. In this implementation, the polarity value s_p is the classifier polarity, therefore, $f_{\mathbf{s}}$ does not consider a polarity value and it is applied over the final hypothesis. Note also that a fixed error value of 1.1 is used to indicate when the individual does not fulfill the constraints. Since the worst possible error value is 1.0, this assigned value is assumed to be always worse than the maximum error value for a valid classifier.

Algorithm 6 Evaluation function for ordinal Haar-like feature chromosome based representation.

Require: A training set X of N pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is the ith image and $y_i \in \{-1, 1\}$ is the category of the object present in $\mathbf{x}_i$, a weight distribution $W = \{w_1, \ldots, w_N\}$ where w_i is the weight of ith sample in X, the chromosome C to be evaluated, and $f_{\mathbf{s}}(x) \mapsto Z$ which calculates the value of the Haar-like feature parameterized in $\mathbf{s} = \{s_x, s_y, s_w, s_h, s_t\}$ over a given image $\mathbf{x}$.

Decode the chromosome C into the parameter vector $\mathbf{s} = \{s_p, s_x, s_y, s_w, s_h, s_t\}$.

if s fulfill the constraints of the problem **then**

  Initialize $\varepsilon \leftarrow 0$

  for $i = 1, \ldots, N$ **do**

    if $f_{\mathbf{s}}(x_i) \geq 0$ **then**

      $h_i \leftarrow +1$

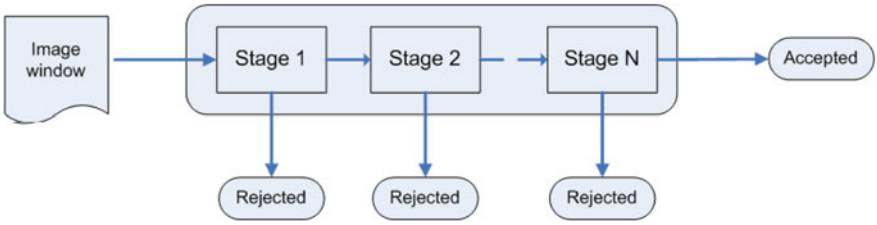


Fig. 3.13 The attentional cascade

else
$h_i \leftarrow -1$
end if
$h_i \leftarrow h_i \times s_p$
if $h_i \neq y_i$ **then**
$\varepsilon \leftarrow \varepsilon + w_i$
end if
end for
else
$\varepsilon \leftarrow 1.1$
end if
Ensure: the final error value ε

3.5 Attentional Cascade

In spite of the optimal image normalization and feature calculation approaches, to obtain good results in a real-world object detection problem, the *strong classifier* learnt by AdaBoost must be a combination of a large number of *weak classifiers*. Since we need to apply this classifier to a huge number of regions (due to the brute force approach explained in Sect. 3.1.1), the final detection time for an image is prohibitive. In order to address this limitation, Viola and Jones introduced a cascade architecture of multiple *strong classifiers*. The underlying idea is to use only the necessary computation cost in order to reject a non-object image, while more complex analyses are performed in more difficult ones (see Fig. 3.13). The regions that reach the last stage of the cascade and are classified as objects, are selected as object regions, and the rest of the regions are rejected.

Each stage analyzes only the objects accepted by the previous stages, and thus, the non-objects are analyzed only until they are rejected by a detector. The number of applied classifiers is reduced exponentially due to the cascade architecture. Notice that with a false alarm of 0.5, half of the input regions to a stage are rejected, while the other half pass to the next stage. In addition, the stages are *strong classifiers* with

weak performance restrictions, and therefore, the number of *weak classifiers* that conform them is smaller.

Learning an *attentional cascade* is not significantly more complex than learning a single *strong classifier*, in the sense that the same algorithm is used, but the training set is changed. Adaboost is used to learn each stage of the cascade, and the rejected samples are replaced by other non-objects which the previous training stages classify by correct objects. The training algorithm for the cascade is shown in Algorithm 7. The final false alarm of the detectors cascade will be $F_{target} = f^n$, where f is the false alarm ratio fixed for each stage of the cascade and n is the number of stages. Analogously, the final hit ratio can be estimated as d^n.

Algorithm 7 Attentional Cascade Training Algorithm

Require: User selected values for f, the maximum acceptable false positive rate per layer, the minimum acceptable detection rate per layer d, and the target overall false positive rate F_{target}. In addition, the algorithm needs a set of positive examples P and a set of negative examples N.
Initialize $F_0 = 1.0$ and $D_0 = 1.0$
Set $i = 1$
while $F_i > F_{target}$ **do**
  $i \leftarrow i + 1$
  $n_i = 0$; $F_i = F_{i-1}$
  while $F_i > f \times F_{i-1}$ **do**
    $n_i \leftarrow n_i + 1$
    Use P and N to train a classifier with n_i features using Adaboost
    Evaluate current cascaded classifier on validation set to determine F_i and D_i
    Decrease threshold for the ith classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects F_i)
  endwhile
  $N \leftarrow 0$
  if $F_i > F_{target}$ **then**
    evaluate the current cascaded detector on the set of non-object images and put any false detections into the set N.
  endif
endwhile
Ensure: A cascade of *strong classifiers*.

Certain aspects of the Algorithm 7 should be highlighted. The first one is that the number of negative samples in N needs to be enormous. At each iteration we need to replace the rejected samples with new samples that are mis-classified by previous stages, and as our classifier increases the rejection rate with the number of stages, each time it is more difficult to find new samples. The number of negative examples can be estimated as:

$$N^{+} = K + \sum_{i=1}^{S-1} \frac{K \times (1 - f)}{f^{i}} \tag{3.18}$$

where S is the number of stages, K the number of negative samples we use to train a stage and f the maximum false rate per stage. That is, if we decide to learn a cascade of 10 stages with $f = 0.5$ and $K = 1,000$, which are common values, we need a negative sample set of about 1,024,000 samples. To avoid creating training sets of this dimension, the negative samples are usually collected on-line from image databases or by analyzing TV programs or other large image sources. Summarizing, to learn a cascade of detectors, usually we need to perform the detection process on multiple images.

References

1. Ullman, S., Sali, E.: Object classification using a fragment-based representation. In: BMVC '00: Proceedings of the First IEEE International Workshop on Biologically Motivated Computer Vision, pp. 73–87. Springer, London (2000)
2. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the International Conference on Computer Vision ICCV, pp. 1150–1157. Corfu.citeseer.ist.psu.edu/lowe99object.html (1999)
3. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: International Conference on Machine Learning, pp. 148–156 (1996)
4. Ullman, S.: High-level vision. object recognition and visual cognition. A Bradford Book, The MIT Press, New York (1996)
5. Abu-Mostafa, Y.S., Pslatis, D.: Optical neural computing. In: Prieditis, A., Russel, S., (eds.), Scientific American, vol. 256, pp. 66–73 (1987)
6. Binford, T.O.: Inferring surfaces from images. Artif. Intell. **17**, 205–244 (1981)
7. Besl, P.J., Jain, R.C.: Tree-dimentional object recognition. Comput. Surv. **17**, 75–145 (1985)
8. Duin, R.P.W., Roli, F., de Ridder, D.: A note on core research issues for statistical pattern recognition. Pattern Recogn. Lett. **23**(4), 493–499 (2002)
9. Levine, M.D.: Feature extraction: a survey. Proc. IEEE **57**(8), 1391–1407 (1969)
10. Marr, D.: Vision. W.H Freeman, (1982)
11. Marr, D., Hildreth, E.: Theory of edge detection. Proc. R. Soc. Lond. B Biol. Sci. **207**, 187–217 (1980)
12. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
13. Deriche, R.: Using canny's criteria to derive a recursively implemented optimal edge detector. Int. J. Comput. Vis. **1**(6), 167–187 (1987)
14. Fleck, M.M.: Multiple widths yield reliable finite differences (computer vision). IEEE Trans. Pattern Anal. Mach. Intell. **14**(4), 412–429 (1992)
15. Viola P., Jones, M.: Rapid object detection using a boosted cascade of simple features. Proc. 2001 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn **1**, 511–518 (2001)
16. Croiser, A., Esteban, D., Galand, C.: Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. In: Proceedings of the International Symposium Information on Circuits and Systems (1976)
17. Crochiere, R., Weber, S., Flanagan, J.: Digital coding of speech in sub-bands. Bell Syst. Tech. J. **55**, 1069–1085 (1976)

18. Vetterli, M., LeGall, D.: Perfect reconstruction FIR filter banks: some properties and factorizations. IEEE Trans. Acoust. Speech Signal Process. **37**(7), 1057–1071 (1989)
19. Lienhart, R., Maydt, J.: An extended set of haar-like features for rapid object detection. In: Proceedings of the International Conference on Image Processing, pp. 900–903. Rochester, USA, September2002. IEEE. (2002)
20. Balas, B.J., Sinha, P.: Dissociated dipoles: Image representation via non-local comparisons. In: Annual Meeting of the Vision Sciences Society, Sarasota, FL (2003)
21. Balas, B.J., Sinha, P.: Receptive field structures for recognition. Neural Comput. **18**(3), 497–520 (2006)
22. van der Walt, C., Barnard, E.: Data characteristics that determine classifier performance. In: 17th Annual Symposium of the Pattern Recognition Association of South Africa, December (2006)
23. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
24. Minsky, M., Papert, S.: Perceptrons pp. 157–169 (1988)
25. Ng, A., Jordan, M.: On discriminative vs. generative classifiers: a comparison of logistic regression and naive bayes. Adv. Neural Inf. Process. Syst. **14**, 841–848 (2002)
26. Jaakkola, T.S., Haussler, D.: Exploiting generative models in discriminative classifiers. In: Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II, pp. 487–493. MIT Press, Cambridge (1999)
27. Hillel, A.B., Weinshall, D., Hertz, T.: Efficient learning of relational object class models. In: Tenth IEEE International Conference on Computer Vision 2005. ICCV 2005, vol. 2, pp. 1762–1769, Oct 2005
28. Fritz, M., Leibe, B., Caputo, B., Schiele, B.: Integrating representative and discriminant models for object category detection. In: Tenth IEEE International Conference on Computer Vision 2005. ICCV 2005, vol. 2, pp. 1363–1370, Oct 2005
29. Perronnin, F., Dance, C.: Fisher kernels on visual vocabularies for image categorization. In: IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR '07, pp. 1–8, June 2007
30. Bosch, A., Zisserman, A., Muñoz, X.: Scene classification using a hybrid generative/discriminative approach. IEEE Trans. Pattern Anal. Mach. Intell. **30**(4), 712–727 (2008)
31. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
32. Dietterich, T.G.: (2002) Ensemble learning. In: Ed. Arbib, M.A. (ed.) The Handbook of Brain Theory and Neural Networks. The MIT Press, Cambridge
33. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
34. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)
35. Freund, Y.: Boosting a weak learning algorithm by majority. Inf. Comput. **121**(2), 256–285 (1995)
36. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Ann. Stat. **28**(2), 337–407 (2000)
37. Friedman, J., Stuetzle, W.: Projection pursuit regression. J. Am. Stat. Assoc. **76**(report) 817–823 (1981)
38. Buja, A., Haste, T., Tibshirani, R.: Linear smoothersand additive models. Ann. Stat. **17**, 453–555 (1989)
39. Mita, T., Kaneko, T., Stenger, B., Hori, O.: Discriminative feature co-occurrence selection for object detection. IEEE Trans. Pattern Anal. Mach. Intell. **30**(7), 1257–1269 (2008)
40. Freeman, W.T., Adelson, E.H.: The design and use of steerable filters. IEEE Trans. Pattern Anal. Mach. Intell. **13**(9), 891–906 (1991)
41. Blackmore, S.: The Meme Machine (Popular Science). Oxford University Press, USA (2000)
42. Calvin, W.H.: The six essentials? minimal requirements for the darwinian bootstrapping of quality (1997)

43. Calvin, W.H.: Visual features of intermediate complexity and their use in classification. Nature **330**, 33–34 (1987)
44. Barricelli, N.A.: Esempi numerici di processi di evoluzione. Methodos, 45–68 (1954)
45. Fogel, D.B.: Nils barricelli—artificial life, coevolution, self-adaptation. IEEE Comput. Intell. Mag. **1**(1), 41–45 (2006)
46. Fraser, A.S.: Simulation of genetic systems by automatic digital computers. I. Introduction. Aust. J. Biol. Sci. **10**, 484–491 (1957)
47. Fraser, A.S., Hansche, P.E.: Simulation of genetic systems. major and minor loci. In: Proceedings of the 11th International Congress on Genetics, vol. 3, pp 507–516. Oxford (1965)
48. Fraser, A.S., Burnell, D.: Computer Models in Genetics. McGraw-Hill, New York (1970)
49. Crosby, J.L.: Computer Simulation in Genetics. Wiley, New York (1973)
50. Rechenberg I., (1971) Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Ph.D thesis, Technical University of Berlin, Department of Process Engineering
51. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, New York (1981)
52. Sedai, S., Rhee, P.K.: Bio-inspired adaboost method for efficient face recognition. In: Frontiers in the Convergence of Bioscience and Information Technologies 2007. FBIT 2007, pp. 715–718, Oct 2007
53. Treptow, A., Zell, A.: Combining adaboost learning and evolutionary search to select features for real-time object detection. In: Congress on Evolutionary Computation 2004. CEC2004, vol. 2, pp. 2107–2113 (2004)
54. Thoresz, K., Sinha, P.: Qualitative representations for recognition. J. Vis. **1**(3), 298–298 (2001)