Real time object detection in images based on an AdaBoost machine learning approach and a small training set

by

Miloš Stojmenović

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

June 2005

Your file Votre référence
ISBN: 0-494-10122-9
Our file Notre référence
ISBN: 0-494-10122-9

Abstract

Learning machine approaches for real time object detection are frequently used in literature. One of the most successful applications was introduced by Viola [39], who used AdaBoost for face detection in images. Our interest is to build fast and reliable object recognizers in images based on small training sets. This is important in cases where the training set needs to be built manually, as in the case that we study, the recognition of a type of car, specifically, the Honda Accord 2004 from rear views. We describe a novel variant of the AdaBoost learning algorithm, which builds a strong classifier by incremental addition of weak classifiers (WCs) that minimize the combined error of the already selected WCs. We describe a set of appropriate feature types for the considered recognition problem, including a redness measure and dominant edge orientations. We proposed to pre-eliminate features which do not satisfy a cumulative error threshold. Compared to existing literature, we achieve the design of a real time object detection machine with the least number of examples, the least number of weak classifiers, and with competitive detection and false positive rates.

Acknowledgements

I would like to thank Franz Oppacher, Jean-Pierre Corriveau, Gerhard Roth, Prosenjit Bose, Stan Matwin, Doron Nussbaum, Amiya Nayak, Mario Marchand, OGS, the School of Computer Science, and my family and friends for having contributed, in one way or another, to the completion of this thesis.

Table of Contents

List of Tables

List of figures

Guide to Notation

Here are some notations that are frequently used in this thesis, in no particular order.

f = feature,

x = tested image,

$h(x, f, p, \theta)$ = weak classifier,

s = the sign (+ or -) in a weak classifier,

α = weight of a weak classifier,

q = the total number of images (training examples), $q=p+n$,

p = the total numbers of positive images,

n = the total numbers of negative images,

T = number of weak classifiers in a strong classifier,

N = number of (strong) classifiers in a cascade,

F = the number of features.

Chapter 1 - Overview

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 in CV can be divided into 3 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 of interest to us 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 in cities 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.

1.2 Problem Statement

The goal of this thesis is to analyze the capability of current machine learning techniques of solving other similar image retrieval problems. The 'capability' includes real time performance, a high detection rate, low false positive rate, and learning with a small training set. We are particularly interested in cases where the training set is not easily available, and most of it needs to be manually created.

We will apply machine learning to the detection of rears of cars in images. Specifically, the system should be able to recognize cars of a certain type such as a Honda Accord, 2004. While Honda's 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. The system will work by directly searching for an occurrence of the positive in the image, while treating all sub windows 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 Honda Accord 2004 will be 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 dataset was collected by taking pictures of Hondas (about 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 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 which is outside the machine learning framework.

In addition to precision of detection, the second major main goal is *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 finds all the heads. The definition of 'real time' depends on the application, but generally speaking we would like to receive 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, our goal is to also design the object detection system based on small number of training examples. We envision applications in cases where training examples are not easily available. For instances, in the case that we studied, we had to take photos of back views of a few hundreds of Honda Accords and other cars 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 weak classifiers, and otherwise analyze the process by observing the behaviour of the generated classifiers.

1.3 Motivation

Our main motivation was to learn how to detect objects of certain types in images in real time (in terms of the speed of the detection program, speed of the training program, and programming efforts to write that program), with the help of machine learning methods. The selection of the rear views of the Honda Accord 2004 for the recognition was motivated by possible applications of this type of problem. The success of this system may be used as a basis for a larger system of traffic analysis and control. Finding cars in images can facilitate the process of finding license plates and

subsequently compare real time information from images to stored information in a database. License plate numbers can be read and looked up in a database so that the make and model of a car can be verified against its appearance in the image for instance (this is not part of our problem statement). It becomes possible to identify stolen cars, pinpoint the location of cars in real time, and automatically send fines to drivers breaking traffic laws and a myriad of other applications.

1.4 Existing Solutions

We will first comment on the car detection problem, which is not the main purpose of our study. However, we would like to show that there are no alternate solutions for the stated case, with the desirable properties. We reviewed solutions to more general problems than the one actually solved in this thesis.

Existing vehicle detection systems, such as those that try to drive cars automatically along a highway do not actually detect cars on the road. They simply assume that anything that is moving on the highway is a vehicle. In scientific literature, some car recognition solutions also exist that are based on shape detectors [35], [36]. An existing shape matching based approach [36] is reported to have a 60%-85% detection rate, which is below our stated goals. The approach based on nearest neighbour matching [29] is too sensitive to viewpoint change, while the approach based on PCA (Principal Component Analysis) [29] is not a real time system.

The most popular example of object detection is the detection of faces. The fundamental application that gave credibility to AdaBoost was Viola's real time face finding system [39]. AdaBoost is the concrete machine learning method that was used by Viola to implement the system. 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's face finding system has been modified in literature in a number of articles. The modifications include inclusion of new features. Of particular interest to us were features based on gradient histograms, and those based on the color of certain parts of an image. The AdaBoost machine itself was modified in literature in several ways. We have considered all modifications proposed in literature, and adopted ideas that were considered helpful for achieving our goals.

We stress again that most of the successful applications of AdaBoost used a large training set. In Viola's original face detector [39], 10,000 images were used in the training set. The smallest known training database for face detection was by Levi and Weiss [21]. They started to achieve detection rates in the 90% category when the number of positive examples reached 250. The number of negative examples was not specified at this level, but the authors say that they randomly downloaded 10,000 images from the Internet containing over 100,000,000 sub windows. They only moderately increased their detection rates as the size of the positive set grew drastically.

1.5 Limitations and generalizations

We will apply machine learning methods 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. A classic application has become the detection of upright forward facing heads as proposed by Viola [39]. Cars are typically found in the same orientation with respect to the road. They can be photographed from various angles (front, side ...) but they are rarely found up-side-down. The situation we are 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. It is also the target of choice of police traffic monitoring equipment, since their cameras are usually positioned to film the license plates for the purposes of vehicle recognition. Therefore, hardware is already in place for various software applications in vehicle detection.

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 -15^0, yaw -30^0 to 30^0, and roll -15^0 to 15^0. This means that pictures of Hondas taken from angles that are off by the stated amounts are still detected by the program. Figure 1.1 clearly shows the deviations allowed. Yaw, pitch and roll is common jargon in aviation describing the three degrees of freedom the pilot has to manoeuvre an aircraft.

Figure 1.1 – Pitch, Roll and Yaw restrictions, respectively for detecting the Honda accord

The face detection scheme introduced by Viola was far more delicate than what we achieved. In [39], it was claimed that pitch and roll can vary $\pm 15^0$, and yaw can vary up to $\pm 45^0$ but experimentation with the implementation of [39] did not support these claims. All three degrees of freedom mentioned above were more restrictive in face detection than in our detector.

Machine learning concepts in the computer vision 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. This is one of the reasons that faces are a popular application.

1.6 Contributions

This thesis does not merely replace the training sets of faces and non-faces with training sets of Honda's and other cars' rear views in Viola's face detector program. We analyzed every step in the process, and made a number of changes to achieve our stated goals. In summary, this thesis describes how machine learning helps in object detection. Our real time system based on machine learning achieves very good accuracy. A static feature selection approach can also be applied, but it is likely to have suboptimal performance. Our expected best feature set for detecting Hondas was far different than the one chosen by our version of AdaBoost. It is difficult for a human to browse through hundreds of images and be able to visually select the best features for classification. Human perception, however, was important in selecting the types of features, and 'feeding' AdaBoost with the proper choices. Types of features to be used depend on the type of object to recognize. We ended up with a completely disjoint set of feature types compared to the set used by Viola for face detection. However, within each type, e.g. redness in certain regions, it is up to AdaBoost to decide what exactly the best weak classifier is (that is, which rectangle to be evaluated for redness), and what is the next

best weak classifier whose addition gives the greatest contribution. This task would be even more difficult and time consuming for a human to perform.

The contributions of this thesis can be summarized as follows:

1. We described a novel variant of AdaBoost based learning algorithm, which builds a strong classifier by incremental addition of Weak Classifiers (WCs) that minimize the combined error of already selected WCs. Each WC is trained only once, and examples do not change their weights. While all the individual components of this approach exist in literature, it was not yet used as a combined whole algorithm the way we propose here.

2. We built a fast and reliable object recognizer based on small training set, consisting of 155 positive and 760 negative images. It detects back views of Honda Accords with a 98.7% detection rate and 0.4% false positive rate on the training set, and with 89.1% detection rate and a 1.48×10^{-6} false positive rate on a test set of 106 images containing roughly 17.5 million tested sub windows.

3. We described a set of appropriate feature types for this problem, including a redness measure and dominant edge orientations. Our experiments indicated that the set of features used by Viola was inefficient for our problem; therefore, each problem requires its own custom-made set of features. The existing edge orientation bin division was improved by shifting so that all horizontal (vertical, respectively) edges belong to the same bin.

4. We proposed to pre-eliminate features whose best threshold value is near the trivial position at the maximum or minimum of feature values. This is a novel method that has reduced the set of available weak classifiers to less than one tenth of its original size (for the case we studied), greatly speeding up training time, and showing no negative impact on the quality of the final classifier.

Compared to existing literature, we have achieved the overall design of a real time object detection machine with the least number of examples, the least number of weak classifiers (30), and with competitive detection and false positive rates. We demonstrated

the significant impact of negative examples on the training process. We employ a semi-testing set of examples. After providing some initial negative examples, false positives from the semi-testing set are added to the negative example pool. This method is known as bootstrapping in some papers. It was introduced in [33]. We have shown that this method has its limits, since the continued application of it (overfitting) starts to 'attack' the best weak classifiers and consequently starts to reduce the accuracy of the classifier.

Chapter 2 – Literature Review

2.1. 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.

Osuna [27] reported a real time face detector using SVMs. Osuna [27] did not use a clean SVM approach. He used a two step SVM detector where the 1st step uses a quick eliminator of candidate windows (which is not explained in his article) before the full SVM is unleashed on the difficult examples. This confirms that SVMs cannot be exclusively used in their natural form for real time tasks. Those SVMs that may contain faces are then processed by a complete SVM classifier. No details are given for the first phase, apparently very important for real time performance. Osuna's article [27] was published in 1998, which is before Viola's AdaBoost was widely adopted (mainly due to the integral image 'trick'). That is, SVM's could have been fastest at that time. SVMs and AdaBoost apply quadratic and linear programming, respectively. Thus for a given fixed number of selected weak classifiers in the testing process, AdaBoost will be much faster. 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 [20] 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-5,924 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 hours of pure run time to test the 106 images. It would take roughly 2 minutes to process an image of size 320x240. This is hardly real time. Thus [20] claims that cascaded AdaBoost is 1000 times faster than SVMs. A regular AdaBoost with 30 features was presented in this thesis. A cascaded design cannot speed up described version by more than 30 times. Thus our program is faster than SVM by over 1000/30 > 30 times. This is just a lower bound, in practice it may be roughly 100 times faster.

Bartlett et al. [4] 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's approach [39]. 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% 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" [4]. Moreover, each of these features is evaluated in constant time, because of integral image pre-processing. 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 in [11]. The connection between AdaBoost and SVMs was discussed in [12]. 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 programming, i.e., 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* [12]. Quadratic programming is more computationally demanding than linear programming [12].

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 [12]. 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 [12] 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 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 which are too weak. Boosting seems to be especially susceptible to noise." [11].

Schapire and Singer [34] described several improvements to Freund and Schapire's original [11] 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 [11]. While essentially proposing a general fuzzy AdaBoost training and testing procedure, [11, 34] do not describe any specific variant, with concrete fuzzy classification decisions. We propose in this thesis a specific variant of fuzzy AdaBoost. Whereas [11] prescribes a specific choice of weights for each classifier, [34] leaves this choice unspecified, with various tunings. Extensions to multiclass classifications problems are also discussed.

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 our research comes from the face detector work done by Viola et al. [39]. All subsequent papers that use and improve upon AdaBoost are inspired by this same paper. The idea of face detection and the logic behind its implementation gave inspiration for a host of applications that make use of machine learning concepts. We try to use these machine learning concepts in our attempts to locate a car in an image.

2.2 Viola's Face detector

The face detector proposed by Viola [39] 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's contributions come in the training and assembly of the AdaBoost machine. He 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 [39]. Cascaded AdaBoost is a series of smaller AdaBoost machines that together provide the same function as one large AdaBoost machine, yet evaluate each sub window 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's face detector in detail.

2.2.1 Finding faces in different scales

In this section, we assume that we have a machine that takes in a square region of size equal to or greater than 24x24 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. We use such a machine to analyze the entire image, as illustrated in Figure 2.1. We pass every sub window of every scale through this machine to find all sub windows that contain faces. A sliding window technique is therefore used. The window is shifted 1 pixel after every analysis of a sub window. The sub window grows in size 10% every time all of the sub windows of the previous size were exhaustively searched. This means that the window size grows exponentially at a rate of $(1.1)^p$, where p is the number of scales. In this fashion, more than 90% of faces of all sizes can be found in each image.

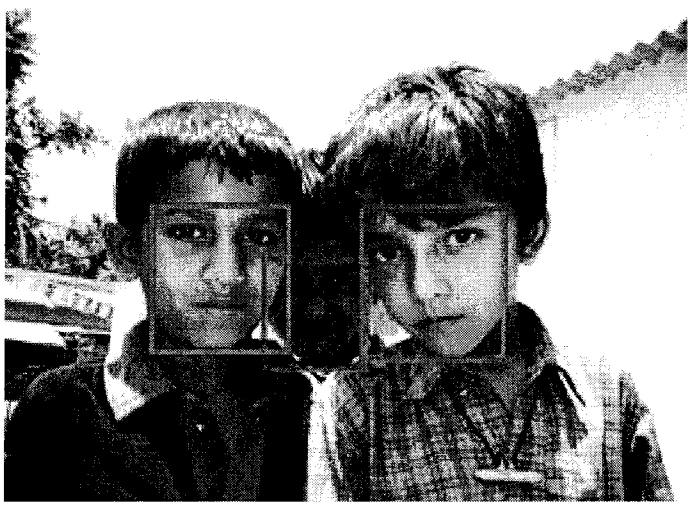

Figure 2.1 - Sub windows of an image

2.2.2 The training set

As with any other machine learning approach, the machine must be trained using positive and negative examples. Viola used 5,000 positive examples of randomly found upright, forward-facing faces and 10,000 negative examples of any other non-face objects as his 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 which are concrete examples. That is, faces are cropped from larger images, and positive examples are basically close up portraits only (see Figure 2.2, taken from [39]). 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, as seen in Figure 2.2. The method is not likely to work properly if the faces change orientation.

Figure 2.2 – Example portraits taken used by Viola [39]

2.2.3 Features

An image ***feature*** is a function that maps an image into a number or a vector (array). Viola [39] 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 were the actual greyscale 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 2c in Figure 2.3 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.

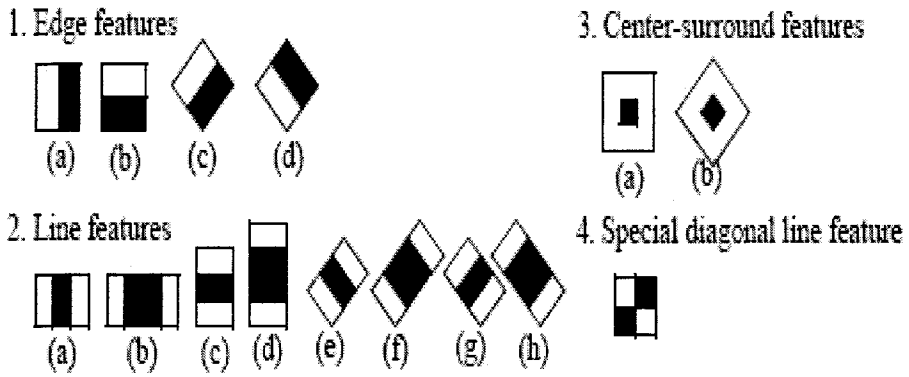

Figure 2.3 – Basic shapes that generate features by translation and scaling

Figure 2.3 shows the set of basic shapes used by Viola [39]. Actually, shapes 1a,b, 2a-d, 3a and 4 were used by Viola, and the other (rotated) shapes were introduced later by [18]. 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 2.3 is scaled and translated anywhere in the test images, consequently forming features. Therefore, each feature includes a basic shape (as seen in Figure 2.3), 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 2.4. Both of the features seen in Figure 2.4 (each defined by its position and scaling factors) are derived from the basic shapes in Figure 2.3.

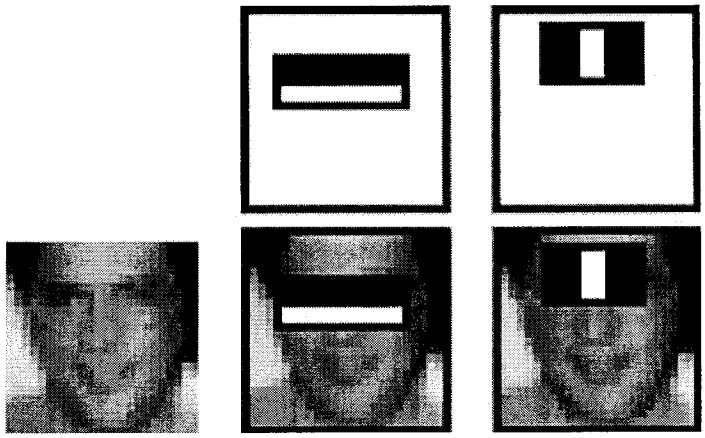

Figure 2.4 – Common features in face detection (taken from [39])

Figure 2.4 shows the first and second features selected by the program [39]. 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 squares containing eyes and the area between them. This feature corresponds to feature 2b in Figure 2.3 where the light and dark areas are inverted. This is not a separate feature; it was drawn this way in Figure 2.4 to better depict the relatively constant number obtained by this feature when it is evaluated in this region on the face. It is relevant to note that both the scale and the position of the features seen in Figure 2.4 best identify a face.

2.2.4 Weak classifiers

A weak classifier is a function of the form $h(x, f, s, \theta)$ where x is the tested sub image, f is the feature used, s is the sign (+ or -) and θ is the threshold. 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 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 $3/17 \approx 0.17$.

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 weak classifier needs only to produce results that are slightly better than chance to be useful. A combination of weak classifiers is assembled to produce a strong classifier as seen in the following text.

2.2.5 Strong Classifiers

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

classifier, labelled $h_1, h_2, \ldots, h_T$, and each of these comes with its own weight labelled $\alpha_1, \alpha_2, \ldots, \alpha_T$. Tested image x is passed through the succession of weak classifiers $h_1(x), h_2(x), \ldots, h_T(x)$, and each weak classifier 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:

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

From this equation, we see that images that pass a weighted average of half of the weak classifier tests are catalogued as positive. It is therefore a weighted voting of selected weak classifiers.

Figure 2.5 shows a 2D ordering of positive and negative examples of faces according to two features and their thresholds which together form weak classifiers. In this 2D space, image x is placed at location $(f_1(x), f_2(x))$. The threshold of weak classifier 1 is the vertical red line and the threshold of weak classifier 2 is the horizontal red line. The dotted line signifies a rough boundary between positive and negative sets. When only two features are used, the classifier in fact returns the result of the one which has a larger weight. If we assume that the weight of classifier 1 is α_1 and the weight of classifier 2 is α_2, and $\alpha_1 > \alpha_2$, then final result is the same as that of weak classifier 1. This illustration can easily be generalized to higher dimensions where voting becomes more democratic.

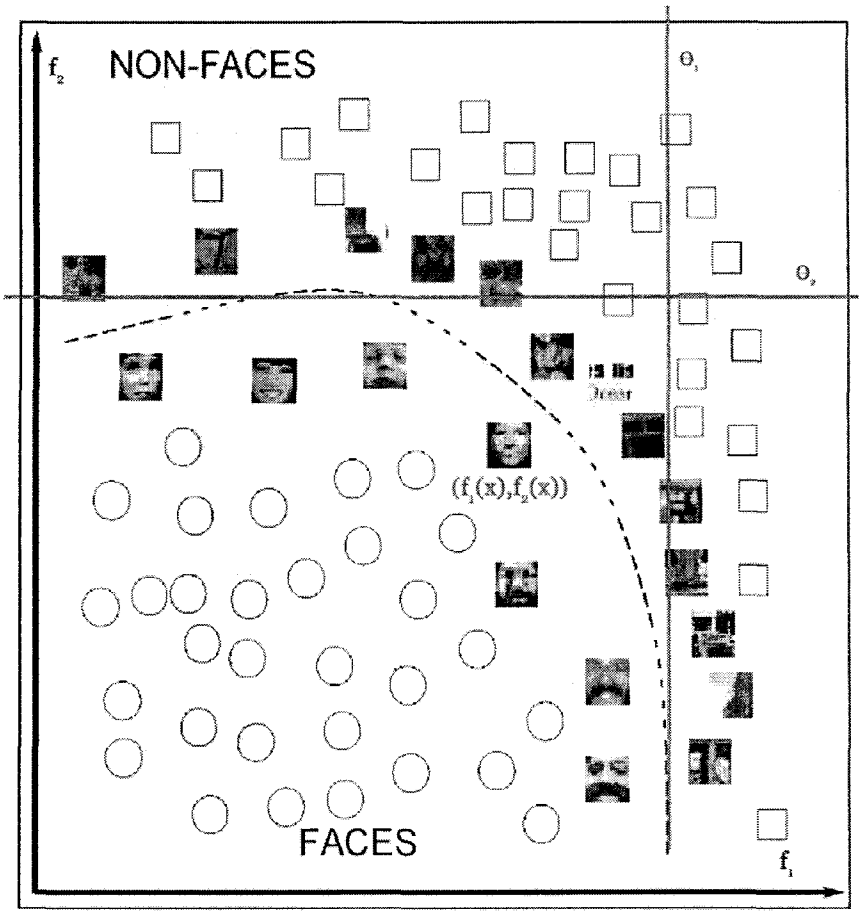

Figure 2.5 – 2D space of faces

2.2.6 AdaBoost: Meta Algorithm

AdaBoost was introduced by Freund and Schapire [11]. In this section we explain the general principles of the AdaBoost (an abbreviation of Adaptive Boosting) learning strategy. First, a huge (possibly hundreds of thousands) 'panel' of experts is identified. Each expert, or weak classifier (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 which 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 which 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 which 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 weak classifiers 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 [39], will be given.

2.2.7 AdaBoost Training Algorithm

We now show how to create a classifier with the AdaBoost machine. It follows the algorithm given in [39]. The machine is given images (x_1,y_1), ..., (x_q, y_q) as input, where $y_i=1$ or 0 for positive and negative examples, respectively. In iteration t, the i^{th} image is assigned the weight $w(t,i)$, that 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, totalling ½, and similarly for all negative images.

The algorithm will select, in step t, the t^{th} 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 $\sum_{i=1}^{N} w(t,i) * |h(x_i, f, s, \theta)- y_i|$ 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 weak classifier.

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 weak classifier 1. We notice the *'1'*, *'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 2 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 weak classifier 2. The actual training algorithm will be described in pseudo code 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 ← sum of weights of all positive images

swn ← sum of weights of all negative images

(* note that *swp* + *swn* =1 *)

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

- 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: $g_1, g_2, ..., g_q$. 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 j-th element g_j is equal to i-th 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*)

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

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

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

$w(t+1, i) = w(t,i)\, \beta_t^{1-e}$ where $e = \begin{cases} 0 \text{ if } x_i \text{ is correctly classified by current } h_t \\ 1 \text{ otherwise} \end{cases}$

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.

2.2.8 Cascaded AdaBoost

Viola [39] also described the option of designing a cascaded AdaBoost. For example, instead of one AdaBoost machine with 100 classifiers, one could design ten 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 [39]. One particular image is first tested on the first classifier. If declared as non-similar, 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 3 illustrates this process.

Each classifier seen in Figure 2.6 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 passed through this classifier, while at the same time accepting nearly 100% of the positives. It is desirable that each classifier eliminates at least 50% 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's face finder cascade, the first classifiers had 2, 10, 25, 25, and 50 features respectively. The number of features grew very rapidly afterwards. Typical numbers of features per classifier ranged in the hundreds. The total number of features used was roughly 6000 in Viola's application.

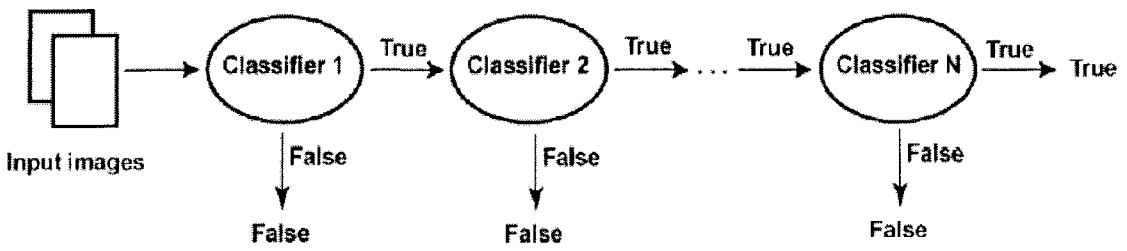

Figure 2.6 – Cascaded decision process

Figure 2.7 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 2.7, we assume to have 2 weak classifiers with weights α_1 and α_2. Together these two weak classifiers make a conceptual hyper plane depicted by the solid dark blue line. In actuality, this line is not a hyper plane, (in this case a line in 2-D 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 weak classifiers form hyperplanes.

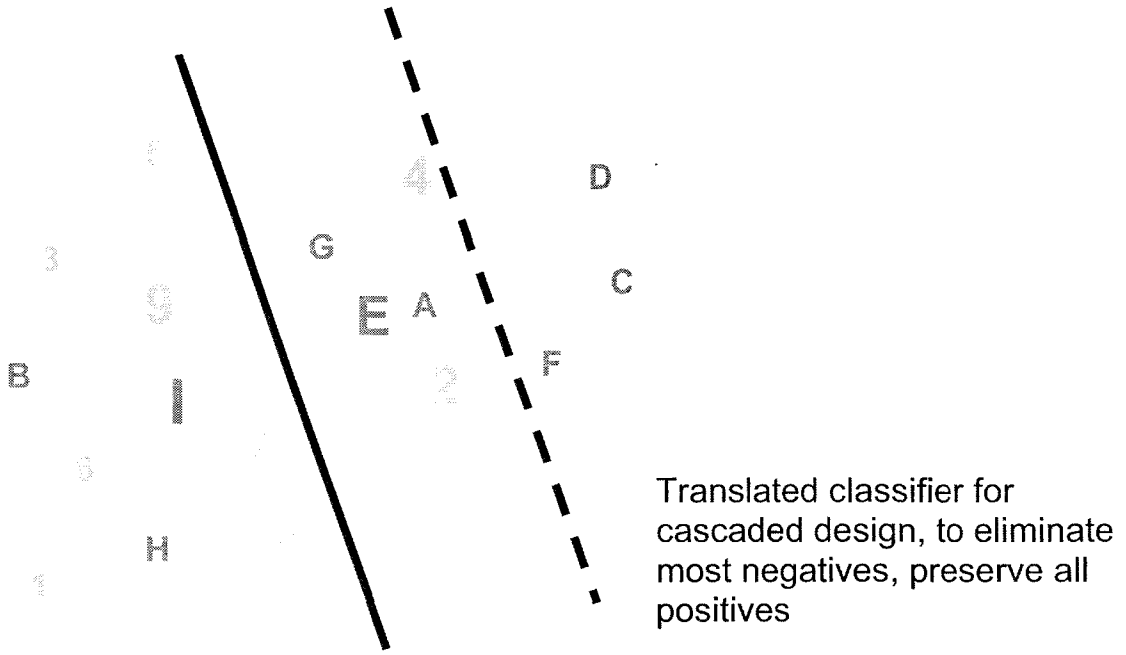

Figure 2.7 – Concept of a classifier

So far in figure 7, we have 2 weak classifiers 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 weak classifiers would be checked against the training set to see if they have a 99% detection rate (this 99% 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 2.7 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% (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 successfully eliminated by a previous classifier. In Figure 2.7, *'D'*, *'C'* and *'F'* would be eliminated from future consideration if the classifier construction were completed at this point.

2.2.9 Integral Images

One of the key contributions in [39] (which is used and/or modified by [21], [23], 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 pre-processing step, Viola [39] finds 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:

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

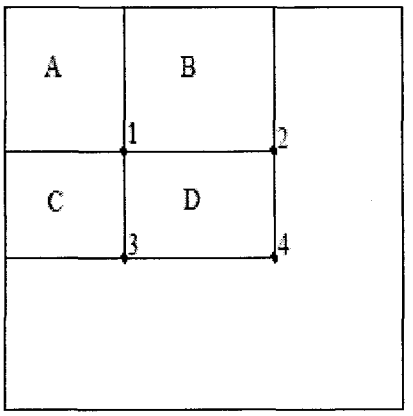

Figure 2.8 – Integral image

Figure 2.8 shows an example of how the 'area' for rectangle 'D' can be calculated using only 4 operations. Let the area mean the sum of pixel intensities of a rectangular region. The pre-processing 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) + (A) – (A+B) – (A+C) = '4' + '1' – '2' – '3'.*

2.2.10 Modifications by Viola to face detection

Tieu and Viola [38] discussed generating a large number of features, used in the AdaBoost learning algorithm. Viola and Jones [40] proposed some modifications to their original machine, to improve its speed. In the first modification they change the weight for classifiers, but report that gains obtained at the beginning were compensated for later on when more classifiers were added. They describe one more improvement for which they claim real benefits; however, the description of that approach is left vague. Note that the journal version of [39] is published three years later, without referring to this improvement made in the same year as the conference version of [39].

Jones and Viola [15] 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. This contrasts their earlier claim in [39] that the detector they built has the capability of recognizing faces with a $\pm 45^0$ freedom in yaw. It is not clear why they needed a large number of classifiers for each view of the face when they claim that their system is so flexible.

2.3 Review of additional references

2.3.1 Rotated features and post optimization

Lienhart and Maydt [18] add a set of classifiers (Haar wavelets) to those already proposed by Viola. Their new classifiers are the same as those proposed by Viola, but they are all rotated 45 degrees. They claim to gain a 10% improvement in the false detection rate at any given hit rate when detecting faces. Figure 2.3 shows the complete set of features tested by Lienhart. *3a* and *3b* are completely new (not used by [39]) and feature 4 is used by [39], but not by Lienhart since it is well approximated by *2g*.

However, there is a post optimization stage involved with the training process. This post optimization stage is credited with over 90% 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 degree rotated images since Lienhart was (and still is) on the development team for OpenCV. OpenCV version 4 beta has been released in September 2004.

2.3.2 Detecting pedestrians

Viola, Jones and Snow [41] 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.

2.3.3 Detecting Penguins

Burghardt et al. [6] 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 [18] 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.

2.3.4 Redeye detection: Colour based feature calculation

Luo, Yen and Tretter [23] introduce an automatic redeye detection and correction algorithm which 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, colour information along with aspect ratios (width to height) of regions of interest as trainable features in their AdaBoost implementation.

Viola [39] only used greyscale intensities, although his solution to face detection could have used colour information. Finding red eyes in photos means literally finding red oval regions which absolutely require the recognition of colour. Another unique addition in their work is a set of new features similar to those proposed by Viola [39], yet designed specifically to easily recognize circular areas. We see these feature templates in Figure 2.9. It is noticeable that the feature templates presented in this figure have 3 distinct colours: white, black and grey. The grey and black regions are taken into consideration when feature values are calculated. Each of the shapes seen in Figure 2.9 is rotated around itself or reflected creating 8 different positions. The feature value of each of the 8 positions is calculated, and the minimum and maximum of these results are taken as output from the feature calculation. To illustrate this procedure, we take the shape in the top right hand corner of Figure 2.9 and perform the 8 rotations and reflections on it. We see the results in Figure 2.10.

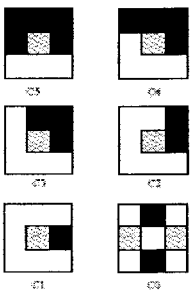

Figure 2.9 – Features for redeye detection

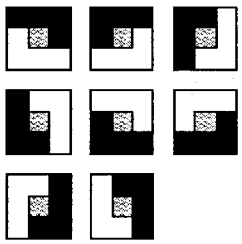

Figure 2.10 – Rotated features

The actual calculations are performed based on the RGB colour space. The pixel values are transformed into a one dimensional space before the feature values are calculated in the following way: $f_r(I(R,G,B)) = w(4R-3G+B)$, where w is a weighting factor. This colour space is biased toward the red spectrum (which is where red eyes occur).

2.3.5 Edge orientation histogram based features

Levi and Weiss [21] add a new perspective on the training features proposed by Viola [39]. They also detect upright, forward facing faces. Among other contributions in [21], 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 [39], which considered sums of pixel intensities, [21] create features based on the most prevalent orientation of edges in rectangular areas. In Figure 2.11, edge orientation histograms are considered for the area around the eyes which has mainly horizontal edges. There are obviously many orientations available for each pixel but they are reduced to 8 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 8 edge orientation histograms (EOH) [21]. 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. [21] claim that

using edge orientation histograms, they are able to achieve higher detection rates at all training database sizes.

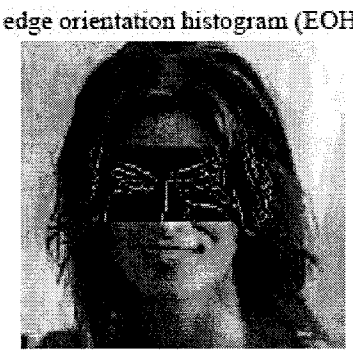

Figure 2.11 – Edge orientation based features

Their goal was to achieve similar or better performance of the system to Viola's 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. Examples of symmetry features are seen in Figure 2.12. [21] exploited the fact that faces have the quality of being vertically symmetric. Every time a weak classifier 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% when both machines were run simultaneously on the test data.

Figure 2.12 – Symmetry features

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.

2.3.6 Fast AdaBoost

Wu, Rehg and Mullin [42] propose a training time performance increase over Viola's 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 i^{th} classifier $(1 \le i \le N)$ is given as input the desired minimum detection rate d_i and the maximum false positive rate 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 fp_i. The obtained weak classifiers 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 weak classifiers (that is, each WC has equal weight in [42]). The illustration of this procedure, in comparison with that of Viola, is seen in figure 2.13.

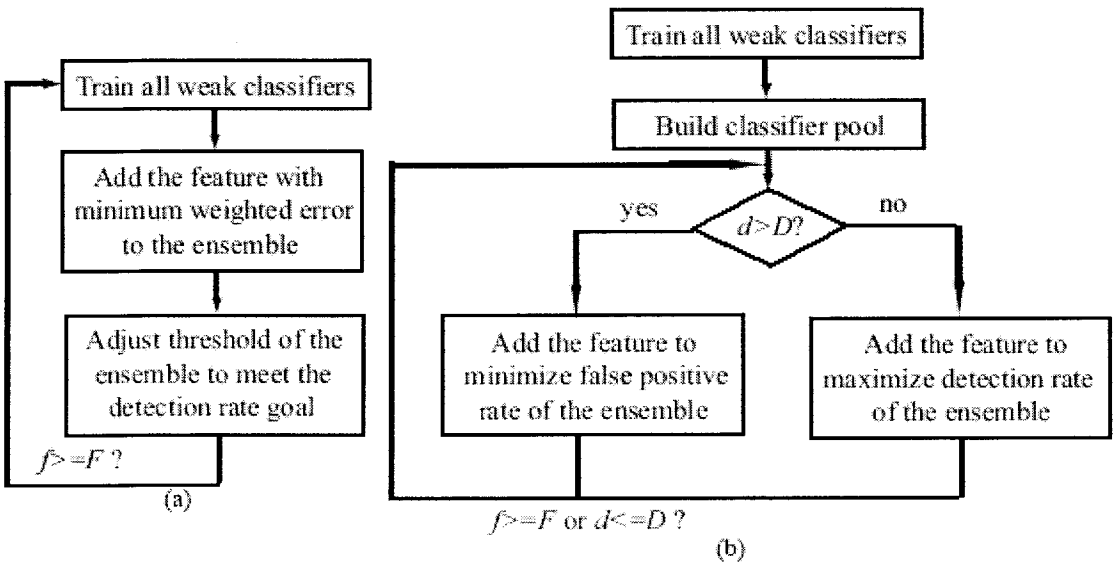

Figure 2.13 – Part a) shows Viola's method, part b) shows the [42] method

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.

The authors [42] state that their method works almost as well as Viola's when applied to the detection of upright, forward facing faces. They however achieve a training time which is two orders of magnitude faster than that of Viola. This is achieved in part by using a training set that was much smaller than Viola's [39], yet generated similar results.

We will now explain the time complexity of both Viola's [39] and Wu's [42] training method. There are three factors to consider when finding the time complexity of each training procedure: the number of features F, the number of weak classifiers 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 weak classifier. Finding the best feature takes $O(Fq \log(q))$ time. Therefore, the construction of the classifier takes $O(TFq \log q)$.

The [42] method takes $O(Fq \log q)$ time to train all of the classifiers in the initial stage. Testing each new weak classifier 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 weak classifier. Therefore it takes $O(TqF)$ time to chose T weak classifiers. We deduce that it would take $O(Fq \log q + TqF)$ time to complete the training using the methods described by [42]. The dominant term in the time complexity of the [42] algorithm is $O(TqF)$. This is order $O(\log q)$ times faster than the training time for Viola's method [39]. For a training set of size q=10000, $log_2 q \approx 13$. For the same size training sets, The [42] 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's method [39].

Froba, Stecher, and Kublbeck [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. Figure 2.14 shows example input for the system.

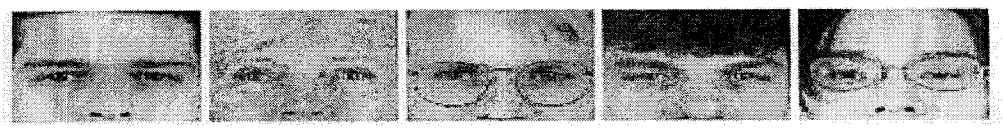

Figure 2.14 – Examples used in face verification [13]

The authors use the same training algorithm for face detection as Viola [39], 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's [39]. The authors mention that a face is detectable and verifiable with roughly 200 features which 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 are found and compared against a threshold as the verification step. This is a sort of nearest neighbour comparison which is used in many other applications.

2.3.7 Downhill feature search

McCane and Novins [26] described two improvements over the Viola [39] 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), [26] propose an optimization search, by applying a 'downhill search' approach. Starting from a feature, a certain numbers of neighbouring 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 neighbours. They observe that [39] applies AdaBoost in each stage to optimize the overall error rate, and then, in a post-processing 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 sub-squares in a feature to be translated. The second improvement in [26] 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 [39] could be performed at earlier stages in the cascade, and replaced by optimized search [26] in later stages.

2.3.8 Bootstrapping

Sung and Poggio [33] applied the following 'bootstrap' strategy to constrain the number of non-face examples in their face detection system. They incrementally select only those non-face 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 'semi-testing' set). All non-face 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 [4], a similar bootstrapping technique was applied. False alarms are collected and used as non-faces for training the subsequent strong classifier in the sequence, when building a cascade of classifiers.

Li, Wang and Sung [22] 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 labelling them is costly and time consuming. They propose an active learning approach, to select the next unlabelled sample which is at the minimum distance from the optimal AdaBoost hyperplane derived from the current set of labelled samples. The sample is then labelled 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 labour required for labelling 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 sub windows with close classification scores, which are then labelled and entered into positive and negative examples. In addition to features from [39], authors also use features with 'control points' from [2].

Authors [43] empirically observe that in the later stages of the boosting process, the non-face examples collected by bootstrapping become very similar to the face examples, and the classification error of Haar-like feature based weak classifiers is thus very close to 50%. As a result, the performance of a face detection method cannot be further improved. Authors [43] propose to use global features, derived from PCA (Principal Component Analysis), in later stages of boosting, when local features do not provide any further benefit. They show that weak classifiers 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 non-faces, not based on the size of coefficient.

2.3.9 Other AdaBoost based object detection systems

Treptow et all [37] described a real-time soccer ball tracking system, using the described AdaBoost based algorithm [39]. The same features were used as in [39]. They add a procedure for predicting ball movement.

Schneiderman and Kanade [31] described how to detect faces and cars using AdaBoost and localized parts. They deviate from the original idea by Viola, and have a very difficult writing style. I was not able to derive any useful ideas from this article.

Claus and Fitzgibbon [9] described a system that detects four dots in an image, located at corners of a square originally (within the image). They use a learning machine that recognizes dots, by giving examples of real dots from images as positive examples. A fast classifier is applied first, which compares intensities in the center and at the border of sub images. If the test passes, then the new sample is assigned to the class of the nearest training example. This is a very simplified learning machine compared to AdaBoost.

Abramson and Steux [2] propose a different type of feature for use in AdaBoost based pedestrian detection. Their features consist of two sets, X and Y, of control points. The feature answers 'yes' if and only if the pixel values in any of point from X is larger than pixel value in any point from Y. Weak classifiers defined in this way have no thresholds and are resistant to changes in scene illumination, and do not require prior normalization of all image sub-windows. The selection of features is done by a genetic algorithm whose details are not given. The features are sensitive to noise, and the authors do not explain the features selected in the training process [2].

Cristinacce and Cootes [8] 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 [44, 24] differs from AdaBoost in a step where the removal of previously selected weak classifiers is possible. After a new weak classifier 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 weak classifiers, the authors [44, 24] 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.

In [28], the authors propose a method for accurate location of vessel borders based on boosting classifiers and feature selection. It deals with medical imaging. Co-occurrence matrices and cumulative moments are used as the feature set, and the search is

performed over the obtained coefficients. The article is difficult to read, and contains a lot of formulas without much insight into what is being done.

Howe [14] looks at boosting for image retrieval and classification, with comparative evaluation of several algorithms. Boosting is shown to perform significantly better than the nearest neighbour approach. Two boosting techniques that are compared are based on feature and vector based boosting. Feature based boosting is the one used in [39]. 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 which 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's 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 [19] 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 sub windows of size 36x36, the second stage is a cascaded classifier with sub windows of size 24x24. The third stage is a SVM classifier for greater precision.

The authors [32] 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.

The authors [5] 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's features [39] improved by Lienhart and Maydt [18]. 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 in [16] which 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.

2.4 Recognizing cars and other objects

Bredeche and Zucker [7] described a different approach to object recognition. Roughly, each image is labeled whether or not it contains a few related objects, such as a human, door, or fire extinguisher. A new image is then classified based on a specific invariant property that can be found in at least one part of the image. The authors refer to additional articles for sophisticated details. This approach is an example of a 'whole picture' approach, where every pixel in the image impacts the feature value, although the searched object is only a part of that image. We are not aware of any widely accepted system for object detection based on such an approach.

In the roof detection system [25], which improves the accuracy of a previously designed system, the features for the best classifiers are selected by a human. Even the training process is manual, since a human responds to questions by system, whether or not offered windows are roofs. The system only adjusts the weights of selected features,

using nearest neighbour techniques. Thus the machine learning component here is quite limited.

2.4.1 Recognizing cars by shape matching

Thureson and Carlesson [36] describe an approach that tries to discriminate between classes of objects instead of finding instances of the same object. They do this by comparing gradients of test images to pre-determined gradient template regions that identify classes of objects. This is seen more clearly in Figure 2.15. In this figure, we see that a template gradient map of a cup, a car and a person walking was established. These templates were compared to images in the test set by shape matching. Their categorization of cars is interesting, but only moderately successful. They were able to detect 90% of the same cars seen from the same viewpoint (side) using the windshield and tires gradient map (which is applied in the shape matching process). The problem is that other cars have differently shaped windshield and tire pairs. Their results for different models of cars vary between 60 – 85%. This approach is interesting since it avoids AdaBoost yet still attempts to quantitatively describe the side of a car (among other objects) using shape detectors.

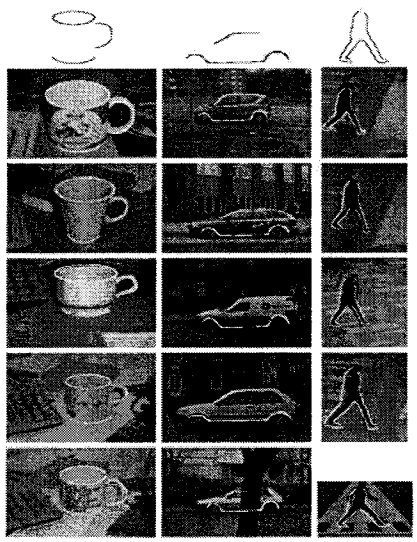

Figure 2.15 – Shape matching for cups, cars, and pedestrians

2.4.2 Recognizing cars by nearest neighbour matching

Petrovic and Cootes [29] researched the most similar problem to the one discussed in this thesis that we could find. They develop a system that automatically finds fronts of cars and is able to detect the class of car. In other words, the authors claim that the system is capable of identifying cars in their parking lot test cases, and identifying the makes and models of the cars (i.e. Fiat Punto, Peugeot 106...). Their solution involves finding the licence plates first, and then expanding the search area around the license plates to identify the car. In the first version of their program, they manually select the license plate, but an automatic license plate detection system based on corner detection was also discussed. The training set of positive examples contains one car per class of car. The images are frontal shots of the cars, and do not vary more than 5 degrees in any rotational sense. See fig 2.16 for details.

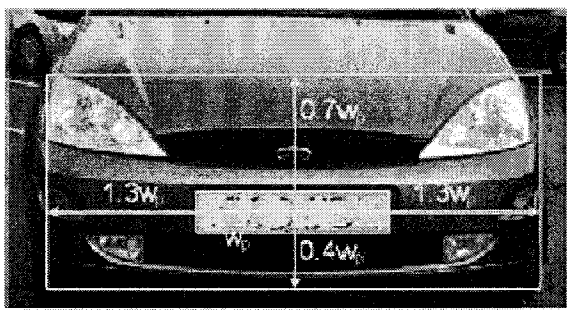

Figure 2.16 – Front of test car

A number of feature vectors are extracted from the area of interest seen as the white rectangle in Figure 2.16. The authors claim that the best feature vector to use as a comparison basis (via a nearest neighbour method) is square mapped gradients defined by the following equation:

$$\left(g_x^{SM}, g_y^{SM}\right) = \left(\frac{s_x^2 - s_y^2}{s_x^2 + s_y^2}, \frac{2s_x s_y}{s_x^2 + s_y^2}\right)$$

The equation seen above defines a kind of orientation map. Histograms were not applied. As an alternative to this direct mapping method, a statistical approach using Principal Component Analysis (PCA) was used to determine a low dimensional optimal structure

that contains the majority of between class variations. The authors claim that they are able to distinguish between 77 different classes of cars, and are able to successfully identify and detect cars 93% of the time if the license plate is manually located, and 87% of the time if the system functions autonomously.

There are several points worth mentioning. The authors admit that the system is very sensitive to even the slightest rotation or viewpoint change of the camera. This is due to the nature of their solution. Nearest neighbour approaches are notoriously sensitive to such changes in viewpoint especially for direct mapping approaches. Also, the authors make no mention of how fast their system works. It can therefore be concluded that it is not a real time system as we are proposing. It is certainly not a real time system for the variant using PCA. Therefore, this system is related to our discussion here, but it falls short of the real time constraint, and does not use machine learning.

Chapter 3 - Fast AdaBoost based on a Small Training Set

This chapter describes all of our contributions in detail and presents our system for detecting cars in real time. This is done in parallel with presenting implementation details, and the 'history' of the design of our classifier. We have revised the AdaBoost based learning environment, for use in our object recognition problem. It is based on some of the ideas from literature, and some new ideas, all combined into a new machine.

Our feature set initially included most of the feature types used by Viola [39] and Lienhart [18]. The set did not include rotated features [18], since the report on their usefulness was not convincing. Edge Orientation Histogram (EOH) based features [21] are considered a valuable addition, and were included in our set. We also added new features that resemble the object being searched for, that is, custom made features.

Viola [39] and most followers used weight based AdaBoost, where the training examples receive weights based on their importance for selecting the next weak classifier, and all weak classifiers are consequently retrained in order to choose the next best one. We believe that it is better to rely on the Fast AdaBoost variant [42], where all of the weak classifiers are trained exactly once, at the beginning. Instead of the weighted error calculation, we believe that it is better to select the next weak classifier 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 weak classifiers. Each selected weak classifier 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' being applied here.

Our variant of the AdaBoost learning algorithm is similar in flavour to the alternative voting AdaBoost variant described by Wu, Rehg and Mullin in [42]. Both algorithms train WCs only once, at the beginning. There are several important differences

in the two methods. In [42], each feature is trained so that it has minimal false positive rate. In our variant, each feature is trained to minimize a single combined error, which includes both false positives and missed positives. Again, this error is equal to the number of misclassifications divided by the number of all training examples. In [42], a new feature is added to either minimize the false positive or maximize the detection rates, depending on the current detection rate. In our variant, a new feature is always one that minimizes the combined error of the classifier. Next, the decision of a classifier in [42] is made by majority voting (where each WC has equal weight). In our approach, the weights of each WC are decided at the beginning of the training process, and the decision of each classifier is made by weighted voting. Next, [42] considered different weights for positive and negative examples. We considered only equal weights, although a similar change of weights can also be incorporated into our framework. Finally, our variant is a single strong classifier while [42] described a cascaded design.

We have 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 [34], 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 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. We therefore described an AdaBoost machine based on a fuzzy WC. More precisely, we propose a specific function for making decisions, while [34] left this choice unspecified. Our system produces a 'doubly weighted' decision. Each weak classifier receives a corresponding weight α, then each decision is made in the interval [-1, 1]. The weak classifier 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.

Several connected pieces of software were developed to make this project whole. The largest of these was the training program which produced the series of best weak classifiers. The analyzer program is the application that scans individual images and marks rectangular regions where the Hondas are found. It uses the series of weak classifiers determined by the training program. There were also two other small programs developed which helped in the creation of the negative training set.

In the remaining sections of this chapter, we discuss how we generated the negative training set and its impact on the machine design, and how we reduced training time by selecting a subset of features in a pre-processing step. We then describe the features used in our design, fuzzy WC, the design of strong classifiers (with both binary and fuzzy WCs), training of optimal weak classifiers, and training of the best classifier.

3.1 Generating the positive training set

All positives in the training set were fixed to be 100 x 50 pixels in size. The entire rear view of the car is captured in this window. Examples of positives are seen in Figure 3.1. The width of a Honda Accord 2004 is 1814cm. Therefore, each pixel in each training image represents roughly 18140/100 = 18.14cm of the car.

Figure 3.1 - Positive training examples

A window 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 used similar logic in determining his training example dimensions. All positives in the training set were photographed at a distance of a few meters from the camera. The camera used for the construction of the positive training

set was a JVC 3.3 mega pixel digital device. There is no database of rears of Honda's on the internet, so the training set was constructed manually by hunting for positives in various parking lots. The construction of the training and testing sets took roughly one month. This lengthy process of finding rare objects limits the efficiency of most machine learning approached for Vision applications since most of them rely on hundreds, if not thousands, of positive training examples.

3.2 Generating the negative training set

Apart from adding examples of other cars to the training set, we implemented a program that takes a single large image and a window of size *100* x *50* pixels (the same size as all of the other training examples) as input, and generates many smaller *100* x *50* pixel images as output. It does this by placing the window over a region of the large input image, and saving the resulting region of interest (ROI). It then shifts the window to another location in the image, and repeats the saving procedure. For an input image of size *ixj*, up to *(i-100)* x *(j-50)* sub images can be generated. We have used this program only in the early stages of negative training set development.

The second program for creating negative training examples is unique in its capacity to use the results of the analyzer program to create negative examples. To clarify, the analyzer program is run with a given strong classifier on images that are known to contain no positive examples (in our case Hondas). If the analyzer finds areas it deems positive in theses images, they are saved and added to the negative training set and the training program is rerun on the updated data set. We applied this process, which corresponds to bootstrapping in literature (e.g. [33]). It was used in our case for generating specific negatives, but tries to avoid the overcrowding of the negative set with redundant examples. It takes some time and a few rounds of training to adjust the negative training set in such a way that it produces acceptable results. We call this a cascaded training procedure since the next round of training uses the errors generated in the current round of training. As the strong classifier evolves, the number of new false

positives that the specific negative generator comes upon decreases, since the accuracy of the strong classifier itself increases over time. Some negative training examples are seen in Figure 3.2.

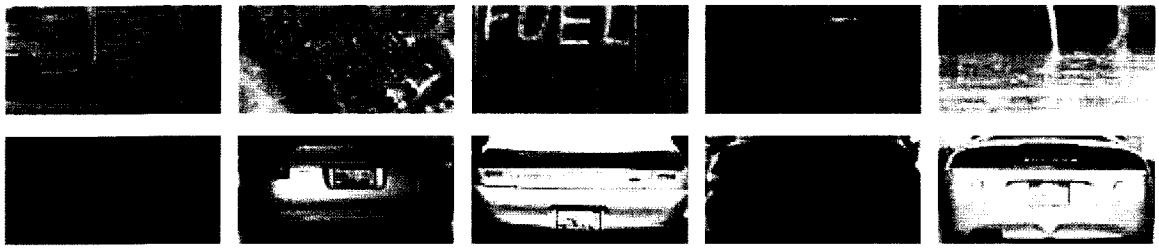

Figure 3.2 - Negative training examples

The training sets that are used greatly impact the training speed, and the accuracy of detection. 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. We say that a Honda accord looks like another Honda accord because we see certain visual cues they share. 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 colours, shapes and edge quantities and orientations. We again remind ourselves that Viola (and many others) used a brute force method of generating negatives by extracting them from a few large images. One of the major problems with using a brute force method for generating negative examples is that negatives extracted from a single image are very similar. They therefore obscure the separating ability of an individual weak classifier. If many of the examples in the negative set are similar, then a given WC can initially be perceived to have a greater ability to distinguish between the positive and negative sets since it can group together similar negative examples and quickly eliminate them. It creates a false representation of the actual accuracy of a classifier. Weak classifiers seem to be able to better separate the two sets, but in reality the results are misleading. To illustrate this point, imagine a negative set of images that are identical. Just one classifier is enough to distinguish between the positive and negative sets with a 100% detection rate, and a 0% false positive rate. When this 'strong classifier' of size 1 is tested on a reasonable test set, it will have a very difficult time

confirming its detection and false positive rates achieved in the training procedure. To overcome this problem in his brute force method of generating negatives, Viola just kept adding to his negative set in hopes of finally offsetting this unfortunate circumstance. The final result of his brute force methodology faced him with a negative set of 10,000 images. This, in combination with his choice of features led him to a training time in the order of months! He was also forced to use roughly 6000 WC to achieve the desired detection and false positive rates.

3.3 Reducing training time by selecting a subset of features

Several obstacles had to be overcome in order for the training algorithm to work. Most of these obstacles came in the form of memory management. The computer that was mainly used in the coding and testing of all programs is an Intel Pentium 4, 3.0 GHz machine with 2GB of RAM. To better illustrate the issues involved with the limited memory, the training parameters of Viola's [39] work will be contrasted with those of our own work. The most obvious difference is the size of the training examples. Viola's faces were *24* x *24* pixels each. Our training examples are *100* x *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 7 basic features used by Viola. They generated a total of 180,000 WCs [39]. We also used 7 basic features (as described below), and they generate a total of approximately 6.5 million WCs! There are 6.5 million WCs since 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. It was found that our computer cannot store more than 240,000 WCs in memory. This created problems that were elegantly solved. 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. Weak classifiers that had a *ce* that was greater than a pre-determined threshold were automatically eliminated from further consideration. This is automatically done in our program, whereas Viola dealt with excess immaterial features visually before training began. The total number of possible features in Viola's implementation was roughly 180,000. Viola placed restrictions on the sizes of features in weak classifiers to reduce their number to 100,000. The main reasons for this are not mentioned in his work, but they amount to faster training time, and more importantly conservation of memory.

In our implementation, no WC could have a worse cumulative error than *min(n,p)/q*. This is due to the fact that the threshold Θ is initially inserted before the beginning or after the end of the sorted list of training records in each weak classifier. This means that it initially classifies all records to be either all negative, or all positive, respectively. If there is no better place for Θ within the list of training records, it remains where it was first placed, with the *ce* that it was originally awarded. The pre-determined threshold mentioned above was set as *min(n,p)/q-0.01* min(n,p)/q*. This means a minimum of 1% improvement over the trivial initial error was needed for a WC to be accepted into the next round of selection. Luckily, the distribution of the WC's is heavily biased past the *min(n,p)/q* boundary. This means that most weak classifiers are very poor, and have the maximum *ce* (which is equal to *min(n,p)/q*) which means that they were eliminated early from the training procedure. This distribution of weak classifier efficiency is illustrated by the training results of the best strong classifier generated by our program. As previously mentioned, we deal with roughly 530,000 weak classifiers in our training program. For example, let us assume that there are 150 positive and 750 negative examples in a training set. Therefore, the greatest cumulative error any one of the weak classifiers could be assigned is approximately 16.67%. We adjust our threshold to accept all weak classifiers that have a cumulative error equal better than 16.67-0.01*16.67=16.5%. According to our experiments, it turns out that there are only 15000 weak classifiers that satisfy this requirement out of a total of 530,000. That means that over 97% of all initial weak classifiers are eliminated from further training. This drastically reduces training time while having little impact on the pool of available weak

classifiers to choose from. The final results of the strong classifier do not suffer from this reduction of unnecessary weak classifiers as is evident from their high detection rates and low false positive rates. The system would not work if the classifiers had performance less than 50%. In fact, AdaBoost machine learning theory is built on the premise that the chosen classifiers have performance slightly better than chance.

3.4 Features used in training

We will give more details on the features used in the training procedure here. We first attempted to solve the problem by Viola's features alone, believing that this would suffice. However, we noted that all these features were more or less random classifiers. For example, the main feature that Viola used around the eyes on the face showed no significance at all when it was applied to recognizing cars. AdaBoost theory specifically asks for weak classifiers that are better than random to be useful; real time performance is another constraint. Performance in practice is greatly impacted by many factors. A set of Viola's features might accurately detect cars in finite time, after introducing thousands of features (which implies non-real time, even if they are cascaded), after a training time expressed in months.

We applied the following strategy. First we decide what it means to have 'real time' performance. We choose that this means roughly one second. Then we calculate, empirically (by measurements), how many features a strong classifier could run per standard size image within one second. Suppose that it means 100 features. This becomes the limit in our classifier when training it. Then we run Viola's features only. The training program always stopped at 100 features without reaching the desired detection and false positive rates. Then we checked the accuracy of such a classifier and get unusable results: the classifier had an abysmal detection rate, and reported thousands of false positives. Simply speaking, 100 features were too few to make it useful.

Viola's Haar wavelets were therefore eliminated from the training procedure completely early in the implementation and testing phase since they failed to produce any usable results. It was even empirically shown that they are useless, since they did not pass the 'triviality' test. Much programming effort went into incorporating them into the training framework. Not only were their cumulative errors in the training process too great to be useful, but their very presence in the feature set greatly, yet fruitlessly, increased training time. Vice versa, the redness feature, which proved very useful for recognizing almost all types of cars, is perhaps irrelevant for recognizing faces.

Two types of basic features were used. They were redness features, and dominant edge orientation features. They proved themselves to be much better than Viola's original set. This only confirmed our view that specific types of features need to be used in specific applications. Not all features are equally useful.

To the best of our knowledge, nobody else has eliminated Viola's features from their set. They added some features that were important for certain objects, but did not show that the system can work well without them. This is an additional contribution.

3.4.1 Importance of good features

Levi and Weiss [21] 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 one that separates the positive and negative training sets well. We applied the same ideology here in hope of saving time in the training process. Initially, all of Viola's features were used in combination with the edge orientation features proposed by [21] and the redness features proposed by [23]. It was determined that the training procedure never selected any of Viola's greyscale 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 colours and these colours 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 taillights 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 WC's in the strong classifier. The fact that the body of the Honda accord comes in its own subset of colours presented problems to the greyscale set of Viola's features. When these body colours are converted to a greyscale space, they basically cover the entire space. No adequate threshold can be chosen to beneficially separate positives from negatives. Subsequently, all of Viola's features were removed due to their inefficiency.

3.4.2 Redness Features

The redness features we refer to are taken from the work of [23]. They concentrated on finding circular red regions in images. Their goal was to find and fix red eyes in pictures taken of people. Most of their work focused on the shape of the red regions found, rather than the techniques involved in finding the colour red. We borrow their idea of finding predominantly red regions. Our work differs in the fact that we look for red regions that signify the stop lights of the Honda accord, as opposed to human eyes. Therefore, our red regions are rectangular, and much larger than theirs. Fig 3.3 shows an example of a redness feature determined by the training process.

Figure 3.3 – Redness feature

A special 'redness' colour space was formed during pre-processing to assist in the detection of red areas. This colour space was taken from [23], and is a one dimensional linear combination of the RGB colour space. All of the positive and negative inputs in the training set are RGB colour images which means that each of their pixels is represented by three 8-bit numbers. These three 8-bit numbers represent the quantity of red, green and blue in a pixel, respectively. Each pixel in the redness colour space is defined as follows:

$$redness_{x,y}(v) = 4 \times input_{x,y}(R) - 3 \times input_{x,y}(G) + input_{x,y}(B)$$

Different combinations of RGB values can be mapped to the same values in the redness space. Some of these combinations might not be red at all, yet map into a very red part of the space. This is taken into consideration in [23], yet the results obtained by using this scheme do not seem to be influenced by this irregularity. The integral image computation was applied to the redness image to produce one of the inputs to the training procedure. The areas of the redness training features were determined in constant time using the integral image of the redness image.

3.4.3 Edge Orientation Features

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 $3px$ x $3px$. It has two configurations, one for finding edges in the x-direction, and one for finding edges in the y-direction of source images ([10], p. 165). These two matrices, h_x and h_y, (shown in figures 3.4 and 3.5) are known as the Sobel kernels.

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

Figure 3.4 – Kernel h_y

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

Figure 3.5 – Kernel h_x

Figure 3.4 shows the typical Sobel kernel for determining vertical edges (y-direction), and Figure 3.5 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 greyscale version of the input image. Greyscale images are determined from RGB colour images by taking a

weighted sampling of the red, green and blue colour spaces. The value of each pixel in a greyscale image was found by considering its corresponding colour input intensities, and applying the following formula: *0.212671*R + 0.715160*G + 0.072169*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 3x3 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:

$$X(i,j) = h_x \cdot I(i,j) = -P(i-1,j-1) + P(i+1,j-1) - 2P(i-1,j) + \\ +2P(i+1,j) - P(i-1,j+1) + P(i+1,j+1)$$

$$Y(i,j) = h_y \cdot I(i,j) = -P(i-1,j-1) - 2P(i,j-1) - P(i+1,j-1) + \\ +P(i-1,j+1) + 2P(i,j+1) + P(i+1,j+1)$$

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 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 greyscale 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. Figure 3.6 shows a picture of a farm house, and Figure 3.7 shows the same picture after the Sobel edge detector has been applied to it (Laplacian image). 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 [21].

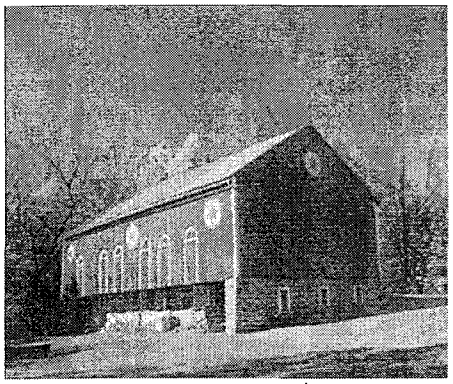

Figure 3.6 Farm House [10]

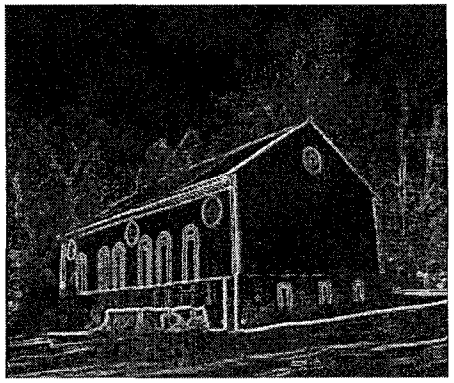

Figure 3.7 – Sobel edge contours [10]

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

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

This formula gives the orientation of each pixel in degrees. The orientations are divided into 6 bins so that similar orientations can be grouped together. The whole circle is divided into 6 bins.

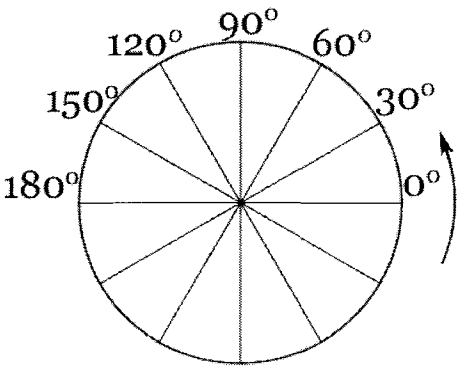

Figure 3.8 – Pixel orientations

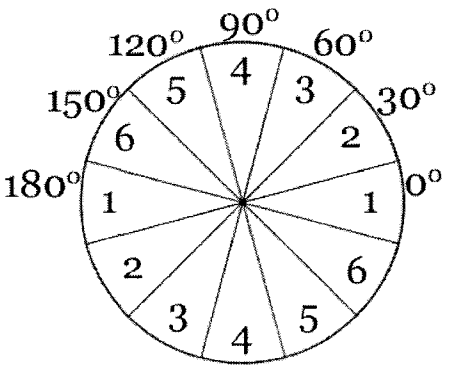

Figure 3.9 – Six orientation bins

It is important to note that the orientations of the 0^0, 90^0, and 180^0 bins are critical in identifying Hondas. They are important since Hondas mainly have horizontal and vertical

edges. The division of the bins in the fashion depicted in Figure 3.8 poses problems since all vertical and horizontal edges can fall into two bins. The vertical orientation, for example, is exactly a bin boundary. We handle this problem by shifting all of the bins by 15 degrees. Note that [21] did not mention any bin shifting, so we believe that they used bins as defined in Figure 3.8. They did however vary the number of bins from 4-8. They did not specify which number of bins worked the best for them. Effectively the number of bins does not impact the performance much, but these boundaries are avoided. To fit all of the orientations into the $0\text{-}180^0$ range, we add 180^0 if the angle is smaller than 0^0, and subtract 180 if the angle is greater than 180^0. The effects of these transformations can be seen in Figures 3.9. In Figure 3.10, we see a Honda accord and its corresponding edge orientation image for the first bin $[-15^0,+15^0]\cup[+165^0,+195^0]$.

Figure 3.10 – Honda and corresponding horizontal edges

Figure 3.11 – Edge orientation bins [2..6] for Honda in Figure 3.10

Since we use 6 bins, we create 6 images where each image represents an edge orientation bin. Each value *B(i, j)* of each orientation bin image is *R(i, j)* if *orientation(i, j)* falls into this bin, and 0 otherwise.

As we can see from Figure 3.11, in a given region in an image, there is typically one orientation that is dominant. We exploited this fact in our use of dominant edge orientations. This idea was first developed by [21]. Dominant edge orientations are calculated as the total edge intensities of a given orientation divided by the sum of all edge intensities of all orientations in the same region. Dominant edge orientation features were used for training. We see some less distinguishing, yet nonetheless noticeable edges

in some of the other bins in Figure 3.11. One of the most successful edge orientations was the horizontal one depicted in figure 3.10. All of the possible dominant edge bins were offered to the training procedure.

Integral images were created from these orientation bin images and were used in the training procedure to find sums of edge intensities. A cumulative integral image of all of the edge orientations was also stored to speed up the processing of this type of feature. 7 features were used in the training procedure. They were: dominant edge orientations (1, 3, 4, 5, and 6) and the redness feature.

3.5 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 a *fuzzy WCs* [34] 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 weak classifier* 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, 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.

θ_{mn} $f(x)$ θ θ_{mx}

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.

3.6 Strong Classifiers

A strong classifier is obtained by running the AdaBoost machine. It is a linear combination of weak classifiers. We assume that there are T weak classifiers in a strong classifier, labelled $h_1, h_2, ..., h_T$, and each of these comes with its own weight labelled $\alpha_1, \alpha_2, ..., \alpha_T$. The tested image x is passed through the succession of weak classifiers $h_1(x), h_2(x), ..., h_T(x)$, and each weak classifier 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:

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

From this equation, we see that images that pass (binary or weighted) weighted recommendations of the weak classifier tests are catalogued as positive. It is therefore a (simple or weighted) voting of selected weak classifiers. 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 α_i = -log (e_i/(1-e_i)). Each e_i is equal to the cumulative error of the WC.

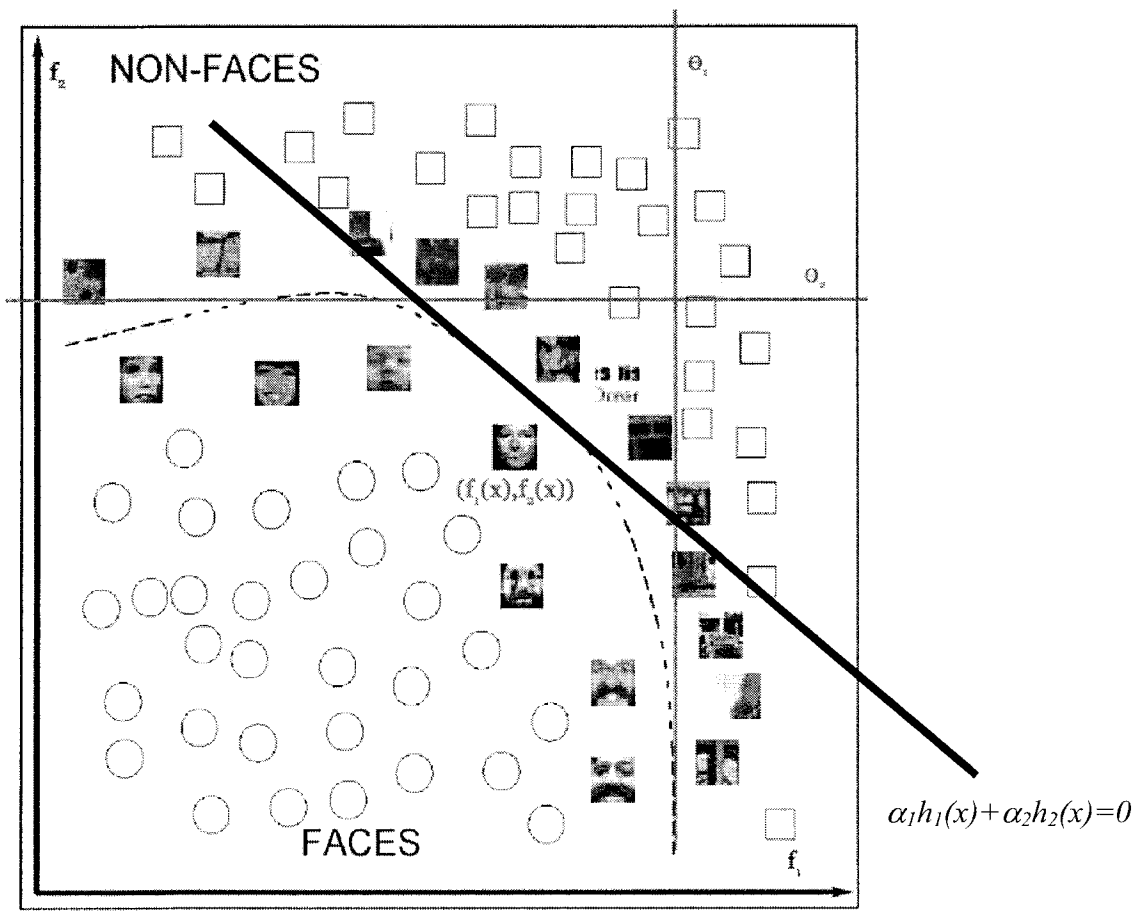

Figure 3.12 - A strong classifier composed of two weak classifiers

The classifier consisting of fuzzy WCs can be interpreted as follows. We use Fig. 3.12 as an illustration with two features. The feature values are first calculated with respect to the thresholds rather than the original axes, effectively placing the origin at the intersection of the threshold hyperplanes (θ_1, θ_2). The obtained coordinates are then scaled with respect to the new origin, so that all images are placed within the square of edge length 2 centered at the new origin. This describes a mapping from feature values $f(x)$ into WC values $h(x)$. In the new space, the classifier decision is based on a hyperplane. For two features, as in Fig. 3.12, positive and negative examples are separated by a straight line $\alpha_1 h_1(x)+\alpha_2 h_2(x)=0$. The hyperplane can be translated, if needed, to separate two sets using line $\alpha_1 h_1(x)+\alpha_2 h_2(x)= \delta$. We also see that the images in the triangle in the middle of Figure 3.12 are classified as positive by the discrete classifier, yet this area could be theoretically correctly classified by a fuzzy classifier.

3.7 Training Optimal Weak Classifiers

In our algorithm, all weak classifiers are trained only once, at the beginning of the training process. They do not change in the process afterwards, therefore the needed values can be memorized. The input consists of feature f and all positive and negative examples. Let q be the total number of images (training examples), and let p and n be the total numbers of positive and negative images, respectively, $q=p+n$. The algorithm scans through the sorted list of feature values, looking for threshold θ and direction s that minimizes the classification error, which is the total number of misclassified examples. The output is specified below.

Algorithm: Training optimal weak classifiers
Input: Feature f, n negative examples, p positive examples,
Output: Threshold Θ, sign s, interval $[\theta_{mn} .. \theta_{mx}]$, *false_pos*, *missed*, weight α.

Calculate records $(f(x_i), y_i)$, where $y_i=1$ for a positive example, and =-1 for a negative example (using integral images where appropriate). Sort these records by the $f(x_i)$ field by any sorting algorithm, e.g. 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 $(g_j, status(j))= (f(x_i), y_i)$, where $g_j=f(x_i)$. That is, if the j-th element g_j is equal to i-th element from the original array $f(x_i)$ then $status(j)=y_i$.

$sp \leftarrow 0$; $sn \leftarrow 0$; (*number of positives/negatives below a considered threshold *)
$s \leftarrow 1$;
$\theta_{mn} \leftarrow g_1$;
$\theta_{mx} \leftarrow g_q$;
If $n < p$ **then** {$misclassified \leftarrow n$; $\theta \leftarrow g_q+1$} (*all declared positive*)
else {$misclassified \leftarrow p$; $\theta \leftarrow g_1-1$ }; (*all declared negative *)
For $j=1$ **to** $q-1$ **do** {
If $status(j)= 1$ **then** $sp \leftarrow sp + 1$ **else** $sn \leftarrow sn + 1$;
If $sp + n - sn < misclassified$
then { $misclassified \leftarrow sp + n - sn$; $s \leftarrow -1$; $\theta \leftarrow (g_j+g_{j+1})/2$
$false\_po\ s \leftarrow n - sn$; $missed \leftarrow sp$ };
If $sn + p - sp < misclassified$
then { $misclassified \leftarrow sn + p - sp$; $s \leftarrow 1$; $\theta \leftarrow (g_j+g_{j+1})/2$;
$false\_pos \leftarrow sn$; $missed \leftarrow p-sp$ }
};

The output is a weak classifier $h(x)=h_i(x, f, s, \theta, \theta_{mn}, \theta_{mx})$. Note that θ_{mn} and θ_{mx} are only used by fuzzy WCs, if they are applied. They denote the extreme values for the corresponding feature. If $s=1$ then $\theta_{mn} =g_1$, $\theta_{mx} =g_q$, otherwise $\theta_{mn} =g_q$, $\theta_{mx} =g_1$. The detection rates and false positive rates of weak classifiers can also be considered output at this stage, as $(p- missed)/p$ and $false\_pos/n$, respectively. Variables *missed* and *false_pos* denote the number of misclassified positive and negative examples, respectively.

The relative error of the constructed WC is $e=misclassified/q=(false\_pos+missed)/(p+n)$, and is used to decide the weight of the constructed WC as follows:
$\beta = e/(1-e)$, and $\alpha = -\log(\beta)$. The assigned weight is α.

End of Algorithm.

Since $misclassified<(p+n)/2$, it follows that $misclassified/(p+n)<0.5$, and we therefore have that $e<0.5$ since $e=misclassified/(p+n)$. The range of e is between [0, 0.5). Error e is 0 when a classifier is able to completely separate the two sets (positive and negative) without error. AdaBoost theory proposed by Freund and Schapire states that all weak classifiers that are incorporated into the strong classifier have to have performance slightly better than chance (0.5). This means that each weak classifier has performance at least > 0.5, which means that it has error < 0.5. Therefore our algorithm satisfies the theoretical condition since classifiers that do not satisfy this norm are not taken into consideration. In fact, due to the incorporation of the acceptance threshold, WCs with error rates greater than $min(n,p)/q - 0.01* min(n,p)/q$ are pre eliminated. In our implementation, this means that the greatest acceptable error per WC is roughly $min(750,155)/905 - 0.1*min(750,155)/905 = 0.17$. Parameter $\beta=e/(1-e)$; therefore, when $e=0$, $\beta=0$ and $\alpha=+\infty$. When $e=0.5$, $\beta=1$ and $\alpha=0$. β is in the interval [0,1]. Parameter $\alpha=-log(\beta)$. Since $\beta \in [0, 1]$, $\alpha \in [0, +\infty]$.

AdaBoost therefore assigns large weights with each good weak classifier and small weights with each poor weak classifier.

3.8 Training the Best Classifier

First, all weak classifiers (WCs) are trained, as described, and the training process returns classification errors *missed* and *false_pos*. It also returns the weight α for each WC $h(x)$. The combined error can be defined in one of several ways, such as $ce=(missed + false\_pos)/q$, $ce= missed/p + false\_pos/n$, $ce= \lambda\, missed/p + (1-\lambda)\, false\_pos/n$, where λ

is a weighting parameter. In our implementation, we use the trivial *ce=(missed + false_pos)/q*. This scheme was used since it is simple, and equally treats all examples. Since our training set is rather small, it was not necessary to adjust the weights of the training examples. For each feature, find the optimal weak classifier as described above. Then the construction of a classifier proceeds as follows.

Algorithm: Training the best classifier

Input: set of weak classifiers $h_i(x)$, weights α_i, n negative examples, p positive examples

Output: series of selected weak classifiers h_1, h_2 ... h_T, and their weights α_1, α_2 ... α_T.

Select the first WC $h_1(x)$ (and its weight α_1) as the one that has

minimal combined error *ce*;

Set $T=1$; (* the number of WCs in the classifier *)

Repeat

For each WC $h(x)$ calculate the combined error of the classifier

$$\alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots + \alpha_T h_T(x) + \alpha h(x) \text{ (with } \delta=0)$$

and select $h(x)$ that minimizes the error; find its weight α;

$T=T+1$, $\alpha_T=\alpha$, $h_T(x)=h(x)$;

Until (detection rate $(p- missed)/p \geq d$ and false positive rate $false\_pos/n \leq fp$) or $T \geq Tmax$.

End of Algorithm.

Note that values $\alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots + \alpha_T h_T(x)$ can be memorized so that testing candidates is faster. In the test, *false_pos* and *missed* are the numbers of incorrectly classified negative and positive examples, respectively, by the tested classifier. This section of code is executed for every feature, and for every example in the training sets, up to T_{max} times. The method takes $O(Fq \log q)$ time to train all of the classifiers in the initial stage, where F is the number of WCs, and q is the number of examples. We are left with f WCs, where f<<F, after the elimination of poor weak classifiers that do not improve the cumulative error more than 1% from the trivial position. Testing each new weak classifier 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 weak classifier. Therefore it takes O(Tqf) time to chose T weak classifiers. We deduce that it would take O(Fq log q) + O(Tqf) time to complete the training using our method (the same time complexity applies to the variant described by [42]). Since $Tf<F$, the dominant term in the time complexity is O(Fq log q). Had F and f been roughly equal, the dominant term would have been O(Tqf).

Chapter 4 – Setting Parameters and Testing the Strong Classifier

In this chapter we describe the results of our experiments. This includes both the training and testing procedures. In the training program, we numerically show the impact of edge orientation bin shifting, and show the selection of the best threshold for feature acceptance.

After the training procedure is complete, we test the strong classifier on a variety of images. The input of the testing phase is a picture which may or may not contain the Honda accord. The output is the same picture with a red rectangle drawn around objects which are deemed to be Honda Accords. Figure 4.1 shows a typical input and its corresponding output.

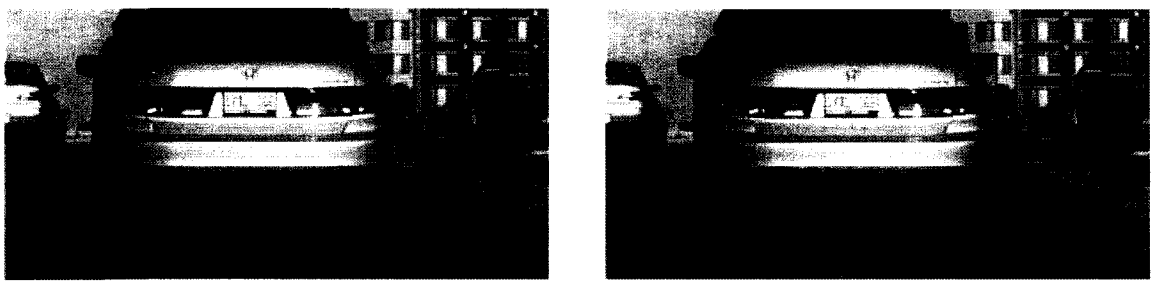

Figure 4.1 – Input and output of the testing procedure

All of the parameters in the final version of the program were set via a trial and error method. All of these chosen parameters were verified quantitatively and the results are presented in the following text.

4.1 Test Methodology

The test methodology we use to verify the claims in this work is parallel to other test methodologies in comparable works. The test data is evaluated in several ways, using

appropriate metrics. The detection rate and false positive rate are measured with respect to the number of Hondas in all images (we call this the per car evaluation), and with respect to the number of images containing at least one Honda (we call this the per image evaluation). In section 4.5, the number of false positives was evaluated against all tested sub windows (per sub window evaluation). Since there exist no standardized test sets for the detection of any cars, let alone one specific car, our machine was tested on one set that was created the same way the training set was created. Pictures were taken of cars and other objects around town. Our test set boasts 106 images which contain 142 cars seen from the rear in sizes detectable by our program. There were 101 positive examples of the Honda Accord 2004, and 41 examples of other cars in this set. We used only one test set since this is the norm in this field of research. That is roughly the same size test set as the standardized MIT CMU test set for faces. The positives in the set are in various scales and positions within the images. There are 62 instances of close ups of cars, and 80 instances where the cars are a further away from the camera. They are also taken from a variety of angles that are detectable by our program. The lighting conditions of pictures within the test set vary drastically; from very well lit to very dark. Most (83) images in the test set were well lit, while 23 were poorly lit. The lighting conditions were varied to test the robustness of the system. This set was put together in order to test in a meaningful way, with as much variance as possible, the performance of the system. The test set images themselves also come in a variety of sizes. The smallest images are 150 x 120 pixels. The largest images in the test set are 640 x 480 pixels. The standard size of image which is often used for this type of testing is 320 x 240 pixels and used by Viola as well. The interpretation of the results was done from two viewpoints: a top level approach taking into account usability of the system, and a technical viewpoint that discussed the impact of varying the parameters of both the training and testing phases, and how these changes impacted results.

Every image in the test set contains at least one car. Tables 4.1 and 4.2 precisely outline the test set that was used.

Images in total	Images including Hondas	Images including other cars	Images including multiple cars
106	98	34	33

Table 4.1 – Outline of test set

	Total number of cars	number of Hondas	Number of other cars
totals	142	101	41
close ups	62	45	17
far shots	80	56	24

Table 4.2 – Outline of test set: per camera distance

4.2 Experimental Results

This section will interpret the results of the program from a top down viewpoint; avoiding the implementation details and parameter adjustments. It will focus on a per image discussion, concentrating on the types of errors that were encountered and the reasons that they occurred. By error, we mean a false positive, which is a single area of a test image that was erroneously identified as a Honda by the program. The detection rate of the program is the number of correctly identified Hondas divided by the total number of Hondas in the training set.

Again, the test set consisted of 106 images, 98 of which contained at least one Honda. 86 of the 98 images that contained at least one Honda were correctly identified by the program. This means that the program has a per image detection rate of 87.8%. This is similar to the actual detection rate of 89.1%. There were a total of 26 false positive detections in the 106 images in the testing procedure. These false positive detections only occur in 22 images. The false positive detections can be broken down into two categories: detection of other cars and detection of other objects. There were only 3 false positive detections of other cars in the testing procedure. These false detections occur on 3 separate images. There were 41 other types of cars in the test set that were clearly visible, and they appeared on 39 separate images. Therefore, there was 3/39=7.7% per image false positive rate of other cars. This means that there is a 7.7% chance that an image containing another type of car would be mislabelled as containing a Honda. Per image,

the total false positive rate was 22/106=20.8%. Table 4.3 summarizes the per image results.

	Images with only Hondas	Images with Hondas and other cars	Images without Hondas	Total
Number of images	67	31	8	106
Images with correctly detected Hondas	59	27	NA	86
Detection rate	88.1%	87.1%	NA	87.8%
Images with falsely detected Hondas	6	15	5	26
False positive rate	9.0%	48.4%	62.5%	NA

Table 4.3 – Detection and false positive rates per image

Figure 4.2 illustrates the per image detection rates of the system for the following categories: images with only Hondas, images with Hondas and other cars, and total per image detection rate.

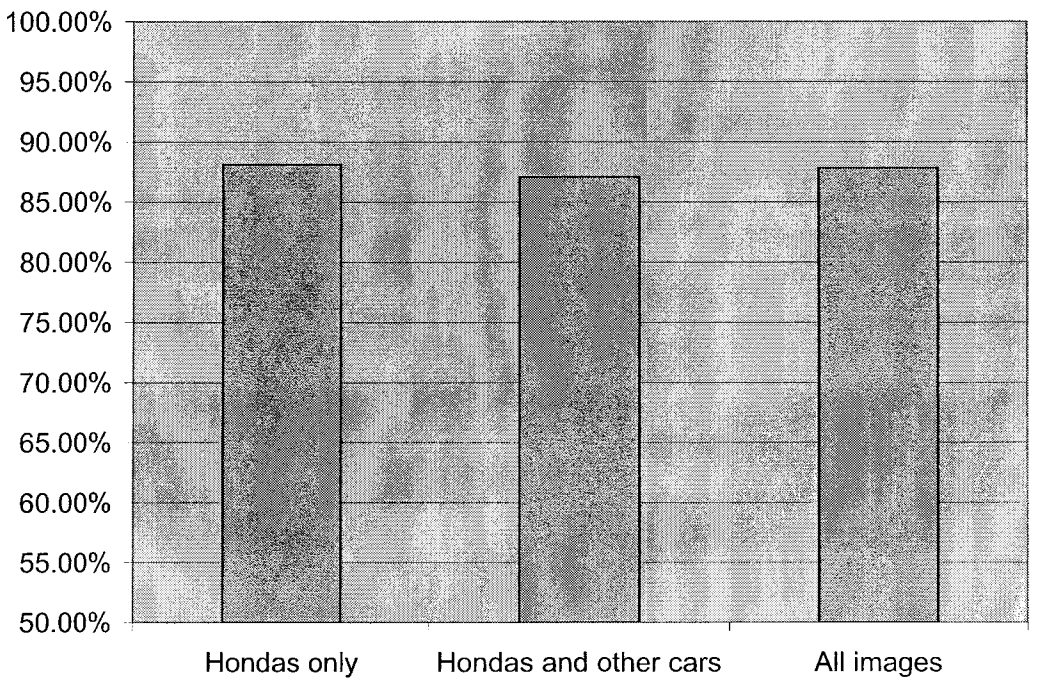

Figure 4.2 – Per image detection rates

When considering the false positive rate with respect to the number of cars, we observe that the program has a 3/41=7.3% chance of detecting another car as a Honda. That leaves 23 other objects being detected as Hondas by the system. To get a better feel as to how accurate the system is, we divide the number of false detections by the number of positives: 26/101=25.7%. This roughly tells us that for every 4 Hondas, we get a false positive. Table 4.4 summarizes the per car detection results.

	Number of Hondas	Number of other cars
Number of cars	101	41
Instances detected as Hondas	90	3
Percentage	89.10%	7.30%

Table 4.4 – Detection and false positive rates per car

The benefit of being able to detect the positives at various angles due to the selection of the weak classifiers also has its negative side. The weak classifier seen in Figure 4.7 and its symmetrical counterpart described in Figure 4.8 have a significant impact on the detection ability of the system as a whole. They are largely responsible for the flexibility in detection, and at the same time, are largely responsible for the false positive rate.

Figure 4.3 shows an example of a false positive detection. The reason this sub window was declared as positive is marked with the light green rectangles. These are the locations of a few of the most dominant weak classifiers in our strong classifier. They help to positively identify this sub window since they have large weights, and they all evaluate to true. The top two small rectangles find dominant edges in the 45 '/'and 135 '\' degree directions. The slender rectangular region (weak classifier) represents a horizontal dominant edge space.

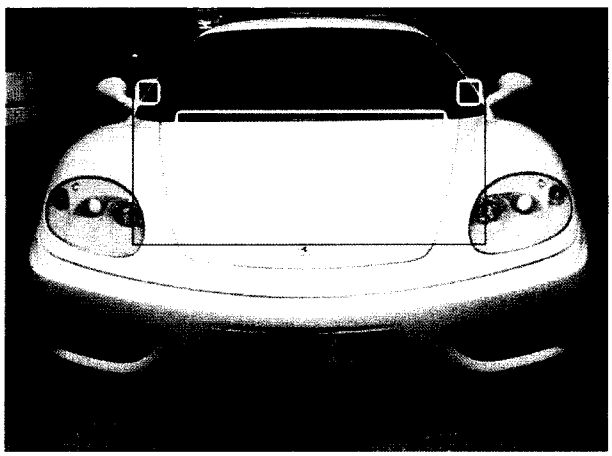

Figure 4.3 – False positive

Apart from the false positive examples having the characteristics defined in Figure 4.3, a few types of cars that are very similar to the Honda Accord 2004 are identified as positive. This includes the Honda Accord 2005. Actually the 2005 model Accord is virtually identical to the 2004 model, and they were interchangeable considered as positive. The 2003 Honda Accord model is similar to the 2005, yet it is not detected as positive. Examples such as the ones found in Figure 4.4 are a bigger problem. Another car that was found to be very similar to our positives was the Toyota Camry 2000. It can be seen in Figure 4.4. It has the same shape rear stop lights and the same characteristic edges on the top left and right sides of the body as the Honda accord 2004. Even to the naked eye, the two types of cars are very similar from the rear.

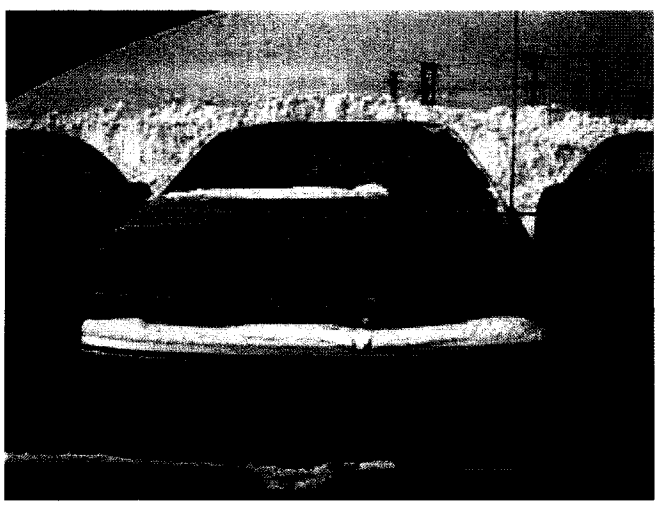

Figure 4.4 – Toyota Camry

Again, out of the 41 other cars in the test set, only 3 were detected as false positives by our system. This means that the system effectively recognizes Hondas, not just the existence of backs of cars.

4.3 Edge orientation bin shifting

One of the major contributions in this work is the shifting of the edge bins. It might seem like a trivial matter which would not impact the results of the algorithm much, but in fact it greatly contributes to the performance of the system. Two variants of the algorithm were trained and tested to illustrate this point: one with the bin shifting approach, and one without. Both systems were trained and subsequently tested on the same training and testing sets. Both systems are composed of 30 WCs. The results are seen in table 4.5.

Approach	detection rate %	false pos detections	false pos rate
bin shifting	89.1	26	1.48×10^{-6}
no bin shifting	74.3	168	9.55×10^{-6}

Table 4.5 – Bin shifting results

From this data, we see that a simple shift of the bin orientations significantly impacts the results of the system. Nearly 7 times as many false positive detections are reported by the non bin shifting method. The detection rates also noticeably differ.

4.4 Acceptance Threshold

Table 4.6 shows the results of various acceptance threshold parameters that were tried. All of theses cases were trained and tested on the same sets, and only their acceptance threshold was changed. We have selected the 1% acceptance threshold which appears to be the best choice.

Acceptance Threshold	WCs after acc. threshold	detection rate	false pos detections	false pos rate
0.50%	35021	82.1	23	1.31×10^{-6}
1.00%	16044	89.1	26	1.48×10^{-6}
1.50%	15321	87.1	29	1.65×10^{-6}
2.00%	14816	89.1	38	2.16×10^{-6}

Table 4.6 – Acceptance threshold comparison

The thresholds that we see in Table 4.6 are the amounts the acceptance threshold was adjusted from the trivial position to accept WC into the second stage of training. The second column in Table 4.6 shows the number of WCs that entered the second stage of training. All of the training runs had roughly 530,000 WCs at their disposal at the beginning of training. It is interesting to note that the system generally improves in detection rate when it has fewer WCs to work with. This finding is counter intuitive since it is generally implied that the larger the choice of WC, the better the system will select the next best WC. On the other hand, the number of false positives increases with the reduced set of WCs. Since we are dealing with a relatively low number of WCs, their combination may result in a local maximum with respect to the combined detection rate and false positive rate. Our main motivation was to speed up training time, without sacrificing performance. In all cases, the training time was about 1.5 hours. There were no significant differences in training times because the training time was dominated by the first step: the evaluation of the 530,000 WCs.

4.5 Detection procedure and scaling sub windows

The detection procedure works much like the one described by Viola, except that there is no cascading of classifiers involved. This means that there were more operations per sub-window than in the first stage of Viola's detection procedure, but we use far fewer weak classifiers in our complete strong classifier to identify Hondas. In order to find Hondas of various scales, almost all of the sub windows of a picture must be tested. It was noticed at the beginning of implementation that the rear width of a Honda accord is

twice as long as its height. Therefore only sub windows of scales 2:1 are tested. The smallest possible window size for a recognizable Honda accord was set at 100 x 50 pixels. Our sub widows grow at a 10% increase in length and width after they are all tested at a given size on a given image. Therefore, we start testing all 100x50 pixel sub windows in an image, we continue with 110x55 pixel sub windows, and so on.

scaling factor	detection rate	false pos	sub windows	Run time units
5%	90.1	35	35173886	1
10%	89.1	26	17586943	1/2
15%	86.1	29	12556364	1/3

Table 4.7 – Scaling factor comparison

Table 4.7 shows the various sub window scaling factors that were tried. It is evident that a small scale increment such as 5% has a higher detection rate than the other two options, yet also has a higher false positive rate. It also tests more sub windows, which makes it slower. We chose the 10% sub window scaling factor since it balances both the detection rate and false positive rate well, and deals with a lower number of sub windows, which makes it faster. The 15% scaling factor is worse in both detection and false positive rates.

4.6 Testing individual sub windows

All of the weak classifiers are evaluated on each tested sub window. Each of them makes a (binary or weighted, depending on the choice of binary or fuzzy WCs) decision which compares the actual feature value for that sub window against the threshold for that feature. This decision is multiplied by the weight of the weak classifier which yields a real number between -1 and 1. All of these weighted decisions are added to form a final decision which classifies the tested sub window as positive or negative. If the final decision is greater than 0, the window is positive, otherwise it is negative.

It was a common occurrence in the testing phase that sub windows around an actual positive example would also be considered positive. All of these slightly offset rectangles from the main positive example would be drawn, and consequently clutter the image and make interpretation of the results very difficult. This problem was solved by storing the positive examples in a vector during the detection phase, and comparing new positives to the ones stored in the vector. If an already stored example was found that had a weaker decision than the new positive and was too close to the positive, it was replaced. Two positive rectangles are judged to be too close if their top left corners are within 60 pixels of each other. Different values were tried in the interval [50, 100] for this proximity parameter and this one was the best; however they all basically produced the same results. This parameter was an insignificant detail when considering the scope of the application. It is only used in producing more visually appealing results. This threshold was experimentally determined. Viola [39] took average corner values of overlapping positive windows to determine a single positive.

4.7 Shifting sub windows

The sub windows in the detection procedure were shifted by various amounts, and their corresponding accuracies are shown in Table 4.8. The first row of Table 4.8 represents the case where sub windows were shifted by 1 pixel. In other words, every possible sub window was tested. The second and third rows represent cases where sub windows were shifted by 2 and 3 pixels in each dimension (width and height) respectively. The last column of the table shows the number of sub windows that were tested in each case. They are proportional to the shifting parameters that were used. For example, row two is the case that was chosen in the implementation, and it deals with 2x2=4 times fewer sub windows than the case in row one. It is also 4 times faster to test a given image with this configuration, yet the detection rate is only marginally reduced. Moreover, the number of false positives is significantly reduced. Shifting sub windows 3x3 does not bring a significant reduction in false positives while reducing the detection

rate by about 3% with respect to the 2x2 shifting method. We had therefore chosen the 2x2 shifting method as the 'winning' option.

pixel shifting (*X*, *Y*)	detection rate	false positives	sub windows tested	Run time (time units)
1x1	90.1	41	69,928,779	1
2x2	89.1	26	17,482,195	1/4
3x3	86.1	24	7,769,864	1/9

Table 4.8 – Sub window shifting of pixels

The last column of table 4.4 represents the relative run time in time units it takes each of the variants to analyze one image. We see that shifting the sub windows by 2 pixels in each dimension results in a four fold improvement in run time. This is logical since there is also a four fold decrease in the number of sub windows that are tested.

4.8 Impact of image resolution on feature values

There were a few things to note when it came to adjusting the size of the sub window during the detection phase. When the sub window increases in size by factor *w* in each dimension, the features of each weak classifier within the sub window also increase in size by factor *w* in each dimension. This change has to be taken into account when we compare feature values of scaled WC's against their respective thresholds. When considering dominant edge orientations in a given region, we do not expect the ratio of the dominant feature in a region to change just because the region is larger. In a way, this dominant edge orientation is normalized and is not influenced by the size of the region it is computed in. One of the problems we anticipated was that edges might not be so clearly defined in larger examples of Hondas. By larger we mean Hondas that are larger than 100x50 pixels. In the examples in our training set, edges determined by the Sobel masks are very distinctive since the images are very small. We feared that in larger examples of Hondas, edges would be represented by thicker lines, and result in different edge orientation maps. This did not happen since Hondas have crisp, well defined lines. Even in larger images, edges are very similarly defined compared to those of smaller

images. Furthermore, it is the dominant edge orientation we are interested in when measuring feature values. Most of the weak classifiers in the strong classifier were in areas of the image in which their orientation was often the only one present. Therefore, any quantity of edges in such an area would be enough to help identify them positively. To illustrate this point, Figures 4.5 shows an image of a Honda and its corresponding horizontal edge orientation bin. The size of the original colour image was 420x210 pixels. It was slightly resized to better fit here. Figure 4.6 shows the same image of the same car as in Figure 4.5, but the resolution of the image was reduced to 100x50 pixels, and then resized to match Figure 4.5 for comparison purposes. We remind ourselves that training was done on images identical in resolution to Figure 4.6. The edge orientations in Figure 4.6 are far coarser than they are in its higher resolution neighbour, yet the end results are the same. The dominant edges that were isolated in the training process in the lower resolution images can be paired with corresponding edges in higher resolution images without many repercussions.

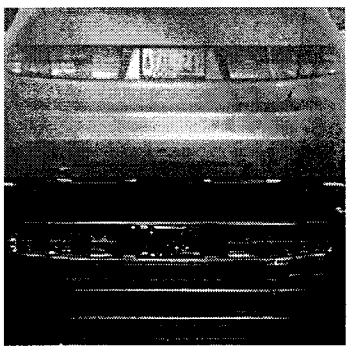

Figure 4.5 – Resized larger Honda

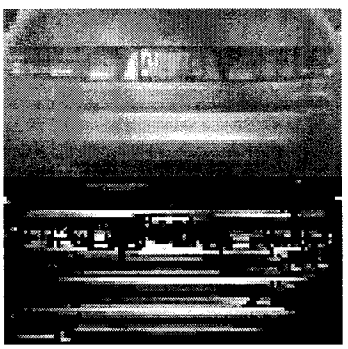

Figure 4.6 – Resized smaller Honda

The redness features are not normalized in any way, and the scaling of these features must be handled differently. Since the redness quantity in a rectangle directly depends on the area of the rectangle, threshold Θ of redness features must be adjusted to compensate for this area. That is why the corresponding Θ is multiplied by the square of the scaling factor before the scaled redness feature is compared against it. This operation normalizes the scaling effect.

4.9 Implementational Experimental Results

There exist no competing methodologies of solving the same problem in literature. We could not find a system that detects cars of a specific type while at the same time conforming to machine learning methodologies that perform in real time. The detection methods and results of systems that detect faces were used as a basis for performance comparison since they are the closest systems that satisfy all of the constraints presented in this work.

4.9.1 Selected Weak Classifiers

The training procedure produced interesting results when it came to the selection of weak classifiers. Figures 4.7 and 4.8 show the best and second best weak classifiers as chosen by the training algorithm.

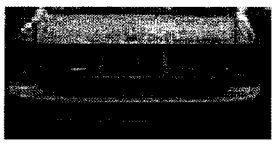

Figure 4.7 – WC1

Figure 4.8 – WC2

The best weak classifier as determined by the system was an edge detector applied to the upper-left hand corner of the Honda. The small green box in Figure 4.7 defines this weak classifier. It detects the 45 degree edge that is dominant in this region. After reflecting back on our test set, it became evident that most of the positive examples share this attribute. It is not a weak classifier that we would have chosen by hand had we tried static selection of features. The second best weak classifier was a redness feature that detected the rear stop lights of the Honda. The selection of this feature validated our assumption that a redness feature used for detecting the rear stop lights would be very important. This just further emphasizes our claim that the basic features selected for a given detection problem should be custom selected before training starts to produce better results down

the road. The other weak classifiers that were chosen identify many areas of horizontal edges that are common, redness features that define each stop light separately, and areas that do not have a specific orientation of edge. An example of such an occurrence is the area just below the license plates. This area is mostly void of any vertical edges.

The selection of the first few weak classifiers significantly influenced the detection and false positive rates of the strong classifier. They are also directly responsible for the degree of flexibility that the system has when it comes to detecting Hondas that are seen from an angle that is offset from normal. Figure 4.9 shows an example from our test set where a Honda is detected from an angle.

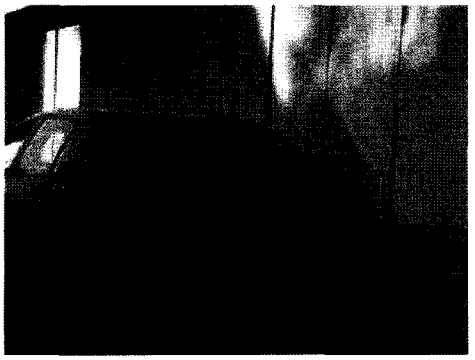

Figure 4.9 – Detection from an angle

Part of the reason that this Honda is detected by our program is that the top right corner of the detected region contains a clear edge symmetric to the one chosen as the best weak classifier in Figure 4.7. The edge orientation in this region is important to the detection process since one of the weak classifiers in the strong classifiers specifically looks for it.

4.9.2 Detection Performance

The detection rate for our classifier on the test set was 89.1%. There were 26 false positive detections among the 106 images tested. The false positive rate is therefore $26/17{,}500{,}000 \approx 1.5 \times 10^{-6}$. Viola [39] claims that a rate of roughly one false detection

per million sub windows is acceptable in a system. The results obtained in this thesis are therefore acceptable. These results were obtained using the binary version of our implementation of AdaBoost. These numbers are very good when compared to other systems such as those put forward by Viola and Levi & Weiss. The [39] and [21] systems were tested on a set that contained 130 images with 507 positives and a total of roughly 75,000,000 sub windows. Keep in mind that gathering such a test set is much easier when positives are faces. Our test has a similar number of images, yet a much smaller number of positives. Nevertheless, it is a sufficient comparison base to use as a basis for discussion. Figure 4.10 depicts the ROC curves for the car detector seen in this thesis, the face detector described in Viola's work [39], and the face detector in the work of Levi and Weiss [21], respectively. Following the ROC curve format of both sets of authors, the curves are produced by considering the number of false detections versus the detection rate.

Viola gave statistics for the number of false positives his system produced at various detection rates. At a detection rate of roughly 89% (such as our system), his system produced roughly 35 false positives. He however used a much greater number of classifiers to achieve this result. Levi and Weiss used the same test set to evaluate their system and they achieve an 89% detection rate at the cost of roughly 45 false positives. They used a 2500 item training database to achieve these results.

It is not as simple as saying that there are roughly 25% false positives in all three cases above since one test example is composed of roughly 60,000 searched sub windows. Out of a total of 106 * 60,000 = 6,360,000 sub windows of the test set, a total of 26 false detections occurred. That is roughly a 0.000004% false positive rate.

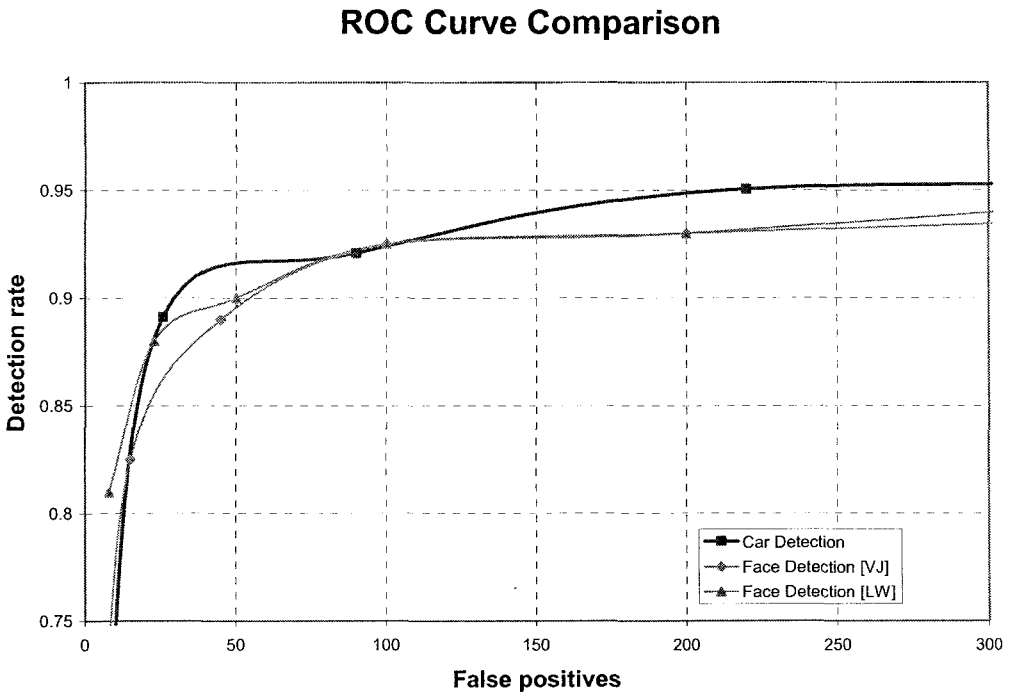

Figure 4.10 – ROC curve comparison

The fuzzy training procedure results were substantially weaker when the training procedure was trained on the same training set as the regular non fuzzy variant. The detection rate was 98.1% with a 0.5% false positive rate on the training set. It was 73.2% on the test set, but 411 false positive detections accompany this detection rate. The fuzzy strong detector was also composed of 30 weak classifiers. There are a few reasons the fuzzy variant did not perform as expected. First, the training set that it used was custom built for the non fuzzy variant. The semi testing set of random images was used in the cascaded construction of the training set. The non fuzzy procedure added false positives from the semi testing set based on its discrete criteria. The two variants were trained on the same set to force similar training times, and compare systems on the same input. Both systems were limited to thirty weak classifiers in order to be able to examine test images in real time. Another reason the fuzzy philosophy was not as successful is that it works with smaller decisions per weak classifier. The product of the decision with the weight of the classifier and the proximity factor to θ is a much smaller number than just the product of the weight of the WC and the decision. The point is that a fuzzy approach would more

slowly be able to change decisions made in the training procedure, and hence need more than thirty weak classifiers to be effective.

Figures 4.11 and 4.12 are examples from the testing set. They are evaluated using the best available strong classifier produced in this work.

Figure 4.11 – test case

Figure 4.12 – test case 2

4.9.3 Real Time Performance

Each picture from the testing set was given as input to the program that was implemented, and the time it takes to give the output is measured. Measurements are done by the software itself. Pictures were divided into groups by their size. The reported times in each group were exactly the same, and they were recorded up to two decimal

places of precision. For example, for images of size 170x227 pixels, the reported times were always: 0.18 sec. The standard deviation of the reported time was 0, for each group, therefore a test sample of any size is enough, according to basic statistics (size of confidence interval is proportional to standard deviation). The zero standard deviation is not a surprise, because each picture is going through the same set of 30 WCs on every sub window that it tests, independently of picture content. Measurements are made and reported only for one selected set of parameters. These measurements are represented in Table 4.9. They were measured on an AMD 64 3200 series machine.

Size(pixels)	time (sec)
150 x 120	0.05
170 x 227	0.18
320 x 240	0.49
500 x 252	1.93

Table 4.9 – Speed of the final strong classifier

The speed at which images are processed for positives in the detection stage is the same for the fuzzy and discrete versions of the program. Images as small as 150 x 120 pixels are processed in 0.05 seconds. Images of size 170 x 227 pixels are processed in 0.18 seconds. Standard size images similar to those used in Viola [39] and other papers of size 320 x 240 are processed in 0.49 seconds. We consider this to be real time. It takes more and more time to process larger pictures. For example, it takes 1.93 seconds to process a picture of size 500 x 253. The size of the picture directly impacts the time it takes to process it. This is logical since larger images contain more sub windows that must be searched. In fact, the running time is proportional to the number of features contained in a window of a given size. In our implementation, widths and heights of sub windows grow by 10%. The time complexity is therefore $O(AT \log(b/c))$, where b is the image width, c is the minimal example width, T is the number of WCs in the strong classifier and A is the area of the searched window, since there are $O(A)$ features of a given size, and $O(\log(b/c))$ incremental steps. When T is fixed, the complexity may also be expressed as $O(b^2 \log(b))$, where $b^2 \approx A$. We see that the complexity grows faster than quadratic time with respect to image width.

Chapter - 5 Conclusions and Future work

5.1 Conclusions

We began our research with the following question. Is there any 'magic' software package that can find any type of object in an image, reasonable accurately, and in real time, merely by replacing the positive and negative sets? If the answer was yes, there would not be so much research in this area today. Which existing software framework were we to therefore choose for the recognition of a new type of object that no one else did before (such as the recognition of a certain type of car)? We studied the literature thoroughly before making a decision. What did people who were using several methodologies conclude? The choice soon narrowed down to Support Vector Machines (SVMs) and AdaBoost. Everyone that used both of them claimed that AdaBoost was significantly faster, both for training and testing. Therefore, AdaBoost was chosen.

The training program can be considered as being composed of three components: an AdaBoost classifier, a Feature set, and a Training set. The AdaBoost classifier is a general framework, which can be safely claimed to be applicable to the recognition of any type of object efficiently, provided the object roughly appears with the same orientation and angle (e.g. straight upfront faces, or backs of horizontal cars, such as our positive). The question here is which framework is suitable for real time performance. In this direction, we contributed by combining a few existing ideas from different sources into a single new framework, not used/claimed before in that form. Thus the accuracy here is traded off somewhat for real time performance.

The feature sets are not as general. We show that the feature set for recognizing faces is completely different (practically disjoint) from the feature set for finding cars. Some existing articles added new features to help in recognizing objects which are different from faces, but we did not see anyone actually making the two sets disjoint. On the other hand, the same set of features could be used to recognize different objects, by

simply replacing one training set with the other. For instance, we believe that one could equally well recognize the back of another car such as the Toyota Camry 2004 by collecting the corresponding pictures for the training set and using the version of AdaBoost described in this thesis. In addition to defining a good feature set for Honda's, we have proposed a general elimination step in the program, by introducing a threshold for the quality of a feature, to reduce training time. This seems to be an original idea. It is an open area to further elaborate on the applicability of common feature sets, fully or partially, in recognition of some objects. One can always merge two sets into one, threshold them for triviality on a given training set, and then claim that the same feature set is applicable to recognition of two totally distinct objects. For instance, one can add the set of dominant edge orientations to the Viola's set for face detectors, and use them to train either faces or cars. When recognizing faces from appropriate training sets, redness features are eliminated, while many dominant edge orientation features may remain. When the same set is applied to recognize Honda's, all of Viola's features are basically removed first before real training, with the idea that we proposed (to the best of our knowledge, our system is the first real time AdaBoost based system that recognizes an object without Viola's features). But one cannot claim that this process can be continued to eventually include any type of objects, given the desired performance metrics. A new type of object may always exist that has its specific feature that works ideally for it, and needs to be added. An example is a circular object where a hypothetical roundness measure may be used to help identify it. We believe that one can further develop the idea presented here of introducing an automatic feature triviality test and link it to this discussion. Simply speaking, features from a large set are put through this test, and only some of them pass. Given two objects to recognize, one can define a measure of their similarity by looking at the number of common and different remaining features. We have not seen any such discussion along these lines in literature.

There exists no 'magic' answer that can easily explain the effort required to apply the techniques discussed here to the recognition of other objects such as faces. More research needs to be done to be able to quantitatively answer this point. The simple answer is that other types of cars can be recognized just be replacing the training sets in

this work. Our program is not restricted to only recognizing cars. We believe that dominant edge orientation features are powerful ones, and are applicable in many scenarios. For instance, fire extinguishers mainly have horizontal and vertical edges. Also, the redness feature is applicable in searching for fire extinguishers, given that fire extinguishers are mostly red. Therefore we have confidence that fire extinguishers (which are vertical in position and visually of the same shape) can be recognized with our software, because they appear quite simple to recognize, much simpler then backs of cars. Our system might also recognize fire extinguishers, even if dominant edge orientation features are removed (that is, based solely on the redness feature), because of their clear rectangular shape, and typical red color.

It is however 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 dataset required for the training and testing of the detection of the Honda Accord. The total number of usable pictures we were able to accumulate in this one month period was slightly less than 300. Consider now that 5000 faces were required to make Viola's version of AdaBoost work. Only 155 positives were required to make our system usable. It is impractical to think that a brute force application of AdaBoost or SVM could produce usable results with such a low number of training examples.

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's standard framework to any Vision problem.

5.2. Future Work

It is easy to see that there is room for improvement in the detection procedure. 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 detect positives faster than the non cascaded detector we have proposed here, but will still work with a limited number of training examples. 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.

Our initial goal was to design two AdaBoost based learning algorithms: with and without cascading, following the two variants in Viola's design. However, we only designed the non-cascaded algorithm where the best classifier is composed of a certain number of weak classifiers. The cascaded design was not implemented for several reasons. The most important reason is that the non-cascaded variant that we obtained was very efficient in terms of accuracy and runtime. This was achieved by a series of unexpected contributions made during the design. We believe that the design of cascaded variants of AdaBoost will pose even greater challenges and will require considerable additional time to perform. Therefore it was left for future study. The cascaded variant of Viola's design relies on a very large training set, and has no consideration of training time. Designing a cascaded version that incorporates a small run time, and a small training set would, in our view, be a topic worthy of researching. This is why we chose not to blindly implement the standard version of Viola's cascaded design.

The design of fuzzy weak classifiers and the corresponding fuzzy training procedure may be worth further investigation. We have perhaps selected a problem which was possible to solve efficiently with standard binary WCs. There are perhaps some more difficult problems, with finer boundaries between positive and negative examples, where fuzzy weak classifiers 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.

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 this thesis. 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] Y. Abramson, Y. Freund, Active learning for visual object recognition, 2004, Submitted for publication.

[2] Y. Abramson, B. Steux, Hardware-Friendly Pedestrian Detection and Impact Prediction, *IEEE Intelligent Vehicles Symposium,* Parma, Italy, June 14-17, pp. 590-595, 2004.

[3] J. Berens, G. D. Finlayson and G. Qiu, "Image indexing using compressed color histogram", IEEE Proceedings, *Vision, Image and Signal Processing,* vol. 147, no. 4, pp. 349 - 355, 2000.

[4] Bartlett, M.S., Littlewort, G., Fasel, I., Movellan, J.R., Real time face detection and expression recognition: Development and application to human-computer interaction, CVPR Workshop on Computer Vision and Pattern Recognition for Human-Computer Interaction, at IEEE CVPR, Madison, Wisconsin, June 17, 2003.

[5] J. Barreto, P. Menezes, J. Dias, Human-robot interaction based on Haar-like features and eigenfaces, *International Conference on Robotics and Automation,* New Orleans, pp. 1888-1893, 2004.

[6] T. Burghardt, B. Thomas, P. Barham, J. Calic, Automated Visual Recognition of Individual African Penguins, *Fifth International Penguin Conference,* Ushuaia, Tierra del Fuego, Argentina, September 2004.

[7] N. Bredeche, J. D. Zucker. WM-plic : combiner des classeurs issus de représentations différentes pour une tâche d'identification d'objets par un robot situé. Proceedings of La Conférence d'Apprentissage 2003 (Cap), pp.17-29. Laval, France, 2003.

[8] D. Cristinacce, T. Cootes, Facial feature detection using AdaBoost with shape constraints, Proceedings of *14th BMVA British Machine Vision Conference* Vol. 1, pp. 231-240, Norwich, UK, September, 2003.

[9] D. Claus and A.W. Fitzgibbon, Reliable fiducial detection in natural scenes, *European Conference on Computer Vision,* May 2004; LNCS 3024, pp. 469–480, 2004.

[10] N. Efford, *Digital Image Processing: a practical introduction using Java,* Addison Wesley, 2000.

[11] Y. Freund, R.E. Schapire, A decision-theoretic generalization on on-line learning and an application to boosting, Proc. 2nd European Conference on Computational Learning Theory (Eurocolt95), Barcelona, Spain, 23-37, 1995; *Journal of Computer and System Sciences*, 55(1): pp. 119–139, August 1997.

[12] Y. Freund, R.E. Schapire, A Short Introduction to Boosting, *Journal of Japanese Society for Artificial Intelligence,* 14(5): pp. 771-780, September, 1999.

[13] B.Fröba, S. Stecher, C.Küblbeck, Boosting a Haar-Like Feature Set for Face Verification, Lecture Notes in Computer Science, pp. 617 – 624, Springer-Verlag, 2003.

[14] N. R. Howe, A closer look at boosted image retrieval, *International Conference on Image and Video Retrieval*, pp. 61-70, July 2003.

[15] M. Jones and P. Viola, 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.

[16] M. Kolsch and M. Turk, Robust hand detection, Proc. *IEEE Int. Conf. on Automatic Face and Gesture Recognition,* pp. 614-619, May 2004.

[17] M. Kolsch and M. Turk, Analysis of rotational robustness of hand detection with a Viola-Jones detector, *International Conference on Pattern Recognition,* Cambridge, U.K., pp. 107-110, August 2004.

[18] R. Lienhart and J. Maydt, An extended set of Haar-Like Features for Rapid Object Detection, Proc. *IEEE Int. Conf. Image Processing,* vol. 1, pp. 900-903, 2002.

[19] D.D. Le, S. Satoh, Feature selection by AdaBoost for SVM-based face detection, *Information Technology Letters, The Third Forum on Information Technology (FIT2004),* 2004.

[20] D. Le and S. Satoh, Fusion of Local and Global Features for Efficient Object Detection, IS & T/SPIE Symposium on Electronic Imaging, 2005.

[21] K. Levi, Y. Weiss, Learning Object Detection from a Small Number of Examples: the Importance of Good Features, *International Conference on Computer Vision and Patern Recognition (CVPR)*, Vol. 2, pp. 53-60, 2004.

[22] X. Li, L. Wang, E. Sung, Improving AdaBoost for classification on small training sample sets with active learning, *Sixth Asian Conference on Computer Vision (ACCV)*, Korea, 2004.

[23] H. Luo, J. Yen, D. Tretter, An Efficient Automatic Redeye Detection and Correction Algorithm, *17th IEEE International Conference on Pattern Recognition, (ICPR'04)*, Cambridge UK *Volume 2,* August 23 - 26, pp. 883-886, 2004.

[24] Stan Z. Li and Z. Zhang, FloatBoost learning and statistical face detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 26, 9, 1112- 1123, Sept. 2004.

[25] M.A. Maloof, P. Langley, T.O. Binford, R. Nevatia, S. Sage, Improved rooftop detection in aerial images with machine learning, Machine Learning 53: pp. 157-191, 2003.

[26] B. McCane, K. Novins, On training cascade face detectors, *Image and Vision Computing,* Palmerston North, New Zealand, pp. 239-244, 2003.

[27] E. Osuna, Applying SVMs to face detection, IEEE Intelligent Systems, 23-26, July/August, 1998.

[28] O. Pujol, M. Rosales, P. Radeva, E. Nofreiras-Fernandez, Intravascular ultrasound images vessel characterization using AdaBoost, *Functional Imaging and Modeling of the Heart, Second International Workshop, FIMH 2003,* Lyon, France, pp. 242-251, June 5-6, 2003.

[29] V.S. Petrovic and T.F. Cootes, Analysis of Features for Rigid Structure Vehicle Type Recognition, Proc. *British Machine Vision Conference,* Vol. 2, pp. 587-596, 2004.

[30] J. Peng, B. Yu, D. Wang, Images Similarity Detection Based on Directional Gradient Angular Histogram, *16th Int. Conference on Pattern Recognition (ICPR'02)*, Quebec City, QC, Canada, Vol. 1, pp. 147- 150, August 11 - 15, 2002.

[31] H. Schneiderman, T. Kanade, Object detection using the statistics of parts, *Int. J. Computer Vision,* 56, 3, pp. 151-177, 2004.

[32] P. Silapachote, D.R. Karuppiah, A.R. Hanson, Feature selection using AdaBoost for face expression recognition, The 4th *IASTED International Conference on Visualization, Imaging, and Image Processing, VIIP 2004,* Marbella, Spain, pp. 452-273, September 2004.

[33] K. Sung, T. Poggio, Example based learning for view-based human face detection, *IEEE Trans. Pattern Anal. Mach. Intelligence, 20,* pp. *39-51, 1998.*

[34] R. Schapire, Y. Singer, Improved Boosting Algorithms Using Confidence-rated Predictions, Machine Learning, 37(3): pp. 297--336, December 1999.

[35] J. Thureson, S. Carlsson, Appearance Based Qualitative Image Description for Object Class Recognition, *European Conference on Computer Vision ECCV,* Prague 11-15 May 2004, LNCS 3022, pp. 518–529, 2004.

[36] J. Thureson, S. Carlsson, Finding Object Categories in Cluttered Images Using Minimal Shape Prototypes, *13th Scandinavian Conference on Image Analysis SCIA,* Goteborg, Sweden, pp. 1122-1129, 2003.

[37] A. Treptow, A. Masselli, A. Zell, Real-time object tracking for soccer-robots without color information, Proceedings of the *European Conference on Mobile Robotics ECMR,* 2003.

[38] K. Tieu, P. Viola, Boosting image retrieval, *Int. J. Computer Vision,* 56, ½, pp. 17-36, 2004.

[39] P. Viola, M. Jones, Robust real-time face detection, *Int. J. Computer Vision,* 57, 2, pp. 137-154, 2004.

[40] P. Viola and M. Jones, Fast and robust classification using asymmetric AdaBoost and a detector cascade, *Neural Information Processing Systems,* 14, 2002.

[41] P. Viola, M. Jones and Daniel Snow, Detecting Pedestrians Using Patterns of Motion and Appearance, Proceedings of the ninth *IEEE International Conference on Computer Vision ICCV,* Vol. 2, pp. 734 – 741, Oct. 2003.

[42] J. Wu, J. Regh, and M. Mullin, Learning a Rare Event Detection Cascade by Direct Feature Selection, Proc. *Advances in Neural Information Processing Systems 16 (NIPS*2003)*, MIT Press, 2004.

[43] D. Zhang, S. Z. Li, and D. Gatica-Perez, Real-Time Face Detection Using Boosting Learning in Hierarchical Feature Spaces, in Proc. *Int. Conf. on Pattern Recognition (ICPR)*, Cambridge, pp. 411 – 414, Aug. 2004.

[44] Z.Q. Zhang, L. Zhu, S.Z. Li, H.J. Zhang, Real-Time Multi-view Face Detection with FloatBoost, In Proceedings of *the 5th International Conference on Automatic Face and Gesture Recognition.* Washington, DC, USA., pp. 149-154, May, 2002.