CHAPTER 11

Algorithms for Real-Time Object Detection in Images

MILOS STOJMENOVIC

11.1 INTRODUCTION

11.1.1 Overview of Computer Vision Applications

The field of Computer Vision (CV) is still in its infancy. It has many real-world applications, and many breakthroughs are yet to be made. Most of the companies in existence today that have products based on CV can be divided into three main categories: auto manufacturing, computer circuit manufacturing, and face recognition. There are other smaller categories of this field that are beginning to be developed in industry such as pharmaceutical manufacturing applications and traffic control. Auto manufacturing employs CV through the use of robots that put the cars together. Computer circuit manufacturers use CV to visually check circuits in a production line against a working template of that circuit. CV is used as quality control in this case. The third most common application of CV is in face recognition. This field has become popular in the last few years with the advent of more sophisticated and accurate methods of facial recognition. Applications of this technology are used in security situations like checking for hooligans at sporting events and identifying known thieves and cheats in casinos. There is also the related field of biometrics where retinal scans, fingerprint analysis, and other identification methods are conducted using CV methods.

Traffic control is also of interest because CV software systems can be applied to already existing hardware in this field. By traffic control, we mean the regulation or overview of motor traffic by means of the already existing and functioning array of police monitoring equipment. Cameras are already present at busy intersections, highways, and other junctions for the purposes of regulating traffic, spotting problems, and enforcing laws such as running red lights. CV could be used to make all of these tasks automatic.

Handbook of Applied Algorithms: Solving Scientific, Engineering and Practical Problems
Edited by Amiya Nayak and Ivan Stojmenović

11.2 MACHINE LEARNING IN IMAGE PROCESSING

AdaBoost and support vector machines (SVMs) are, among others, two very popular and conceptually similar machine learning tools for image processing. They are both based on finding a set of hyperplanes to separate the sets of positive and negative examples. Current image processing culture involving machine learning for real-time performance almost exclusively uses AdaBoost instead of SVMs. AdaBoost is easier to program and has proven itself to work well. There are very few papers that deal with real-time detection using SVM principles. This makes the AdaBoost approach a better choice for real-time applications. A number of recent papers, using both AdaBoost and SVMs, confirm the same, and even apply a two-phase process. Most windows are processed in the first phase by AdaBoost, and in the second phase, an SVM is used on difficult cases that could not be easily eliminated by AdaBoost. This way, the real-time constraint remains intact.

Le and Satoh [16] maintain that "The pure SVM has constant running time of 554 windows per second (WPS) regardless of complexity of the input image, the pure AdaBoost (cascaded with 37 layers—5924 features) has running time of 640, 515 WPS." If a pure SVM approach was applied to our test set, it would take $17,500,000/554 \approx 9$ h of pure run time to test the 106 images. It would take roughly 2 min to process an image of size 320×240. Thus, Lee and Satoh [16] claim that cascaded AdaBoost is 1000 times faster than SVMs. A regular AdaBoost with 30 features was presented in the works by Stojmenovic [24,25]. A cascaded design cannot speed up the described version by more than 30 times. Thus, the program in the works by Stojmenovic [24,25] is faster than SVM by over $1000/30 > 30$ times.

Bartlett et al. [3] used both AdaBoost and SVMs for their face detection and facial expression recognition system. Although they state that "AdaBoost is significantly slower to train than SVMs," they only use AdaBoost for face detection, and it is based on Viola and Jones' approach [27]. For the second phase, facial expression recognition on detected faces, they use three approaches: AdaBoost, SVMs, and a combined one (all applied on Gabor representation), and reported differences within 3 percent of each other. They gave a simple explanation for choosing AdaBoost in the face detection phase, "The average number of features that need to be evaluated for each window is very small, making the overall system very fast" [3]. Moreover, each of these features is evaluated in constant time, because of integral image preprocessing. That performance is hard to beat, and no other approach in image processing literature for real-time detection is seriously considered now.

AdaBoost was proposed by Freund and Schapire [8]. The connection between AdaBoost and SVMs was also discussed by them [9]. They even described two very similar expressions for both of them, where the difference was that the Euclidean norm was used by SVMs while the boosting process used Manhattan (city block) and maximum difference norms. However, they also list several important differences. Different norms may result in very different margins. A different approach is used to efficiently search in high dimensional spaces. The computation requirements are different. The computation involved in maximizing the margin is mathematical pro-

gramming, that is, maximizing a mathematical expression given a set of inequalities. The difference between the two methods in this regard is that SVM corresponds to *quadratic programming*, while AdaBoost corresponds only to *linear programming* [9]. Quadratic programming is more computationally demanding than linear programming [9].

AdaBoost is one of the approaches where a "weak" learning algorithm, which performs just slightly better than random guessing, is "boosted" into an arbitrarily accurate "strong" learning algorithm. If each weak hypothesis is slightly better than random, then the training error drops exponentially fast [9]. Compared to other similar learning algorithms, AdaBoost is adaptive to the error rates of the individual weak hypotheses, while other approaches required that all weak hypotheses need to have accuracies over a parameter threshold. It is proven [9] that AdaBoost is indeed a boosting algorithm in the sense that it can efficiently convert a weak learning algorithm into a strong learning algorithm (which can generate a hypothesis with an arbitrarily low error rate, given sufficient data).

Freund and Schapire [8] state "Practically, AdaBoost has many advantages. It is fast, simple, and easy to program. It has no parameters to tune (except for the number of rounds). It requires no prior knowledge about the weak learner and so can be flexibly combined with *any* method for finding weak hypotheses. Finally, it comes with a set of theoretical guarantees given sufficient data and a weak learner that can reliably provide only moderately accurate weak hypotheses. This is a shift in mind set for the learning-system designer: instead of trying to design a learning algorithm that is accurate over the entire space, we can instead focus on finding weak learning algorithms that only need to be better than random. On the other hand, some caveats are certainly in order. The actual performance of boosting on a particular problem is clearly dependent on the data and the weak learner. Consistent with theory, boosting can fail to perform well given insufficient data, overly complex weak hypotheses, or weak hypotheses that are too weak. Boosting seems to be especially susceptible to noise."

Schapire and Singer [23] described several improvements to Freund and Schapire's [8] original AdaBoost algorithm, particularly in a setting in which hypotheses may assign confidences to each of their predictions. More precisely, weak hypotheses can have a range over all real numbers rather than the restricted range $[-1, +1]$ assumed by Freund and Schapire [8]. While essentially proposing a general fuzzy AdaBoost training and testing procedure, Howe and Coworkers [11, 34] do not describe any specific variant, with concrete fuzzy classification decisions. We propose in this chapter a specific variant of fuzzy AdaBoost. Whereas Freund and Schapire [8] prescribe a specific choice of weights for each classifier, Schapire and Singer [23] leave this choice unspecified, with various tunings. Extensions to multiclass classifications problems are also discussed. [Q1]

In practice, the domain of successful applications of AdaBoost in image processing is any set of objects that are typically seen from the same angle and have a constant orientation. AdaBoost can successfully be trained to identify any object if this object is viewed from an angle similar to that in the training set. Practical real-world examples that have been considered so far include faces, buildings, pedestrians, some animals,

and cars. The backbone of this research comes from the face detector work done by Viola et al. [27]. All subsequent papers that use and improve upon AdaBoost are inspired by it.

11.3 VIOLA AND JONES' FACE DETECTOR

The face detector proposed by Viola and Jones [27] was the inspiration for all other AdaBoost applications thereafter. It involves different stages of operation. The training of the AdaBoost machine is the first part and the actual use of this machine is the second part. Viola and Jones' contributions come in the training and assembly of the AdaBoost machine. They had three major contributions: integral images, combining features to find faces in the detection process, and use of a cascaded decision process when searching for faces in images. This machine for finding faces is called cascaded AdaBoost by Viola and Jones [27]. Cascaded AdaBoost is a series of smaller AdaBoost machines that together provide the same function as one large AdaBoost machine, yet evaluate each subwindow more quickly, which results in real-time performance. To understand cascaded AdaBoost, regular AdaBoost will have to be explained first. The following sections will describe Viola and Jones' face detector in detail.

Viola and Jones' machine takes in a square region of size equal to or greater than 24×24 pixels as input and determines whether the region is a face or is not a face. This is the smallest size of window that can be declared a face according to Viola and Jones. We use such a machine to analyze the entire image, as illustrated in Figure 11.1. We pass every subwindow of every scale through this machine to find all subwindows that contain faces. A sliding window technique is therefore used. The window is shifted 1 pixel after every analysis of a subwindow. The subwindow grows in size 10 percent every time all of the subwindows of the previous size were exhaustively searched. This means that the window size grows exponentially at a rate of $(1.1)^p$,

FIGURE 11.1 Subwindows of an image.

where p is the number of scales. In this fashion, more than 90 percent of faces of all sizes can be found in each image.

As with any other machine learning approach, the machine must be trained using positive and negative examples. Viola and Jones used 5000 positive examples of randomly found upright, forward-facing faces and 10,000 negative examples of any other nonface objects as their training data. The machine was developed by trying to find combinations of common attributes, or features of the positive training set that are not present in the negative training set.

The library of positive object (head) representatives contains face pictures that are concrete examples. That is, faces are cropped from larger images, and positive examples are basically closeup portraits only. Moreover, positive images should be of the same size (that is, when cut out of larger images, they need to be scaled so that all positive images are of the same size). Furthermore, all images are frontal upright faces. The method is not likely to work properly if the faces change orientation.

11.3.1 Features

An image *feature* is a function that maps an image into a number or a vector (array). Viola and Jones [27] used only features that map images into numbers. Moreover, they used some specific types of features, obtained by selecting several rectangles within the training set, finding the sum of pixel intensities in each rectangle, assigning a positive or negative sign and/or weight to each sum, and then summing them. The pixel measurements used by Viola and Jones were the actual grayscale intensities of pixels. If the areas of the dark (positive sign) and light (negative sign) regions are not equal, the weight of the lesser region is raised. For example, feature 2.1 in Figure 11.2 has a twice greater light area than a dark one. The area of the dark rectangle in this case would be multiplied by 2 to normalize the feature. The main problem is to find which of these features, among the thousands available, would best distinguish positive and negative examples, and how to combine them into a learning machine.

Figure 11.2 shows the set of basic shapes used by Viola and Jones [27]. Adding features to the feature set can increase the accuracy of the AdaBoost machine at the cost of additional training time. Each of the shapes seen in Figure 11.2 is scaled and translated anywhere in the test images, consequently forming features. Therefore, each feature includes a basic shape (as seen in Fig. 11.2), its translated position in the image, and its scaling factors (height and width scaling). These features define the separating ability between positive and negative sets. This phenomenon is illustrated in Figure 11.3. Both of the features seen in Figure 11.3 (each defined by its position and scaling factors) are derived from the basic shapes in Figure 11.2.

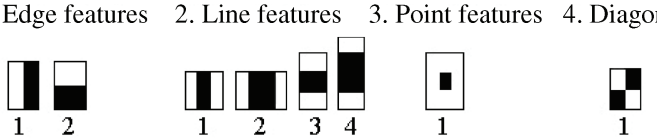

FIGURE 11.2 Basic shapes that generate features by translation and scaling.

FIGURE 11.3 First and second features in Viola and Jones face detection.

Figure 11.3 shows the first and second features selected by the program [27]. Why are they selected? The first feature shows the difference in pixel measurements for the eye area and area immediately below it. The "black" rectangle covering the eyes is filled with predominantly darker pixels, whereas the area immediately beneath the eyes is covered with lighter pixels. The second feature also concentrates on the eyes, showing the contrast between two rectangles containing eyes and the area between them. This feature corresponds to feature 2.1 in Figure 11.2 where the light and dark areas are inverted. This is not a separate feature; it was drawn this way in Figure 11.3 to better depict the relatively constant number obtained by this feature when it is evaluated in this region on each face.

11.3.2 Weak Classifiers (WCs)

A WC is a function of the form $h(x, f, s, \theta)$, where x is the tested subimage, f is the feature used, s is the sign (+ or −), and θ is the threshold. The sign s defines on what side of the threshold the positive examples are located. Threshold θ is used to establish whether a given image passes a classifier test in the following fashion: when feature f is evaluated on image x, the resulting number is compared to threshold θ to determine how this image is categorized by the given feature. The equation is given as $sf(x)<s\theta$. If the equation evaluates true, the image is classified as positive. The function $h(x, f, s, \theta)$ is then defined as follows: $h(x, f, s, \theta) = 1$ if $sf(x) < s\theta$ and 0 otherwise. This is expected to correspond to positive and negative examples, respectively. There are a few ways to determine the threshold θ. In the following example, the green numbers are considered to be the positive set, and the red letters are considered to be the negative set. The threshold is set to be the black vertical line after the "7" since at this location overall classification error is minimal. All of the positions are tried, and the one with minimal error is selected. The error function that is used is the number of misclassifications divided by the total number of examples. The array of evaluated feature values is sorted by the values of $f(x)$, and it shows positive examples as 1, 2, 3, ... in green and negatives as A, B, C, D, ... in red. The error of the threshold selected below is $3/17 \approx 0.17$.

1 6 8 4 I 3 5 7 | B E A 9 C 2 F D G H

In general, the threshold is found to be the value θ that best separates the positive and negative sets. When a feature f is selected as a "good" distinguisher of images between positive and negative sets, its value would be similar for images in the positive set and different for all other images. When this feature is applied to an individual image, a number $f(x)$ is generated. It is expected that values $f(x)$ for positive and negative images can be separated by a threshold value of θ.

It is worthy to note that a single WC needs only to produce results that are slightly better than chance to be useful. A combination of WCs is assembled to produce a strong classifier as seen in the following text.

11.3.3 Strong Classifiers

A strong classifier is obtained by running the AdaBoost machine. It is a linear combination of WCs. We assume that there are T WCs in a strong classifier, labelled $h_1, h_2, \ldots, h_T$, and each of these comes with its own weight labeled $\alpha_1, \alpha_2, \ldots, \alpha_T$. Tested image x is passed through the succession of WCs $h_1(x), h_2(x), \ldots, h_T(x)$, and each WC assesses if the image passed its test. The assessments are discrete values: $h_i(x) = 1$ for a pass and $h_i(x) = 0$ for a fail. $\alpha_i(x)$ are in the range $[0, +\infty]$. Note that $h_i(x) = h_i(x, f_i, s_i, \theta_i)$ is abbreviated here for convenience. The decision that classifies an image as being positive or negative is made by the following inequality:

$$\alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots + \alpha_T h_T(x) > \alpha/2 \quad \text{where}\, \alpha = \sum_{\mathrm{i}=1}^{\mathrm{T}} \alpha_{\mathrm{i}}.$$

From this equation, we see that images that pass a weighted average of half of the WC tests are cataloged as positive. It is therefore a weighted voting of selected WCs.

11.3.4 AdaBoost: Meta Algorithm

In this section we explain the general principles of the AdaBoost (an abbreviation of Adaptive Boosting) learning strategy [8]. First, a huge (possibly hundreds of thousands) "panel" of experts is identified. Each expert, or WC, is a simple threshold-based decision maker, which has a certain accuracy. The AdaBoost algorithm will select a small panel of these experts, consisting of possibly hundreds of WCs, each with a weight that corresponds to its contribution in the final decision. The expertise of each WC is combined in a classifier so that more accurate experts carry more weight.

The selection of WCs for a classifier is performed iteratively. First, the best WC is selected, and its weight corresponds to its overall accuracy. Iteratively, the algorithm identifies those records in the training data that the classifier built so far was unable to capture. The weights of the misclassified records increase since it becomes more important to correctly classify them. Each WC might be adjusted by changing its threshold to better reflect the new weights in the training set. Then a single WC is selected, whose addition to the already selected WCs will make the greatest contribution to improving the classifier's accuracy. This process continues iteratively

until a satisfactory accuracy is achieved, or the limit for the number of selected WCs is reached. The details of this process may differ in particular applications, or in particular variants of the AdaBoost algorithm.

There exist several AdaBoost implementations that are freely available in Weka (Java-based package http://www.cs.waikato.ac.nz/ml) and in R (http://www.r-project.org). Commercial data mining toolkits that implement AdaBoost include TreeNet, Statistica, and Virtual Predict. We did not use any of these packages for two main reasons. First, our goal was to achieve real-time performance, which restricted the choice of programming languages. Next, we have modified the general algorithm to better suit our needs, which required us to code it from scratch.

AdaBoost is a general scheme adaptable to many classifying tasks. Little is assumed about the learners (WCs) used. They should merely perform only a little better than random guesses in terms of error rates. If each WC is always better than a chance, then AdaBoost can be proven to converge to a perfectly accurate classifier (no training error). Boosting can fail to perform if there is insufficient data or if WCs are overly complex. It is also susceptible to noise. Even when the same problem is being solved by different people applying AdaBoost, the performance greatly depends on the training set being selected and the choice of WCs (that is, features).

In the next subsection, the details of the AdaBoost training algorithm, as used by Viola and Jones [27], will be given. In this approach, positive and negative training sets are separated by a cascade of classifiers, each constructed by AdaBoost. Real time performance is achieved by selecting features that can be computed in constant time. The training time of the face detector appears to be slow, even taking months according to some reports. Viola and Jones' face finding system has been modified in literature in a number of articles. The AdaBoost machine itself was modified in literature in several ways.

11.3.5 AdaBoost Training Algorithm

We now show how to create a classifier with the AdaBoost machine. It follows the algorithm given in the work by Viola and Jones [27]. The machine is given images $(x_1, y_1), \ldots, (x_q, y_q)$ as input, where $y_i = 1$ or 0 for positive and negative examples, respectively. In iteration t, the ith image is assigned the weight $w(t, i)$, which corresponds to the importance of that image for a good classification. The initial weights are $w(1, i) = 1/(2p), 1/(2n)$, for $y_i = 0$ or 1, respectively, where n and p are the numbers of negatives and positives, respectively, $q = p + n$. That is, all positive images have equal weight, totaling $\frac{1}{2}$, and similarly for all negative images. The algorithm will select, in step t, the tth feature f, its threshold value θ, and its direction of inequality $s(s = 1\, or\, -1)$. The classification function is $h(x, f, s, \theta) = 1$ (declared positive) if $sf(x)<s\theta$, and 0 otherwise (declared negative).

The expression $|h(x_i, f, s, \theta) - y_i|$ indicates whether or not $h(x, f, s, \theta)$ correctly classified image x_i. Its value is 0 for correct classification, and 1 for incorrect classification. The sum $\sum_{i=1}^{N} w(t, i) \times |h(x_i, f, s, \theta) - y_i|$ then represents the weighted misclassification error when using $h(x, f, s, \theta)$ as the feature-based classifier. The goal is to minimize that sum when selecting the next WC.

We revisit the classification of numbers and letters example to illustrate the assignment of weights in the training procedure. We assume that feature 1 classifies the example set in the order seen below. The threshold is chosen to be just after the "7" since this position minimizes the classification error. We will call the combination of feature 1 with its threshold WC 1. We notice that "*I*", "9," and "2" were incorrectly classified. The number of incorrect classifications determines the weight α_1 of this classifier. The fewer errors that it makes, the heavier the weight it is awarded.

The weights of the incorrectly classified examples (*I*, 9, and 2) are increased before finding the next feature in an attempt to find a feature that can better classify cases that are not easily sorted by previous features. We assume that feature two orders the example set as seen below.

Setting the threshold just after the "2" minimizes the error in classification. We notice that this classifier makes more mistakes in classification than its predecessor. This means that its weight, α_2, will be less that α_1. The weights for elements "*E*", "*I*," "8," and "4" are increased. These are the elements that were incorrectly classified by WC 2. The actual training algorithm will be described in pseudocode below.

For t=1 to T do:

Normalize the weights $w(t, i)$, by dividing each of them with their sum (so that the new sum of all weights becomes 1);
$swp \leftarrow$ sum of weights of all positive images
$swn \leftarrow$ sum of weights of all negative images
(* note that $swp + swn = 1$ *)

FOR each candidate feature *f*, find $f(x_i)$ and $w(t, i)^* f(x_i)$, $i = 1, \ldots, q$.

- Consider records $(f(x_i), y_i, w(t, i))$. Sort these records by the $f(x_i)$ field with mergesort, in increasing order. Let the obtained array of the $f(x_i)$ field be $g_1, g_2, \ldots, g_q$. The corresponding records are $(g_j, status(j), w'(j)) = (f(x_i), y_i, w(t, i))$, where $g_j = f(x_i)$. That is, if the jth element g_j is equal to ith element from the original array $f(x_i)$ then $status(j) = y_i$ and $w'(j) = w(t, i)$.

(*Scan through the sorted list, looking for threshold θ and direction s that minimizes the error $e(f, s, \theta)$*)

$sp \leftarrow 0; sn \leftarrow 0$; (*weight sums for positives/negatives below a considered threshold *)
$emin \leftarrow$ minimal total weighted classification error
If $swn < swp$ **then** $\{emin \leftarrow swn; smin \leftarrow 1; \theta min \leftarrow g_n + 1$ (*all declared positive*)

else $\{\ emin \leftarrow swp; smin \leftarrow 1; \theta min \leftarrow g_1 - 1\ \}$ (*all declared negative *)

For $j \leftarrow 1$ **to** q-1 **do** $\{$

If $status(j) = 1$ **then** $sp \leftarrow sp + w'(j)$ **else** $sn \leftarrow sn + w'(j)$

$\theta \leftarrow (g_j + g_{j+1})/2$

If $sp + swn - sn < emin$ **then** $\{emin \leftarrow sp + swn - sp; smin \leftarrow -1; \theta min \leftarrow \theta\ \}$

If $sn + swp - sp < emin$ **then** $\{emin \leftarrow sn + swp - sp; smin \leftarrow 1; \theta min \leftarrow \theta\ \}\ \}$

EndFOR

Set $s_t \leftarrow smin$; set $\theta_t \leftarrow \theta min$(*s and θ of current stage are determined*)

$\beta_t \leftarrow emin/(1 - emin)$;

$\alpha_T \leftarrow -\log(\beta_t)$ (* α_T is the output of AdaBoost for the second part*)

Update the weights for the next weak classifier, if needed:

$$w(t+1, i) \leftarrow w(t, i)\beta_t^{1-e}, \text{where } e = \left\{ \begin{matrix} 0 \text{ if } x_i \text{is correctly classified bycurrent } h_t \\ 1 \text{otherwise} \end{matrix} \right\}$$

EndFor;

AdaBoost therefore assigns large weights with each good classification and small weights with each poor function. The selection of the next feature depends on selections made for previous features.

11.3.6 Cascaded AdaBoost

Viola and Jones [27] also described the option of designing a cascaded AdaBoost. For example, instead of one AdaBoost machine with 100 classifiers, one could design 10 such machines with 10 classifiers in each. In terms of precision, there will not be much difference, but testing for most images will be faster [27]. One particular image is first tested on the first classifier. If declared as nonsimilar, it is not tested further. If it cannot be rejected, then it is tested with the second machine. This process continues until either one machine rejects an image, or all machines "approve" it, and similarity is confirmed. Figure 11.4 illustrates this process. Each classifier seen in Figure 11.4 comprises one or more features. The features that define a classifier are chosen so that their combination eliminates as much as possible all negative images that are

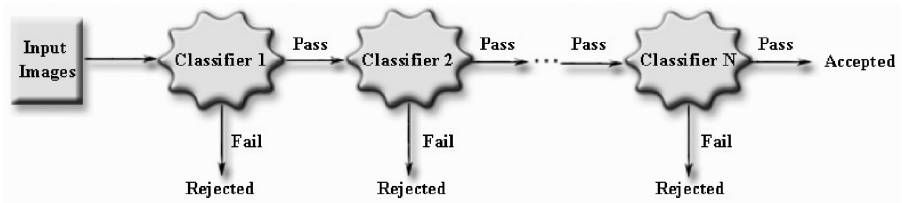

FIGURE 11.4 Cascaded decision process.

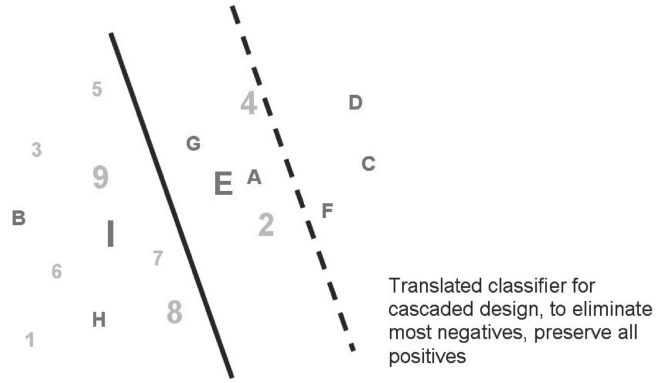

FIGURE 11.5 Concept of a classifier.

passed through this classifier, while at the same time accepting nearly 100 percent of the positives. It is desirable that each classifier eliminates at least 50 percent of the remaining negatives in the test set. A geometric progression of elimination is created until a desired threshold of classification is attained. The number of features in each classifier varies. It typically increases with the number of classifiers added. In Viola and Jones' face finder cascade, the first classifiers had 2, 10, 25, 25, and 50 features, respectively. The number of features grew very rapidly afterward. Typical numbers of features per classifier ranged in the hundreds. The total number of features used was roughly 6000 in Viola and Jones' application.

Figure 11.5 will help explain the design procedure of the cascaded design process. We revisit the letters and numbers example in our efforts to show the development of a strong classifier in the cascaded design. At the stage seen in Figure 11.5, we assume to have two WCs with weights α_1 and α_2. Together these two WCs make a conceptual hyperplane depicted by the solid dark blue line. In actuality, this line is not a hyperplane (in this case a line in two-dimensional space), but a series of orthonormal dividers. It is, however, conceptually easier to explain the design of a strong classifier in a cascade if we assume that WCs form hyperplanes.

So far in Figure 11.5, we have two WCs where the decision inequality would be of the form $\alpha_1 h_1(x) + \alpha_2 h_2(x) > \alpha/2$, where $\alpha = \alpha_1 + \alpha_2$. At this stage, the combination of the two WCs would be checked against the training set to see if they have a 99 percent detection rate (this 99 percent is a design parameter). If the detection rate is below the desired level, the threshold $\alpha/2$ is replaced with another threshold γ such that the detection rate increases to the desired level. This has the conceptual effect of translating the dark blue hyperplane in Figure 11.5 to the dotted line. This also has a residual effect of increasing the false positive rate. At the same time, once we are happy with the detection rate, we check the false positive rate of the shifted threshold detector. If this rate is satisfactory, for example, below 50 percent (also a design parameter), then the construction of the classifier is completed. The negative examples that were correctly identified by this classifier are ignored from further consideration by future classifiers. There is no need to consider them if they are already success-

fully eliminated by a previous classifier. In Figure 11.5, "D", "C," and "F" would be eliminated from future consideration if the classifier construction were completed at this point.

11.3.7 Integral Images

One of the key contributions in the work by Viola and Jones [27] (which is used and/or modified by Levi and Weiss [17], Luo et al. [19], etc.) is the introduction of a new image representation called the "integral image," which allows the features used by their detector to be computed very quickly.

In the preprocessing step, Viola and Jones [27] find the sums $ii(a, b)$ of pixel intensities $i(a', b')$ for all pixels (a', b') such that $a' \leq a, b' \leq b$. This can be done in one pass over the original image using the following recurrences:

$$s(a, b) = s(a, b - 1) + i(a, b),$$

$$ii(a, b) = ii(a - 1, b) + s(a, b),$$

where $s(a, b)$ is the cumulative row sum, $s(a, -1) = 0$, and $ii(-1, b) = 0$. In prefix sum notation, the expression for calculating the integral image values is

$$ii(a, b) = \sum_{a' \leq a, b' \leq b} i(a', b').$$

Figure 11.6 shows an example of how the "area" for rectangle "D" can be calculated using only four operations. Let the area mean the sum of pixel intensities of a rectangular region. The preprocessing step would have found the values of corners 1, 2, 3, and 4, which are in effect the areas of rectangles A, $A + B$, $A + C$, and $A + B + C + D$, respectively. Then the area of rectangle D is $= (A + B + C + D) +$

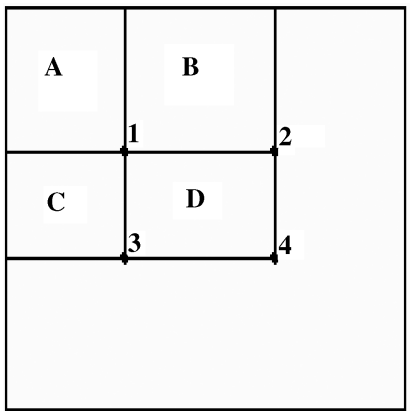

FIGURE 11.6 Integral image.

$(A) - (A + B) - (A + C) =$ "4" + "1" − "2" − "3". Jones and Viola [12] built one face detector for each view of the face. A decision tree is then trained to determine the viewpoint class (such as right profile or rotated 60 degrees) for a given window of the image being examined. The appropriate detector for that viewpoint can then be run instead of running all of the detectors on all windows.

11.4 CAR DETECTION

The most popular example of object detection is the detection of faces. The fundamental application that gave credibility to AdaBoost was Viola and Jones' real-time face finding system [27]. AdaBoost is the concrete machine learning method that was used by Viola and Jones to implement the system. The car detection application was inspired by the work of Viola and Jones. It is based on the same AdaBoost principles, but a variety of things, both in testing and in training, were adapted and enhanced to suit the needs of the CV system described in the works by Stojmenovic [24,25]. The goal of this chapter is to analyze the capability of current machine learning techniques of solving similar image retrieval problems. The "capability" of the system includes real-time performance, a high detection rate, low false positive rate, and learning with a small training set. Of particular interest are cases where the training set is not easily available, and most of it needs to be manually created.

As a particular case study, we will see the application of machine learning to the detection of rears of cars in images [24,25]. Specifically, the system is able to recognize cars of a certain type such as a Honda Accord 2004. While Hondas have been used as an instance, the same program, by just replacing the training sets, could be used to recognize other types of cars. Therefore, the input should be an arbitrary image, and the output should be that same image with a rectangle around any occurrence of the car we are searching for (see Fig. 11.7). The system will work by directly searching for an occurrence of the positive in the image, while treating all subwindows of the image the same way. It will not first search for a general vehicle class and then specify the model of the vehicle. This is a different and much more complicated task that is not easily solvable by machine learning. Any occurrence of a rectangle around a part of the image that is not a rear of a Honda Accord 2004 is considered a negative detection.

The image size in the testing set is arbitrary, while the image sizes in both the negative and positive training sets are the same. Positive training examples are the rears of Hondas. The data set was collected by taking pictures of Hondas (about

FIGURE 11.7 Input and output of the testing procedure.

300 of them) and other cars. The training set was actually manually produced by cropping and scaling positives from images to a standard size. Negative examples in the training set include any picture, of the same fixed size, that cannot be considered as a rear of a Honda. This includes other types of cars, as close negatives, for improving the classifier's accuracy. Thus, a single picture of a larger size contains thousands of negatives. When a given rectangle around a rear of a Honda is slightly translated and scaled, one may still obtain a positive example, visually and even by the classifier. That is, a classifier typically draws several rectangles at the back of each Honda. This is handled by a separate procedure that is outside the machine learning framework.

In addition to precision of detection, the second major main goal of the system was real-time performance. The program should quickly find all the cars of the given type and position in an image, in the same way that Viola and Jones finds all the heads. The definition of "real time" depends on the application, but generally speaking the system delivers an answer for testing an image within a second. The response time depends on the size of the tested image, thus what appears to be real-time for smaller images may not be so for larger ones.

Finally, this object detection system is interesting since it is based on a small number of training examples. Such criteria are important in cases where training examples are not easily available. For instance, in the works by Stojmenovic [24,25], photos of back views of a few hundred Honda Accords and other cars were taken manually to create training sets, since virtually no positive images were found on the Internet. In such cases, it is difficult to expect that one can have tens of thousands of images readily available, which was the case for the face detection problem. The additional benefit of a small training set is that the training time is reduced. This enabled us to perform a number of training attempts, adjust the set of examples, adjust the set of features, test various sets of WCs, and otherwise analyze the process by observing the behavior of the generated classifiers.

11.4.1 Limitations and Generalizations of Car Detection

Machine learning methods were applied in the work by Stojmenovic [24] in an attempt to solve the problem of detecting rears of a particular car type since they appear to be appropriate given the setting of the problem. Machine learning in similar image retrieval has proven to be reliable in situations where the target object does not change orientation. As in the work of Viola and Jones [27], cars are typically found in the same orientation with respect to the road. The situation Stojmenovic [24] is interested in is the rear view of cars. This situation is typically used in monitoring traffic since license plates are universally found at the rears of vehicles.

The positive images were taken such that all of the Hondas have the same general orthogonal orientation with respect to the camera. Some deviation occurred in the pitch, yaw, and roll of these images, which might be why the resulting detector has such a wide range of effectiveness. The machine that was built is effective for the following deviations in angles: pitch -15°; yaw -30° to 30°; and roll -15° to 15°. This means that pictures of Hondas taken from angles that are off by the stated amounts

are still detected by the program. Yaw, pitch, and roll are common jargon in aviation describing the three degrees of freedom the pilot has to maneuver an aircraft.

Machine learning concepts in the CV field that deal with retrieving similar objects within images are generally faced with the same limitations and constraints. All successful real-time applications in this field have been limited to successfully finding objects from only one view and one orientation that generally does not vary much. There have been attempts to combine several strong classifiers into one machine, but discussing only individual strong classifiers, we conclude that they are all sensitive to variations in viewing angle. This limits their effective range of real-world applications to things that are generally seen in the same orientation. Typical applications include faces, cars, paintings, posters, chairs, some animals, and so on. The generalization of such techniques to problems that deal with widely varying orientations is possible only if the real-time performance constraint is lifted. Another problem that current approaches are faced with is the size of the training sets. It is difficult to construct a sufficiently large training database for rare objects.

11.4.2 Fast AdaBoost Based on a Small Training Set for Car Detection

This section describes the contributions and system [24] for detecting cars in real time. Stojmenovic [24] has revised the AdaBoost-based learning environment, for use in their object recognition problem. It is based on some of the ideas from literature, and some new ideas, all combined into a new machine.

The feature set used in the work Stajmenovic [24,25] initially included most of the feature types used by Viola and Jones [27] and Lienhart [14]. The set did not include rotated features [14], since the report on their usefulness was not convincing. Edge orientation histogram (EOH)-based features [17] were considered a valuable addition and were included in the set. New features that resemble the object being searched for, that is, custom-made features, were also added.

Viola and Jones [27] and most followers used weight-based AdaBoost, where the training examples receive weights based on their importance for selecting the next WC, and all WCs are consequently retrained in order to choose the next best one. Stojmenovic [24,25] states that it is better to rely on the Fast AdaBoost variant [30], where all of the WCs are trained exactly once, at the beginning. Instead of the weighted error calculation, Stojmenovic [24] believes that it is better to select the next WC to be added as the one that, when added, will make the best contribution (measured as the number of corrections made) to the already selected WCs. Each selected WC will still have an associated weight that depends on its accuracy. The reason for selecting the Fast AdaBoost variant is to achieve an $O(\log q)$ time speed-up in the training process, believing that the lack of weights for training examples can be compensated for by other "tricks" that were applied to the system.

Stojmenovic [24,25] has also considered a change in the AdaBoost logic itself. In existing logic, each WC returns a binary decision (0 or 1) and can therefore be referred to as the *binary WC*. In the machine proposed by Schapire and Singer [23], each WC will return a number in the range $[-1, 1]$ instead of returning a binary decision (0 or 1), after evaluating the corresponding example. Such a WC will be referred to as a *fuzzy*

FIGURE 11.8 Positive training examples.

WC. Evaluation of critical cases is often done by a small margin of difference from the threshold. Although the binary WC may not be quite certain about evaluating a particular feature against the adopted threshold (which itself is also determined heuristically, therefore is not fully accurate), the current AdaBoost machine assigns the full weight to the decision on the corresponding WC. Stojmenovic [24,25] therefore described an AdaBoost machine based on a fuzzy WC. More precisely, the described system proposes a specific function for making decisions, while Schapire [23] left this choice unspecified. The system produces a "doubly weighted" decision. Each WC receives a corresponding weight α, then each decision is made in the interval $[-1, 1]$. The WC then returns the product of the two numbers, that is, a number in the interval $[-\alpha, \alpha]$ as its "recommendation." The sum of all recommendations is then considered. If positive, the majority opinion is that the example is a positive one. Otherwise, the example is a negative one.

11.4.3 Generating the Training Set

All positives in the training set were fixed to be 100×50 pixels in size. The entire rear view of the car is captured in this window. Examples of positives are seen in Figure 11.8. The width of a Honda Accord 2004 is 1814 mm. Therefore, each pixel in each training image represents roughly $1814/100 = 18.14$ mm of the car.

A window of this size was chosen due to the fact that a typical Honda is unrecognizable to the human eye at lower resolutions; therefore, a computer would find it impossible to identify accurately. Viola and Jones used similar logic in determining their training example dimensions. All positives in the training set were photographed at a distance of a few meters from the camera. Detected false positives were added in the negative training set (bootstrapping), in addition to a set of manually selected examples, which included backs of other car models. The negative set of examples perhaps has an even bigger impact on the training procedure than the positive set. All of the positive examples look similar to the human eye. It is therefore not important to overfill the positive set since all of the examples there should look rather similar. The negative set should ideally combine a large variety of different images. The negative images should vary with respect to their colors, shapes, and edge quantities and orientations.

11.4.4 Reducing Training Time by Selecting a Subset of Features

Viola and Jones' faces were 24×24 pixels each. Car training examples are 100×50 pixels each. The implications of having such large training examples are immense from a memory consumption point of view. Each basic feature can be scaled in both height and width, and can be translated around each image. There are seven basic

features used by Viola and Jones. They generated a total of 180,000 WCs [27]. Stojmenovic [24,25] also used seven basic features (as described below), and they generate a total of approximately 6.5 million WCs! Each feature is shifted to each position in the image and for every vertical and horizontal scale. By shifting our features by 2 pixels in each direction (instead of 1) and making scale increments of 2 during the training procedure, we were able to cut this number down to approximately 530,000, since every second position and scale of feature was used. In the initial training of the WCs, each WC is evaluated based on its cumulative error of classification (CE). The cumulative error of a classifier is CE = (false positives + number of missed examples)/total number of examples. WCs that had a CE that was greater than a predetermined threshold were automatically eliminated from further consideration. Details are given in the works by Stojmenovic [24,25].

11.4.5 Features Used in Training for Car Detection

Levi and Weiss [17] stress the importance of using the right features to decrease the sizes of the training sets, and increase the efficiency of training. A good feature is the one that separates the positive and negative training sets well. The same ideology is applied here in hopes of saving time in the training process. Initially, all of Viola and Jones' features were used in combination with the dominant edge orientation features proposed by Levi and Weiss [17] and the redness features proposed by Luo et al. [19]. It was determined that the training procedure never selected any of Viola and Jones' grayscale features to be in the strong classifier at the end of training. This is a direct consequence of the selected positive set. Hondas come in a variety of colors and these colors are habitually in the same relative locations in each positive case. The most obvious example is the characteristic red tail lights of the Honda accord. The redness features were included specifically to be able to use the redness of the tail lights as a WC. The training algorithm immediately exploited this distinguishing feature and chose the red rectangle around one of the tail lights as one of the first WCs in the strong classifier. The fact that the body of the Honda accord comes in its own subset of colors presented problems to the grayscale set of Viola and Jones' features. When these body colors are converted to a grayscale space, they basically cover the entire space. No adequate threshold can be chosen to beneficially separate positives from negatives. Subsequently, all of Viola and Jones' features were removed due to their inefficiency.

The redness features we refer to are taken from the work of Luo et al. [19]. More details are given in the works by Stojmenovic [24,25]. Several dominant edge orientation features were used in the training algorithm. To get a clearer idea of what edge orientation features are, we will first describe how they are made. Just as their name suggests, they arise from the orientation of the edges of an image. A Sobel gradient mask is a matrix used in determining the location of edges in an image. A typical mask of this sort is of size 3×3 pixels. It has two configurations, one for finding edges in the x-direction and the other for finding edges in the y-direction of source images ([7], p. 165). These two matrices, $\mathbf{h_x}$ and $\mathbf{h_y}$ (shown in Figs. 11.9 and 11.10), are known as the Sobel kernels.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

FIGURE 11.9 Kernel $\mathbf{h}_y$.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

FIGURE 11.10 Kernel $\mathbf{h}_x$.

Figure 11.9 shows the typical Sobel kernel for determining vertical edges (y-direction), and Figure 11.10 shows the kernel used for determining horizontal edges (x-direction). Each of these kernels is placed over every pixel in the image. Let P be the grayscale version of the input image. Grayscale images are determined from RGB color images by taking a weighted sampling of the red, green, and blue color spaces. The value of each pixel in a grayscale image was found by considering its corresponding color input intensities, and applying the following formula: $0.212671 \times R + 0.715160 \times G + 0.072169 \times B$, which is a built in function in OpenCV, which was used in the implementation.

Let $P(x, y)$ represent the value of the pixel at point (x, y) and $I(x, y)$ is a 3×3 matrix of pixels centered at (x, y). Let X and Y represent output edge orientation images in the x and y directions, respectively. X and Y are computed as follows:

$$\begin{aligned} X(i, j) = \mathbf{h}_x \cdot I(i, j) &= -P(i-1, j-1) + P(i+1, j-1) - 2P(i-1, j) \\ &\quad +2P(i+1, j) - P(i-1, j+1) + P(i+1, j+1), \end{aligned}$$

$$\begin{aligned} Y(i, j) = \mathbf{h}_y \cdot I(i, j) &= -P(i-1, j-1) - 2P(i, j-1) - P(i+1, j-1) \\ &\quad +P(i-1, j+1) + 2P(i, j+1) + P(i+1, j+1) \end{aligned}$$

A Sobel gradient mask was applied to each image to find the edges of that image. Actually, a Sobel gradient mask was applied both in the x-dimension, called $X(i, j)$, and in the y-dimension, called $Y(i, j)$. A third image, called $R(i, j)$, of the same dimensions as X, Y, and the original image, was generated such that $R(i, j) = \sqrt{X(i, j)^2 + Y(i, j)^2}$. The result of this operation is another grayscale image with a black background and varying shades of white around the edges of the objects in the image. The image $R(i, j)$ is called a Laplacian image in image processing literature, and values $R(i, j)$ are called Laplacian intensities. One more detail of our implementation is the threshold that was placed on the intensities of the Laplacian values. We used a threshold of 80 to eliminate the faint edges that are not useful. A similar threshold was employed in the work by Levi and Weiss [17].

The orientations of each pixel are calculated from the $X(i, j)$ and $Y(i, j)$ images. The orientation of each pixel $R(i, j)$ in the Laplacian image is found as

$$\text{orientation}(i,j) = \arctan(\text{Y}(i,j), \text{X}(i,j)) \times 180/\pi.$$

This formula gives the orientation of each pixel in degrees. The orientations are divided into six bins so that similar orientations can be grouped together. The whole circle is divided into six bins. Bin shifting (rotation of all bins by 15°) is applied

to better capture horizontal and vertical edges. Details are given in the work by Stojmenovic [24].

11.5 NEW FEATURES AND APPLICATIONS

11.5.1 Rotated Features and Postoptimization

Lienhart and Maydt [14] add a set of classifiers (Haar wavelets) to those already proposed by Viola and Jones. Their new classifiers are the same as those proposed by Viola and Jones, but they are all rotated 45°. They claim to gain a 10 percent improvement in the false detection rate at any given hit rate when detecting faces. The features used by Lienhart were basically Viola and Jones' entire set rotated 45° counterclockwise. He added two new features that resembled the ones used by Viola and Jones, but they too failed to produce notable gains.

However, there is a postoptimization stage involved with the training process. This postoptimization stage is credited with over 90 percent of the improvements claimed by this paper. Therefore, the manipulation of features did not impact the results all that much; rather the manipulation of the weights assigned to the neural network at the end of each stage of training is the source of gains. OpenCV supports the integral image function on 45° rotated images since Lienhart was on the development team for OpenCV.

11.5.2 Detecting Pedestrians

Viola et al. [29] propose a system that finds pedestrians in motion and still images. Their system is based on the AdaBoost framework. It considers both motion information and appearance information. In the motion video pedestrian finding system, they train AdaBoost on pairs of successive frames of people walking. The intensity differences between pairs of successive images are taken as positive examples. They find the direction of motion between two successive frames, and also try to establish if the moving object can be a person. If single images are analyzed for pedestrians, no motion information is available, and just the regular implementation of AdaBoost seen for faces is applied to pedestrians. Individual pedestrians are taken as positive training examples. It does not work as well as the system that considers motion information since the pedestrians are relatively small in the still pictures, and also relatively low resolution (not easily distinguishable, even by humans). AdaBoost is easily confused in such situations. Their results suggest that the motion analysis system works better than the still image recognizer. Still, both systems are relatively inaccurate and have high false positive rates.

11.5.3 Detecting Penguins

Burghardt et al. [5] apply the AdaBoost machine to the detection of African penguins. These penguins have a unique chest pattern that AdaBoost can be trained on. They

were able to identify not only penguins in images, but distinguish between individual penguins as well. Their database of penguins was small and taken from the local zoo. Lienhart's [14] adaptation of AdaBoost was used with the addition of an extra feature: the empty kernel. The empty kernel is not a combination of light and dark areas, but rather only a light area so that AdaBoost may be trained on "pure luminance information." AdaBoost was used to find the chests of penguins, and other methods were used to distinguish between different penguins. Their technique did not work very well for all penguins. They gave no statistics concerning how well their approach works. This is another example of how the applications of AdaBoost are limited to very specialized problems.

11.5.4 Redeye Detection: Color-Based Feature Calculation

Luo et al. [19] introduce an automatic redeye detection and correction algorithm that uses machine learning in the detection of red eyes. They use an adaptation of AdaBoost in the detection phase of redeye instances. Several novelties are introduced in the machine learning process. The authors used, in combination with existing features, color information along with aspect ratios (width to height) of regions of interest as trainable features in their AdaBoost implementation.

Viola and Jones [27] used only grayscale intensities, although their solution to face detection could have used color information. Finding red eyes in photos means literally finding red oval regions, which absolutely requires the recognition of color. Another unique addition in their work is a set of new features similar to those proposed by Viola and Jones [27], yet designed specifically to easily recognize circular areas. We see these feature templates in Figure 11.11. It is noticeable that the feature templates presented in this figure have three distinct colors: white, black, and gray. The gray and black regions are taken into consideration when feature values are calculated. Each of the shapes seen in Figure 11.11 is rotated around itself or reflected creating eight different positions. The feature value of each of the eight positions is calculated, and the minimum and maximum of these results are taken as output from the feature calculation.

The actual calculations are performed based on the RGB color space. The pixel values are transformed into a one-dimensional space before the feature values are calculated in the following way: Redness $= 4R - 3G + B$. This color space is biased toward the red spectrum (which is where red eyes occur). This redness feature was used in the car detection system [24].

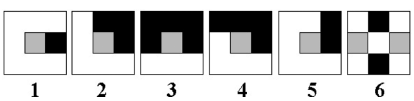

FIGURE 11.11 Features for redeye detection.

11.5.5 EOH-Based Features

Levi and Weiss [17] add a new perspective on the training features proposed by Viola and Jones [27]. They also detect upright, forward-facing faces. Among other contributions in their work [17], their most striking revelation was adding an edge orientation feature that the machine can be trained on. They also experimented with mean intensity features, which means taking the average pixel intensity in a rectangular area. These features did not produce good results in their experiments and were not used in their system. In addition to the features used by Viola and Jones [27], which considered sums of pixel intensities, Levi and Weiss [17] create features based on the most prevalent orientation of edges in rectangular areas. There are obviously many orientations available for each pixel but they are reduced to eight possible rotations for ease of comparison and generalization. For any rectangle, many possible features are extracted. One set of features is the ratio of any two pairs of the eight EOHs [17]. There are therefore 8 choose $2 = 28$ possibilities for such features. Another feature that is calculated is the ratio of the most dominant EOH in a rectangle to the sum of all other EOHs. Levi and Weiss [17] claim that using EOHs, they are able to achieve higher detection rates at all training database sizes.

Their goal was to achieve similar or better performance of the system to Viola and Jones' work while substantially reducing training time. They primarily achieve this because EOH gives good results with a much smaller training set. Using these orientation features, symmetry features are created and used. Every time a WC was added to their machine, its vertically symmetric version was added to a parallel yet independent cascade. Using this parallel machine architecture, the authors were able to increase the accuracy of their system by 2 percent when both machines were run simultaneously on the test data. The authors also mention detecting profile faces. Their results are comparable to those of other proposed systems but their system works in real-time and uses a much smaller training set.

11.5.6 Fast AdaBoost

Wu et al. [30] propose a training time performance increase over Viola and Jones' training method. They change the training algorithm in such a way that all of the features are tested on the training set only once (per each classifier). The ith classifier ($1 \leq i \leq N$) is given as input the desired minimum detection rate d_i and the maximum false positive rate $\mathrm{fp_i}$. These rates are difficult to predetermine because the performance of the system varies greatly. The authors start with optimistic rates and gradually decrease expectations after including over 200 features until the criterion is met. Each feature is trained so that it has minimal false positive rate $\mathrm{fp_i}$. The obtained WCs h_j are sorted according to their detection rates. The strong classifier is created by incrementally adding the feature that either increases the detection rate (if it is $<d_i$) or minimizes false positives until desired levels are achieved in both categories. Since the features are tested independently, the weights of the positive and negative training examples that are incorrectly classified are not changed. The decision of the

ensemble classifier is formed by a majority vote of the WCs (that is, each WC has equal weight in the work by wu et al. [30]). The authors state that using their model of training, the desired detection rate was more difficult to achieve than the desired false positive rate. To improve this defect, they introduce asymmetric feature selection. They incorporated a weighting scheme into the selection of the next feature. They chose weights of 1 for false positive costs and λ for false negative costs. λ is the cost ratio between false negatives and false positives. This setup allows the system to add features that increase the detection rate early on in the creation of the strong classifier.

Wu et al. [30] state that their method works almost as well as that of Viola and Jones when applied to the detection of upright, forward-facing faces. They however achieve a training time that is two orders of magnitude faster than that of Viola and Jones. This is achieved in part by using a training set that was much smaller than Viola and Jones' [27], yet generated similar results.

We will now explain the time complexity of both Viola and Jones' [27] and Wu's [30] training methods. There are three factors to consider when finding the time complexity of each training procedure: the number of features F, the number of WCs in a classifier T, and the number of examples in the training set q. One feature in one example takes $O(1)$ time because of integral images. One feature on q examples takes $O(q)$ time to evaluate, and $O(q \log(q))$ to sort and find the best WC. Finding the best feature takes $O(Fq \log(q))$ time. Therefore, the construction of the classifier takes $O(TFq \log q)$. Wu's [30] method takes $O(Fq \log q)$ time to train all of the classifiers in the initial stage. Testing each new WC while assuming that the summary votes of all classifiers are previously stored would take $O(q)$ time. It would then take $O(Fq)$ time to select the best WC. Therefore, it takes $O(TqF)$ time to chose T WC. We deduce that it would take $O(Fq \log q + TqF)$ time to complete the training using the methods described by Wu et al. [30]. The dominant term in the time complexity of Wu's [30] algorithm is $O(TqF)$. This is order $O(\log q)$ times faster than the training time for Viola and Jones' method [27]. For a training set of size $q = 10,000$, $\log_2 q \approx 13$. For the same size training sets, Wu's [30] algorithm would be 13 times faster to train, not a 100 times as claimed by the authors. The authors compared training times to achieve a predetermined accuracy rate, which requires fewer training items than Viola and Jones' method [27]. Froba et al. [13] elaborate on a face verification system. The goal of this system is to be able to recognize a particular person based on his/her face. The first step in face verification is face detection. The second is to analyze the detected sample and see if it matches one of the training examples in the database. The mouths of input faces into the system are cropped because the authors claim that this part of the face varies the most and produces unstable results. They however include the forehead since it helps with system accuracy. The authors use the same training algorithm for face detection as Viola and Jones [27], but include a few new features. They use AdaBoost to do the training, but the training set is cropped, which means that the machine is trained on slightly different input than Viola and Jones [27]. The authors mention that a face is detectable and verifiable with roughly 200 features that are determined by AdaBoost during the training phase. The actual verification or recognition step of individual people based on these images is done

using information obtained in the detection step. Each face that is detected is made up of a vector of 200 numbers that are the evaluations of the different features that made up that face. These numbers more or less uniquely represent each face and are used as a basis of comparison of two faces. The sum of the weighted differences in the feature values between the detected face and the faces of the individual people in the database is found and compared against a threshold as the verification step. This is a sort of nearest-neighbor comparison that is used in many other applications.

11.5.7 Downhill Feature Search

McCane and Novins [20] described two improvements over the Viola and Jones' [27] training scheme for face detection. The first one is a 300-fold speed improvement over the training method, with an approximately three times slower execution time for the search. Instead of testing all features at each stag (exhaustive search), McCane and Novins [20] propose an optimization search, by applying a "downhill search" approach. Starting from a feature, a certain number of neighboring features are tested next. The best one is selected as the next feature, and the procedure is repeated until no improvement is possible. The authors propose to use same size adjacent features (e.g., rectangles "below" and "above" a given one, in each of the dimensions that share one common edge) as neighbors. They observe that the work by Viola and Jones [27] applies AdaBoost in each stage to optimize the overall error rate, and then, in a postprocessing step, adjust the threshold to achieve the desired detection rate on a set of training data. This does not exactly achieve the desired optimization for each cascade step, which needs to optimize the false positive rate subject to the constraint that the required detection rate is achieved. As such, sometimes adding a level in an AdaBoost classifier actually increases the false positive rate. Further, adding new stages to an AdaBoost classifier will eventually have no effect when the classifier improves to its limit based on the training data. The proposed optimization search allows it to add more features (because of the increased speed), and to add more parameters to the existing features, such as allowing some of the subsquares in a feature to be translated. The second improvement in the work by McCane and Novins [20] is a principled method for determining a cascaded classifier of optimal speed. However, no useful information is reported, except the guideline that the false positive rate for the first cascade stage should be between 0.5 and 0.6. It is suggested that exhaustive search [27] could be performed at earlier stages in the cascade, and replaced by optimized search [20] in later stages.

11.5.8 Bootstrapping

Sung and Poggio [22] applied the following "bootstrap" strategy to constrain the number of nonface examples in their face detection system. They incrementally select only those nonface patterns with high utility value. Starting with a small set of non-face examples, they train their classifier with current database examples and run the face detector on a sequence of random images (we call this set of images a "semitesting"

set). All nonface examples that are wrongly classified by the current system as faces are collected and added to the training database as new negative examples. They notice that the same bootstrap technique can be applied to enlarge the set of positive examples. In the work by Bartlett et al. [3], a similar bootstrapping technique was applied. False alarms are collected and used as nonfaces for training the subsequent strong classifier in the sequence, when building a cascade of classifiers.

Li et al. [18] observe that the classification performance of AdaBoost is often poor when the size of the training sample set is small. In certain situations, there may be unlabeled samples available and labeling them is costly and time consuming. They propose an active learning approach, to select the next unlabeled sample that is at the minimum distance from the optimal AdaBoost hyperplane derived from the current set of labeled samples. The sample is then labeled and entered into the training set. Abramson and Freund [1] employ a selective sampling technique, based on boosting, which dramatically reduces the amount of human labor required for labeling images. They apply it to the problem of detecting pedestrians from a video camera mounted on a moving car. During the boosting process, the system shows subwindows with close classification scores, which are then labeled and entered into positive and negative examples. In addition to features from the work by Viola and Jones [27], authors also use features with "control points" from the work by Burghardt and Calic [2].

Zhang et al. [31] empirically observe that in the later stages of the boosting process, the nonface examples collected by bootstrapping become very similar to the face examples, and the classification error of Haar-like feature based WC is thus very close to 50 percent. As a result, the performance of a face detection method cannot be further improved. Zhang et al. [31] propose to use global features, derived from Principal component analysis (PCA), in later stages of boosting, when local features do not provide any further benefit. They show that WCs learned from PCA coefficients are better boosted, although computationally more demanding. In each round of boosting, one PCA coefficient is selected by AdaBoost. The selection is based on the ability to discriminate faces and nonfaces, not based on the size of coefficient.

11.5.9 Other AdaBoost Based Object Detection Systems

Treptow et al. [26] described a real-time soccer ball tracking system, using the described AdaBoost based algorithm [27]. The same features were used as in the work by Viola and Jones [27]. They add a procedure for predicting ball movement.

Cristinacce and Cootes [6] extend the global AdaBoost-based face detector by adding four more AdaBoost based algorithms that detect the left eye, right eye, left mouth corner, and right mouth corner within the face. Their placement within the face is probabilistically estimated. Training face images are centered at the nose and some flexibility in position of other facial parts with a certain degree of rotation is allowed in the main AdaBoost face detector, because of the help provided by the four additional machines.

FloatBoost [31,32] differs from AdaBoost in a step where the removal of previously selected WCs is possible. After a new WC is selected, if any of the previously added

classifiers contributes to error reduction less than the latest addition, this classifier is removed. This results in a smaller feature set with similar classification accuracy. FloatBoost requires about a five times longer training time than AdaBoost. Because of the reduced set of selected WCs, Zhang et al. [31,32] built several face recognition learning machines (about 20), one for each of face orientation (from upfront to profiles). They also modified the set of features. The authors conclude that the method does not have the highest accuracy.

Howe [11] looks at boosting for image retrieval and classification, with comparative evaluation of several algorithms. Boosting is shown to perform significantly better than the nearest-neighbor approach. Two boosting techniques that are compared are based on feature- and vector-based boosting. Feature-based boosting is the one used in the work by Viola and Jones [27]. Vector-based boosting works differently. First, two vectors, toward positive and negative examples, are determined, both as weighted sums (thus corresponding to a kind of average value). A hyperplane bisecting the angle between them is used for classification. The dot product of the tested example that is orthogonal to that hyperplane is used to make a decision. Comparisons are made on five training sets containing suns, churches, cars, tigers, and wolves. The features used are color histograms, correlograms (probabilities that a pixel B at distance x from pixel A has the same color as A), stairs (patches of color and texture found in different image locations), and Viola and Jones' features. Vector boosting is shown to be much faster than feature boosting for large dimensions. Feature-based boosting gave better results than vector based when the number of dimensions in the image representation is small.

Le and Satoh [15] observe AdaBoost advantages and drawbacks, and propose to use it in the first two stages of the classification process. The first stage is a cascaded classifier with subwindows of size 36×36, the second stage is a cascaded classifier with subwindows of size 24×24. The third stage is an SVM classifier for greater precision. Silapachote et al. [21] use histograms of Gabor and Gaussian derivative responses as features for training and apply them for face expression recognition with AdaBoost and SVM. Both approaches show similar results and AdaBoost offers important feature selections that can be visualized.

Barreto et al. [4] described a framework that enables a robot (equipped with a camera) to keep interacting with the same person. There are three main parts of the framework: face detection, face recognition, and hand detection. For detection, they use Viola and Jones's features [27] improved by Lienhart and Maydt [14]. The eigenvalues and PCA are used in the face recognition stage of the system. For hand detection, they apply the same techniques used for face detection. They claim that the system recognizes hands in a variety of positions. This is contrary to the claims made by Kolsch et al. [13] who built one cascaded AdaBoost machine for every typical hand position and even rotation.

Kolsch and Turk [16,17] describe and analyze a hand detection system. They create a training set for each of the six posture/view combinations from different people's right hands. Then both training and validation sets were rotated and a classifier was trained for each angle. In contrast to the case of the face detector, they found poor accuracy with rotated test images for as little as a 4° rotation. They then added rotated

example images to the same training set, showing that up to 15° of rotation can be efficiently detected with one detector.

11.5.10 Binary and Fuzzy Weak Classifiers

Most AdaBoost implementations that we found in literature use binary WCs, where the decision of a WC is either accept or reject, which will be valued at $+1$ and -1, respectively (and described in Chapter 2). We also consider *fuzzy WCs* [23] as follows. Instead of making binary decisions, fuzzy WCs make a 'weighted' decision, as a real number in the interval $[-1, 1]$. Fuzzy WCs can then simply replace binary WCs as basic ingredients in the training and testing programs, without affecting the code or structure of the other procedures.

A *fuzzy WC* is a function of the form $h(x, f, s, \theta, \theta_{mn}, \theta_{mx})$ where x is the tested sub image, f is the feature used, s is the sign ($+$ or $-$), θ is the threshold, and θ_{mn} and θ_{mx} are the adopted extreme values for positive and negative images. The sign s defines on what side the threshold the positive examples are located. Threshold θ is used to establish whether a given image passes a classifier test in the following fashion: when feature f is applied to image x, the resulting number is compared to threshold θ to determine how this image is categorized by the given feature. The equation is given below

$$sf(x) < s\theta.$$

If the equation evaluates true, the image is classified as positive. The function $h(x, f, s, \theta, \theta_{mn}, \theta_{mx})$ is then defined as follows. If the image is classified as positive ($sf(x) < s\theta$) then $h(x, f, s, \theta, \theta_{mn}, \theta_{mx}) = \min(1, |(f(x) - \theta)/(\theta_{mn} - \theta)|)$. Otherwise $h(x, f, s, \theta, \theta_{mn}, \theta_{mx}) = \max(-1, -|(f(x) - \theta)/(\theta_{mx} - \theta)|)$. This definition is illustrated in the following example.

Let $s = 1$, thus the test is $f(x) < \theta$. One way to determine θ_{mn} and θ_{mx} (used in our implementation) is to find the minimal feature value of the positive examples (example "1" seen here), and maximal negative value (example "H" seen here) and assign them to θ_{mn} and θ_{mx}, respectively. If $s = -1$, then the definitions are modified accordingly. Suppose that an image is evaluated to be around the letter "I" in the example (it could be exactly the letter "I" in the training process or a tested image at runtime). Since $f(x) < \theta$, the image is estimated as positive. The degree of confidence in the estimation is $|(f(x) - \theta)/(\theta_{mn} - \theta)|$, which is about 0.5 in the example. If the ratio is > 1, then it is replaced by 1. The result of the evaluation is then $h(x, f, s, \theta, \theta_{mn}, \theta_{mx}) = 0.5$, which is returned as the recommendation.

11.5.11 Strong Classifiers

A strong classifier is obtained by running the AdaBoost machine. It is a linear combination of WCs. We assume that there are T WCs in a strong classifier, labeled

$h_1, h_2, \ldots, h_T$, and each of these comes with its own weight labeled $\alpha_1, \alpha_2, \ldots, \alpha_T$. The tested image x is passed through the succession of WCs $h_1(x), h_2(x), \ldots, h_T(x)$, and each WC assesses if the image passed its test. In case of binary WCs, the recommendations are either -1 or 1. In case of using fuzzy WCs, the assessments are values ρ in the interval $[-1, 1]$. Values ρ from interval $(0, 1]$ correspond to a pass (with confidence ρ) and in the interval $[0, -1]$ a fail. Note that $h_i(x) = h_i(x, f_i, s_i, \theta_i, \theta_{mn}, \theta_{mx})$ is abbreviated here for convenience (parameters θ_{mn} and θ_{mx} are needed only for fuzzy WCs). The decision that classifies an image as being positive or negative is made by the following inequality:

$$\alpha = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_T h_T(x) > \delta.$$

From this equation, we see that images that pass (binary or weighted) weighted recommendations of the WC tests are cataloged as positive. It is therefore a (simple or weighted) voting of selected WCs. The value α also represents the confidence of overall voting. The error is expected to be minimal when $\delta = 0$, and this value is used in our algorithm. The α values are determined once at the beginning of the training procedure for each WC, and are not subsequently changed. Each $\alpha_i = -\log(e_i/(1 - e_i))$. Each e_i is equal to the cumulative error of the WC.

11.6 CONCLUSIONS AND FUTURE WORK

It is not so trivial to apply any AdaBoost approach to the recognition of a new vision problem. Pictures of the new object may not be readily available (such as those for faces). A positive training set numbering in the thousands is easily acquired with a few days spent on the internet hunting for faces. It took roughly a month to collect the data set required for the training and testing of the detection of the Honda Accord [24]. Even if a training set of considerable size could be assembled, how long would it take to train? Certainly, it would take in the order of months. It is therefore not possible to easily adapt Viola and Jones' standard framework to any vision problem. This is the driving force behind the large quantity of research that is being done in this field. Many authors still try to build upon the AdaBoost framework developed by Viola and Jones, which only credits this work further. The ideal object detection system in CV would be the one that can easily adapt to finding different objects in different settings while being autonomous from human input. Such a system is yet to be developed.

It is easy to see that there is room for improvement in the detection procedures seen here. The answer does not lie in arbitrarily increasing the number of training examples and WCs. The approach of increasing the number of training examples is brute force, and is costly when it comes to training time. Increasing the number of WCs would result in slower testing times. We propose to do further research in designing a cascaded classifier that will still work with a limited number of training examples, but can detect a wide range of objects. This new cascaded training procedure must also work in very limited time; in the order of hours, not days or months as proposed by predecessors.

The design of fuzzy WCs and the corresponding fuzzy training procedure may be worth further investigation. We have perhaps only seen applications that were solvable efficiently with standard binary WCs. There are perhaps some more difficult problems, with finer boundaries between positive and negative examples, where fuzzy WCs would produce better results. Since the change that is involved is quite small, affecting only a few lines of code, it is worth trying this method in future object detection cases.

All of the systems that were discussed here were mainly custom made to suit the purpose of detecting one object (or one class of objects). Research should be driven to find a flexible solution with a universal set of features that is capable of solving many detection problems quickly and efficiently.

An interesting open problem is to also investigate constructive learning of good features for object detection. This is different from applying an automatic feature triviality test on existing large set of features, proposed in the works by Stojmenovic [24,25]. The problem is to design a machine that will have the ability to build new features that will have good performance on a new object detection task. This appears to be an interesting ultimate challenge for the machine learning community.

REFERENCES

1. Abramson Y, Freund Y. Active learning for visual object recognition. Forthcoming.
2. Burghardt T, Calic J. Analysing animal behaviour in wildlife videos using face detection and tracking, IEE Proc Vision, Image Signal Proces. Special issue on the Integration of Knowledge, Semantics and Digital Media Technology; March 2005.
3. Bartlett MS, Littlewort G, Fasel I, Movellan JR. Real-time face detection and expression recognition: development and application to human-computer interaction. In: CVPR Workshop on Computer Vision and Pattern Recognition for Human–Computer Interaction, IEEE CVPR; Madison, Wi; 2003 June 17.
4. Barreto J, Menezes P, Dias J. Human–robot interaction based on Haar-like features and eigenfaces. In: Proceedings of the New Orleans: International Conference on Robotics and Automation; 2004. 1888–1893.
5. Burghardt T, Thomas B, Barham P, Calic J. Automated visual recognition of individual african penguins. In: Proceedings of the Fifth International Penguin Conference; Ushuaia, Tierra del Fuego, Argentina; September 2004.
6. Cristinacce D, Cootes T. Facial feature detection using AdaBoost with shape constraints. In: Proceedings of 14th BMVA British Machine Vision Conference; Volume 1, Norwich, UK; September, 2003. p. 231–240,
7. Efford N. Digital Image Processing: A Practical Introduction Using Java. Addison Wesley; 2000.
8. Freund Y, Schapire RE. A decision-theoretic generalization on on-line learning and an application to boosting. In: Proceedings of the 2nd European Conference on Computational Learning Theory (Eurocolt95); Barcelona, Spain;1995 p. 23–37; J Comput Syst Sci 1997;55(1):119–139.
9. Freund Y, Schapire RE. A short introduction to boosting. J J Soc Artif Intell 1999;14(5): 771–780.

10. Fröba B, Stecher S, Küblbeck C. Boosting a Haar-like feature set for face verification. Lecture Notes in Computer Science; 2003. 617–624.
11. Howe NR. A closer look at boosted image retrieval. In: Proceedings of the International Conference on Image and Video Retrieval; July 2003. p 61–70.
12. Jones M, Viola P. Fast multi-view face detection. Mitsubishi Electric Research Laboratories, TR2003-96 July 2003, http://www.merl.com; shown as demo at IEEE Conference on Computer Vision and Pattern Recognition (CVPR); June 2003.
13. Kolsch M, Turk M. Robust hand detection. In: Proceedings of the IEEE Interanational Conference on Automatic Face and Gesture Recognition; May 2004. p. 614–619.
14. Lienhart R, Maydt J. An extended set of haar-like features for rapid object detection. In: Proceedings of the IEEE Interanational Conference Image Processing. Volume 1, 2002. p 900–903.
15. Le DD, Satoh S. Feature selection by AdaBoost for SVM-based face detection. Information Technology Letters, The Third Forum on Information Technology (FIT2004); 2004.
16. Le D, Satoh S. Fusion of local and global features for efficient object detection. IS & T/SPIE Symposium on Electronic Imaging; 2005.
17. Levi K, Weiss Y. Learning object detection from a small number of examples: the importance of good features. In Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR). Volume 2, 2004. p 53–60.
18. Li X, Wang L, Sung E. Improving AdaBoost for classification on small training sample sets with active learning. In: Proceedings of the Sixth Asian Conference on Computer Vision (ACCV)., Korea, 2004.
19. Luo H, Yen J, Tretter D. An efficient automatic redeye detection and correction algorithm. In: Proceedings of the 17th IEEE International Conference on Pattern Recognition, (ICPR'04); Volume 2, 2004 Aug 23–26 Cambridge UK: p 883–886.
20. McCane B, Novins K. On training cascade face detectors. Image and Vision Computing. Palmerston North, New Zealand, 2003. p 239–244.
21. Silapachote P, Karuppiah DR, Hanson AR. Feature selection using AdaBoost for face expression recognition. In: Proceedings of the 4th IASTED International Conference on Visualization, Imaging, and Image Processing, VIIP 2004; Marbella, Spain, September 2004. p 452–273.
22. Sung K, Poggio T. Example based learning for view-based human face detection. IEEE Trans Pattern Anal Mach Intell 1998; 20:39–51.
23. Schapire R, Singer Y. Improved boosting algorithms using confidence-rated predictions. Mach Learn 1999. 37(3):297–336.
24. Stojmenovic M. Real time machine learning based car detection in images with fast training. Mach Vis Appl, 2006;17(3):163–172.
25. Stojmenovic M. Real time object detection in images based on an AdaBoost machine learning approach and a small training set. Master thesis, Carleton University; June 2005.
26. Treptow A, Masselli A, Zell A. Real-time object tracking for soccer-robots without color information. In: Proceedings of the European Conference on Mobile Robotics ECMR, 2003.
27. Viola P, Jones M. Robust real-time face detection. Int J Comput Vis 2004. 57(2):137–154.
28. Viola P, Jones M. Fast and robust classification using asymmetric AdaBoost. Neural Inform Processing Syst 2002;14.

29. Viola P, Jones M, Snow D. Detecting pedestrians using patterns of motion and appearance. In: Proc Ninth IEEE Int Conf Comput Vision ICCV 2003; 2:734–741.
30. Wu J, Regh J, Mullin M. Learning a rare event detection cascade by direct feature selection. In: Proceedings of the Advances in Neural Information Processing Systems 16 (NIPS*2003), MIT Press, 2004.
31. Zhang D, Li S, Gatica-Perez D. Real-time face detection using boosting learning in hierarchical feature spaces. In: Proceedings of the International Conference on Pattern Recognition (ICPR); Cambridge, Aug. 2004. p 411–414.
32. Li SZ, Zhang Z. FloatBoost learning and statistical face detection. IEEE Trans Pattern Anal Machine Intell Sept. 2004. 26(9):1112–1123.

[**Q1**]:- There are only 32 ref. in the reference list, but ref. 34 has been cited here, Please check.