2008

Detecting Curved Objects Against Cluttered Backgrounds

Jan Prokaj
University of Central Florida

DETECTING CURVED OBJECTS AGAINST CLUTTERED BACKGROUNDS

by

JAN PROKAJ
B.S. University of Central Florida, 2006

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2008

Major Professor: Niels da Vitoria Lobo

ABSTRACT

Detecting curved objects against cluttered backgrounds is a hard problem in computer vision. We present new low-level and mid-level features to function in these environments. The low-level features are fast to compute, because they employ an integral image approach, which makes them especially useful in real-time applications. The mid-level features are built from low-level features, and are optimized for curved object detection.

The usefulness of these features is tested by designing an object detection algorithm using these features. Object detection is accomplished by transforming the mid-level features into weak classifiers, which then produce a strong classifier using AdaBoost. The resulting strong classifier is then tested on the problem of detecting heads with shoulders.

On a database of over 500 images of people, cropped to contain head and shoulders, and with a diverse set of backgrounds, the detection rate is 90% while the false positive rate on a database of 500 negative images is less than 2%.

ACKNOWLEDGMENTS

This project would not have been possible without my research advisor, Dr. Niels Lobo, who has guided me through this difficult field. I would also like to thank the Computer Vision Lab at University of Central Florida for funding this project.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Object detection is a fundamental problem in Computer Vision. Given an image or a frame of a video sequence, a computer needs to identify what objects are in it, and their locations. As we have learned over the last several decades of work, this task is very difficult to do for a machine. Yet, humans can do this task very easily, and very fast, recognizing hundreds of objects every day. What makes this problem difficult?

Large intra-class variation. Detecting one instance of an object class is easy, but useless for the real world. Take cars for example. In an automotive vision system that is designed to signal the activation of brakes in dangerous situations, being able to detect a blue 2008 Ford Focus will not help when a silver 2006 Toyota Camry appears ahead. Therefore, the intra-class variation is a significant concern for any object detection algorithm. The variation in appearance can be large (Volkswagen New Beetle vs. Hummer). In the car industry, there are hundreds of car manufacturers, each producing several models every year, and each available in many colors. However, there is something in common between all these cars. Identifying this "something," a pattern, and quantifying it is the essence of object detection.

Viewpoint changes. Most objects look very different when they are rotated, or scaled by a large factor. They look different enough to be considered a different object. However, considering each view and scale as a separate object is very inefficient. Therefore, a robust object detection algorithm must keep this in mind.

Occlusion. An object may not be fully visible, but if it provides enough clues that a human can identify it, an object detection algorithm needs to identify it as well. In general,

this means that the algorithm can not have only one way of solving the task. When one attempt fails, it needs to be able to recover from it, and attempt to solve the problem another way. In object detection, this translates to using multiple object properties, or features, that can be independently calculated on different parts of the object.

Illumination changes. Color, brightness, and shading of an object all change under different illuminations. One way a good object detection algorithm minimizes the effect of illumination changes is by using gradient information rather than the raw intensity or color. This achieves invariance with respect to linear changes in illumination. Nonlinear changes in illumination, however, are more difficult to solve.

Dynamic object structure. Articulate objects are objects that have a dynamic structure, as opposed to rigid objects. This includes hands, humans, and animals. These objects look different even if the above variables are constant. For example, a hand with a closed fist looks different when the same hand is open under the same viewpoint and illumination. Detecting articulate objects requires much more effort than detecting rigid objects like cars.

Sensor noise. The last, but not least, problem is noise, and the quality of input to the algorithm. Noisy data, or low image contrast, can significantly affect calculations. Image pre-processing, such as smoothing, can help alleviate this problem. Still, an object detection algorithm should not impose fixed thresholds in the calculations, which cause chaotic performance. Rather, it should gracefully handle the failure of any one part of the calculation.

Given all these issues, it is clear that object detection is challenging. Solving one problem, such as intra-class variation, usually means poor performance on the other problems. However, each step brings us closer to the ultimate goal, which is what matters the most.

In this work, a step is taken in solving the problem of curved object detection. New low-level and mid-level features are introduced, which do not suffer from background clutter problem. The low-level features are fast to compute, which makes them especially useful

in real-time applications. The mid-level features are built from low-level features, and are optimized for curved objects. The usefulness of these features is tested by designing an object detection algorithm. Object detection is accomplished by transforming the mid-level features into weak classifiers, paving the way for boosting a strong classifier. The resulting strong classifier is then tested on the problem of head and shoulders detection.

In the next section, previous work in feature detection and object detection is discussed. In Section 1.1.3 the AdaBoost algorithm is reviewed, as it plays an important role in the design of the object detection algorithm used in this work. The next chapter describes in detail the contributions made by this thesis: low-level features, codebook of these features, and mid-level features. It also describes an object detection algorithm using these features. The following chapter presents the experimental setup for testing the performance of the features and the results achieved. Finally, this thesis is closed by discussion of the results, future work, and conclusions.

1.1 Previous Work

Given the fundamental nature of the object detection problem, it is not surprising to see an extensive work on all aspects of the subject. An important aspect of many object detection algorithms is image features. There are many approaches to the feature detection, and feature descriptor problem. The most important of these are reviewed in the next subsection. Subsequently, previous work in object detection is discussed, with one subsection dedicated to AdaBoost, which is used as an object detection algorithm in this thesis.

1.1.1 Image Features

Image features play an important role in many areas of computer vision, including correspondence problem, object tracking, and object detection. This importance is evident in the number of works on this issue in the literature. Since the practical application of this

thesis is in object detection, the focus is on works with features used in object detection. But first, what are image features, and what is their role in object detection?

A feature is a quantity, usually in high-dimensional space, that captures some characteristic of an object. These characteristics can be explicitly defined, such as color or brightness gradient, but usually they are implicitly found by a pattern recognition technique. The types of features that can capture this information include wavelet coefficients, histograms of oriented gradients, and many others. The goal in object detection is to find a pattern of feature instances in a training set that reliably represent the object. Then the object is detected in an unseen image when this pattern of feature instances appears.

The design of an image feature is divided into a feature detection algorithm, and a feature descriptor algorithm. A feature detection algorithm outputs a list of interest points in an image, then a feature descriptor algorithm takes each one, and calculates the feature descriptor from a small image patch around it. Using a feature detector is necessary, because calculating feature descriptors for every patch in the image would not only be computationally inefficient, but would give a lot of useless information. The best features are ones that occur often and consistently in the same place on the object.

The first significant feature detection algorithm is the Harris corner detector [HS88]. For each pixel, it calculates a second moment matrix, which measures the gradient distribution in a local neighborhood. The "cornerness" is then defined as the determinant minus the trace squared. Local peaks in the "cornerness" indicate the locations of interest points. Since the introduction of this detector, there have been many variants proposed, including Harris-Affine [MS02], and Harris-Laplace [MS01], which are scale-invariant.

Another feature detection algorithm was introduced by Lowe [Low04]. Here the interest points are local 3D extrema in the difference of Gaussian pyramid. The extrema determine the localization and the scale of the interest points. The difference of Gaussian

pyramid is obtained by subtracting two adjacent Gaussian blurred images, and it is a very efficient approximation of the Laplacian pyramid.

There have also been many feature descriptors developed [MS05]. The first, and the simplest, descriptor used is a vector of raw pixel intensities in a local image patch. Similarity between such descriptors is measured using normalized cross-correlation. This descriptor, however, is not invariant to rotation, and the descriptor similarity calculation is not efficient.

Descriptors based on histograms have been shown to be very successful. The most famous of these is the SIFT descriptor [Low04]. This descriptor stores a distribution of gradients in a local patch. The histogram quantizes gradient locations and orientations. Each histogram entry is weighted by gradient magnitude, but the descriptor is normalized to unit length to be invariant to linear illumination changes.

One problem with histogram based descriptors is their sensitivity to background for features points near the boundaries of an object. This does not matter for objects that are mostly planar, such as books, cars, or boxes, but it is a serious concern for articulate objects like hands, and humans, where the most important features occur on the boundary. This problem is illustrated in Figure 1.1, where SIFT descriptors are used to recognize an object. When the same object is moved to a different background, the number of matches decreases considerably. This shows that the feature descriptors include background information, which prevents a recognition of the object in varying backgrounds.

This problem was recognized by Mikolajczyk et al in [MZS03]. The solution proposed there divides the local patch around an interest point into two parts, foreground and a background. The division is along a chain of dominant edges. Then, a SIFT descriptor is calculated for each part separately. During matching, the foreground pair of descriptors is identified as the one with a minimum distance. One problem with this algorithm is that it is not computationally efficient.

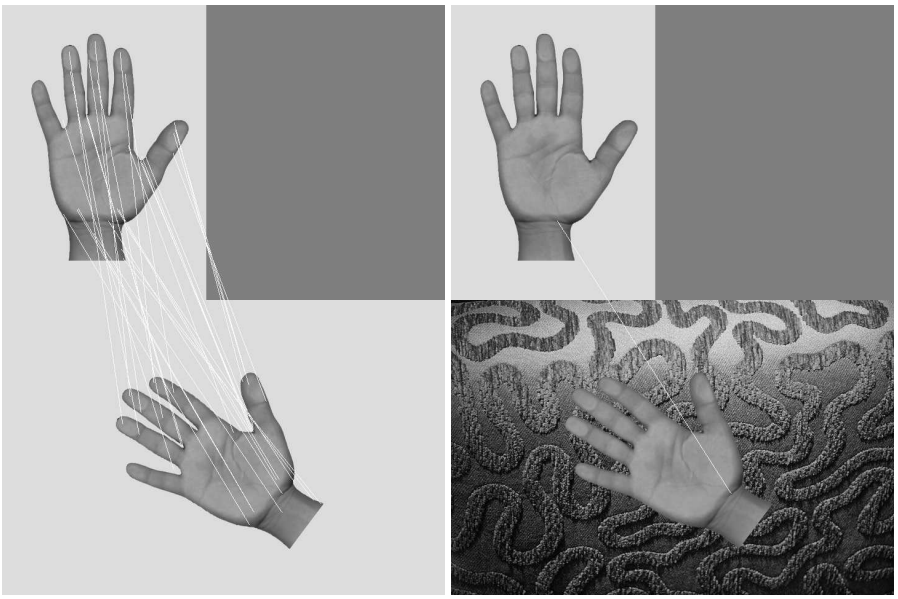


Figure 1.1: Object recognition using SIFT descriptors. When the background changes, the number of correct descriptor matches decreases significantly.

In this thesis, the descriptors for low-level features are designed with this problem in mind. The descriptors are histogram-based, however, the histograms do not capture the background information. Also, they are very efficient to compute. This will be discussed in Section 2.2.

1.1.2 Object Detection

The number of existing object detection algorithms is large, each one performing really well in some context. While the algorithms differ a lot in the details, many of them use image features at some point in the algorithm, and can be categorized into two approaches. They are either single detection window based [POP98, SK00, VJ01, DT05, SM07], or part-based [MPP01, FPZ03, LLS04, MSZ04, OPZ06]. Each approach has its advantages and disadvantages. A detection window based approaches generally offer better detection speed than part-based approaches, because there is no analysis of an object's structure. Their only concern is to find a pattern in a window. An object is then detected on a test image by scanning all possible locations in the image. Part-based approaches, however,

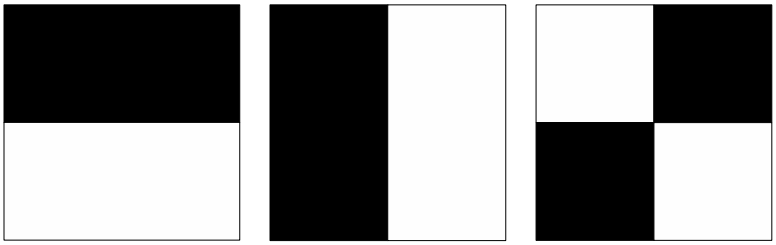


Figure 1.2: Selection of Haar wavelets used in object detection.

offer more model flexibility and thus capture more information in a compact representation. This work is motivated by works across this spectrum, and important algorithms from both approaches will be reviewed.

A general approach to object detection was presented in [POP98]. In this work, an object class is represented using an overcomplete set of Haar wavelets. Some of these are illustrated in Figure 1.2. The value of each wavelet is a difference in average intensity between different regions. The descriptor of a detection window contains these values for many instances of these wavelets (different sizes and locations in the window). The dimensionality of the descriptor is reduced to only contain the most promising wavelets. This is done by simple statistical analysis. A support vector machine (SVM) classifier is then used with the reduced descriptors to discriminate between the object class and background. The method shows good results on face detection and pedestrian detection.

A statistical approach to object detection was demonstrated in [SK00]. A probability distribution of appearance is estimated for an object class and background, and classification is made using the likelihood ratio test. Appearance is modeled as a product of class conditional probabilities of several visual attributes. Each visual attribute is a histogram of quantized wavelet coefficients at a particular location in the detection window. Wavelet coefficients capture information in space, frequency, and orientation. The histograms can be calculated using AdaBoost [FS97] to minimize classification error. This method is not very efficient, but results on face detection and car detection show that a large team of simple wavelet coefficients can capture complex patterns. This is confirmed in the next work.

The AdaBoost algorithm as an object detection tool was made famous by the landmark work of Viola and Jones [VJ01]. Here Haar wavelets are used as weak classifiers. AdaBoost selects the set of weak classifiers that minimizes the classification error. The weak classifiers are organized in a cascade to achieve real-time face detection. The speed of the algorithm is possible thanks to a new auxiliary data structure, the integral image, which allows fast computation of wavelet coefficients. Given the importance of AdaBoost, and its use in this thesis, this algorithm will be more covered in Section 1.1.3. Improvements to Viola and Jones' algorithm have been presented in [LM02, LW04], where additional types of features were added to the feature pool. These are rotated Haar-like features, and edge orientation histograms. These improvements illustrate the importance of good features in AdaBoost.

Recently, a lot of work has been done on human detection [DT05, ZYCA06, SM07]. Dalal and Triggs [DT05] showed that gradient orientation histograms are very useful features in this context. They trained an SVM classifier to discriminate between such histograms of human images and background images. This algorithm achieved state of the art performance, while being quite efficient. Interestingly, Zhu et al [ZYCA06] achieved the same performance at a significantly larger detection speed by using a larger variety of histograms, and implementing the algorithm in a cascaded AdaBoost framework. The currently best performance on human detection was achieved by [SM07], with a two stage AdaBoost classifier. In the first stage, AdaBoost is used to combine low-level features (oriented gradient responses) into mid-level features (shapelets). In the second stage, these mid-level features are combined into a strong classifier. The detection speed, however, does not match [ZYCA06].

All of these approaches show that wavelet coefficients, histograms, and AdaBoost are powerful ingredients in a detection window based object detection algorithm. When mixed

correctly, state of the art performance is achieved. Previous work in part-based approaches to object detection is reviewed next.

Part-based approaches try to model the object as a collection of parts and relationships between them. In [MPP01], a human was modeled as having 4 parts, which were manually determined. Each part was represented as a set of Haar wavelet coefficients. The resulting part detector was built using an SVM. Geometric constraints were applied to detected parts, before further processing. The parts were combined by storing the raw output of each SVM in a vector. After this, another SVM was trained on these vectors. This final classifier learned what combinations of part detections amount to an object detection. The method was tested on human detection, and showed that using a parts-based approach is advantageous in partially occluded scenes.

A more general algorithm was presented in [FPZ03], where an object is modeled as a constellation of parts. The number of parts is fixed, but the identity of parts is learned automatically by the algorithm. Each part has appearance, relative scale, and an occlusion state. Part appearance, part scale, and object shape are all modeled as Gaussian densities. The parameters of these densities for the object class are estimated using the expectation maximization algorithm. The parameters for background are estimated directly from a training set. The classification decision is made in a Bayesian manner. This algorithm is tested on a variety of objects, and achieves good performance.

An improvement over the previous algorithms was introduced in [LLS04]. Here, the identity as well as the number of parts is determined from training. The parts are automatically constructed using agglomerative clustering of image patches as in [AR02]. This set of parts is a codebook of local appearance. Each part in the codebook then records possible locations of the object centroid relative to it. This information is then used during detection to perform a generalized Hough transform and vote for the object centroid. The maxima in the voting space are found using Mean-Shift Mode Estimation. This method achieves state

of the art performance on car detection (side view). The disadvantage of this algorithm is that it requires a segmented training set.

A method similar to [LLS04] was presented in [OPZ06]. The difference is that the parts are not average image patches, but boundary fragments extracted from an edge map. Each fragment stores possible locations of the centroid as before. Because a single boundary fragment is not very discriminating, boundary fragments are combined to groups of two or three to form weak classifiers. The response of a weak classifier is a Chamfer distance of each boundary fragment to an image. A strong classifier is then found by AdaBoost. During detection, as before, the weak classifiers vote for the location of the centroid. The presence of an object is indicated by a maximum in the voting space. Very good results are achieved on the Caltech dataset without requiring a segmented training set.

It is clear that the AdaBoost algorithm is an invaluable tool in object detection. It has been successfully used with detection window based approaches, as well as parts-based approaches. Since this algorithm is also used in this thesis, it is explained in detail in the next section.

1.1.3 AdaBoost

The AdaBoost algorithm [FS97] is based on the idea of using a team of experts for making a decision. An expert does not have to be a genius, and be correct all the time. In fact, the algorithm only requires that an expert is right at least 51% of the time. For this reason, an expert is called a weak classifier. The purpose of AdaBoost is to find a team of weak classifiers and combine their decisions in such a way that the resulting team decision is correct every time. The team is then a strong classifier.

AdaBoost is a supervised approach, requiring a training set with positive examples and negative examples of input. Selecting the members of a team is done in an iterative fashion. The key idea is the reweighting of training examples. Initially, the weight of each training

example is set uniformly. The next member of the team is the weak classifier that has the lowest error on the weighted training set. As soon as a new member is selected, the training examples are reweighted such that the weights of correctly classified examples are decreased. Each member is also assigned a weight that corresponds to its error on the training set. This loop continues until meeting some convergence criterion. The algorithm is illustrated in Figure 1.3.

One of the most important properties of AdaBoost, which was proved by Freund and Shapire [FS97], is that if each weak classifier is slightly better than random, then the training error drops exponentially fast. They have also shown that the decision by the final strong classifier is identical to the Bayes optimal decision rule.

Viola and Jones [VJ01] found a very useful transformation of AdaBoost, the cascade. This small modification to the algorithm builds several strong classifiers in stages that together have the equivalent performance as a single strong classifier found by a non-cascaded AdaBoost. The strong classifiers are arranged in a pipeline, with each strong classifier deciding whether to let the input continue in the pipeline or not. Input that passes through every strong classifier is classified as positive. Conversely, any input that fails at any strong classifier is classified as negative, and is not considered further by the subsequent strong classifiers. This construction allows the classification process allocate resources more efficiently. Input that is clearly negative fails early in the pipeline and uses little computation cost, since the subsequent classifiers do not see it. On the other hand, input that is very difficult to classify uses as much resources as possible, passing through every classifier.

This method naturally requires that each strong classifier has a very high true positive rate, such as 99%. The false positive rate does not have to be very low, however. For example, if the goal is to have an overall true positive rate of 90% and a false positive rate

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 1.3: AdaBoost algorithm. This figure is courtesy of [VJ01].

of 0.10%, in a 10 stage cascade, each strong classifier needs to have a true positive rate of 99% ($0.99^{10} \approx 0.90$), but a false positive rate can be as high as 50% ($0.50^{10} \approx 0.001$).

2. METHODS

The general approach followed here is that of a typical object detection algorithm. In the training phase, a pattern of image features is found from a training set and in testing, image features are matched to this pattern. The details of this approach, however, differ from the object detection algorithms reviewed so far. The training phase has three stages. In the first stage, low-level features are computed from raw training images. These low-level features are designed to be stable and not suffer from problems caused by background clutter. In the second stage, mid-level features are built from the low-level features. The mid-level features are optimized for curved object detection and are discriminative enough to be used as weak classifiers. In the final stage, the mid-level features are combined into a strong classifier using AdaBoost.

Each stage of the algorithm is explained in the following sections. The low-level features, curves, are introduced first. In order to build the mid-level features, a codebook of curves needs to be constructed. This process is explained second. Subsequently, the mid-level features, triplets of curves, are introduced. The final stage of the algorithm, boosting of triplets as weak classifiers, closes this chapter.

2.1 Feature Detector

The low-level features used in this work are curves in multiple scales. These naturally describe an object's structure. An object can be thought of as being composed of many points, or lines, but neither of these is general enough to provide a compact representation. In addition, curves can be found consistently, because the center of a curve is a point with a

high gradient magnitude. This high-frequency content is stable across images. Also, since curves naturally partition the object's structure, they can have a high-level meaning and describe the object's parts.

In order to achieve high detection speed, the feature extraction stage must be fast. Since this is the first stage, it determines the algorithm's speed limit. A fast detection of curves is accomplished by creating a convolution filter, which responds to the presence of curves and is designed for a fast convolution using the integral image.

Integral image is a data structure that stores the sum of pixel values in the rectangle defined by the top left corner of an image and the current pixel. Sometimes, it is called "summed area table" from the field of computer graphics. It can be computed in one pass over an image. The most important property of the integral image is that it enables the calculation of a sum of image values in any rectangle in four additions. Therefore, if a convolution filter is composed of a few rectangles, this allows a rapid calculation of convolution.

Of course, the goal here is to detect curves, and thus curves need to be approximated by rectangles. By approximating a curve with rectangles, there is a tradeoff in accuracy for speed. A higher number of rectangles better approximates a curve's shape, but does not provide as much speedup. Another constraint in the design of the filter is that the sum of the filter values must be zero. Otherwise, the filter would respond even in areas with constant intensity. The filter must be symmetric as well, so that the response is the highest when the filter is over the center of the curve. The smallest possible filter size that satisfies all these constraints is shown in Figure 2.1. The white area has elements with value equal to 1, and the dark area has elements with value equal to -1.

Flipping this filter vertically, rotating by 180°, and flipping horizontally captures the response of curves of four different orientations. More orientations can be captured by rotating the image by 45° or by using a rotated filter in the manner of [LM02], where a rotated

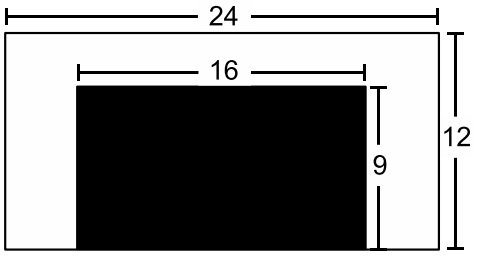


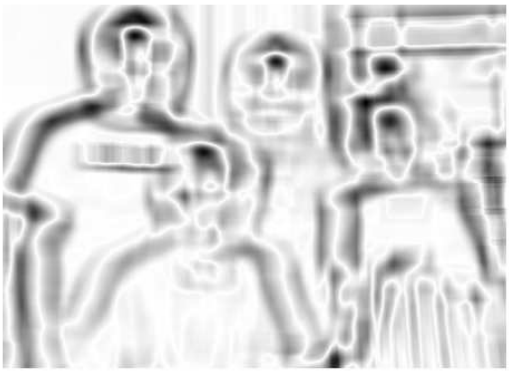
Figure 2.1: Curve filter.

integral image data structure is developed. This gives an additional four orientations, for a total of eight. Eight orientations allow us to find the most important curves of most objects. However, higher precision may be required in certain domains.

To get responses at different scales, the filter is resized accordingly. The only restriction in resizing is that the properties of the filter are preserved. This means that the filter size must be a multiple of 24x12. As a result of this restriction, this filter will not be able to find curves less than this size. However, this does not cause problems, as curves smaller than this are not well defined and are not likely to be found consistently across images. In the current implementation, the number of scales is as high as possible. For example, in a 320x240 image, 10 scales would be used.

Given an integral image, and a filter, the calculation of the filter response is straightforward. For each pixel in the image, except for border pixels where the response can not be accurately calculated, the filter is centered on it, and the areas of the big 24x12 rectangle and small 16x9 rectangle are calculated from the integral image. Then the area of the small rectangle is subtracted from the area of the big rectangle, and the result is stored as the response at this pixel. The calculation of the responses for different orientations can be speeded up by taking advantage of the fact that flipping the filter vertically or horizontally only changes the position of the small rectangle. Therefore, calculations of the responses for opposite orientations of the filter can share the result of the area of the big rectangle.

(a) Original



(b) Response of a filter oriented up

(c) Response of a filter oriented left

(d) Response of a filter oriented down

Figure 2.2: Curve filter responses.

The filter responses for an example image are shown in Figure 2.2. A high filter response is indicated by black color, whereas no filter response is indicated by white color. These responses are only from one scale. It is clear that curved areas show high filter response. For example, tops of heads have a high response from the filter oriented up. Areas with very little texture show no response at all.

Curve features can be identified as peaks in these responses. However, there is one complication. Areas in the image that have weak filter response, can still appear to be peaks. Therefore, response images from filters with opposite orientation are subtracted from each other, and the absolute value is calculated. The resulting image will have high values only for pixels that are oriented in two opposite directions. Pixels with weak response from both

(a) Subtracted filter response (b) (a) enhanced in green and overlapped with input

Figure 2.3: Subtraction of opposite vertical directions.

filters will have values close to zero in the new image. This is illustrated in Figure 2.3. Here residual filter responses disappeared and the vertical directions clearly stand out.

Local peaks in this image are taken as centers of curves. A peak is defined to be a pixel with a value that is greater than or equal to other values in a 3x3 neighborhood. The number of neighbors with equal value as the peak is limited to two. Also, the number of peaks in the image is limited by thresholding the peak values to be at least 45% of the maximum response in the image. This avoids including peaks that are not very strong. The orientation of a peak is assigned by comparing the values of the responses in the response images with opposite orientations before subtraction. A higher response in one direction than the other determines the peak's orientation.

Since peaks can have neighbors with equal values, it is possible to get multiple peaks responding to the same area in the image. This is undesirable, because an unnecessarily high number of features will slow down processing down the line. This problem is solved by clustering the peaks based on their location in the image. Peaks in each direction are clustered separately. Agglomerative clustering is used, with a stopping criterion of a minimum distance between two clusters. When the minimum distance becomes greater than $6 * (s + 1)$, where s is a 0-based scale number, the clustering stops. The effect of this step

(a) Before clustering (b) After clustering

Figure 2.4: Effect of feature point clustering (scale=2).

can be seen in Figure 2.4. It is clear that duplicate peaks are filtered out. At the end of this process, the number of curve features in a 320x240 image varies from 200 to 900.

2.2 Feature Descriptor

Now that the low-level features are identified, there needs to be a way to match them across images. This is accomplished by designing a scale-invariant feature descriptor. The descriptor is an 8-dimensional histogram, storing the number of occurrences of features (oriented curves) in the neighborhood of the feature. Each entry in the descriptor is weighted by a Gaussian centered on the feature. This means that points close to the center of the neighborhood are weighted more than points at the boundary of the neighborhood. Also, each entry is linearly interpolated into the neighboring bins to avoid histogram bin boundary effects. Finally, the descriptor is normalized to unit length.

To achieve scale invariance, the descriptors are calculated in the scale in which the feature was found, and the size of the neighborhood is kept proportional to the feature scale. At the finest scale, the neighborhood size is 3x3. At coarser scales, the size increases to 6x6, 9x9, and so on. The reason for the small neighborhood size is to avoid noisy descriptors. As the size of the neighborhood increases, the descriptor is affected by features farther away,

Figure 2.5: Head as a set of curves.

which are less likely to co-occur with the center feature. Keeping the neighborhood size small ensures that features that have matching descriptors really occur in the same image areas.

2.3 Codebook

Curves and their descriptors on their own do not have enough discriminative power for object detection. A group of curves, is much more useful. For example, a certain arrangement of curves naturally describes the omega shape of a head. This is illustrated in Figure 2.5. However, in order to define a group of curves, each curve needs to have a label. A group can then be defined by the labels of curves in the group. For example, group 3-4-7 indicates a group of three curves: curve-3, curve-4 and curve-7. The idea is then to look at the pattern of groups of curves, rather than single curves.

In our case, assigning a curve a label means assigning the curve's descriptor a label. This is achieved by generating a set of possible descriptors, each with a unique label attached to it. This set is often called a "codebook of local appearance," or a "visual word alphabet" in the literature. A curve is then assigned a label of its nearest neighbor in the codebook.

An optimal number of labels in the codebook is critical. Too many labels will make it difficult to get consistent matches between images. For example, the curve on the top of the head in one image may be labeled 12, and the same curve on a slightly different image may be labeled 15. The labels must be consistent in the image area they describe. A very low number of labels, however, will not allow the algorithm to distinguish between different curves. Having less than eight labels will not even allow different orientations of curves to be distinguished. An optimal number of labels lies somewhere between the two extremes.

Determining the optimal number of labels is the same problem as determining an optimal number of clusters in clustering, where each data sample is a curve's descriptor. This is a well studied problem. One solution to this problem is offered by Tibshirani et al in [TWH01]. The authors use a heuristic that the optimal number of clusters occurs when the decrease in within-cluster dispersion flattens out as the number of clusters is increased. On a plot of the within-cluster dispersion versus the number of clusters, this is the location of the "elbow." The within-cluster dispersion for k clusters, W_k, is defined as

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} D_r \ , \tag{2.1}$$

where

$$D_r = \sum_{i,i' \in C_r} d_{ii'} \ , \tag{2.2}$$

C_r is a cluster r, n_r is the size of C_r, and $d_{ii'}$ is the distance between two samples in a cluster.

The approach is to standardize the graph of $\log W_k$ by comparing it with its expectation under a null reference distribution of the data. The optimal number of clusters is then the the value of k for which W_k falls the farthest below this reference curve. This value of k maximizes the gap statistic, which is

$$Gap_n(k) = E_n^*\{\log W_k\} - \log W_k \ , \tag{2.3}$$

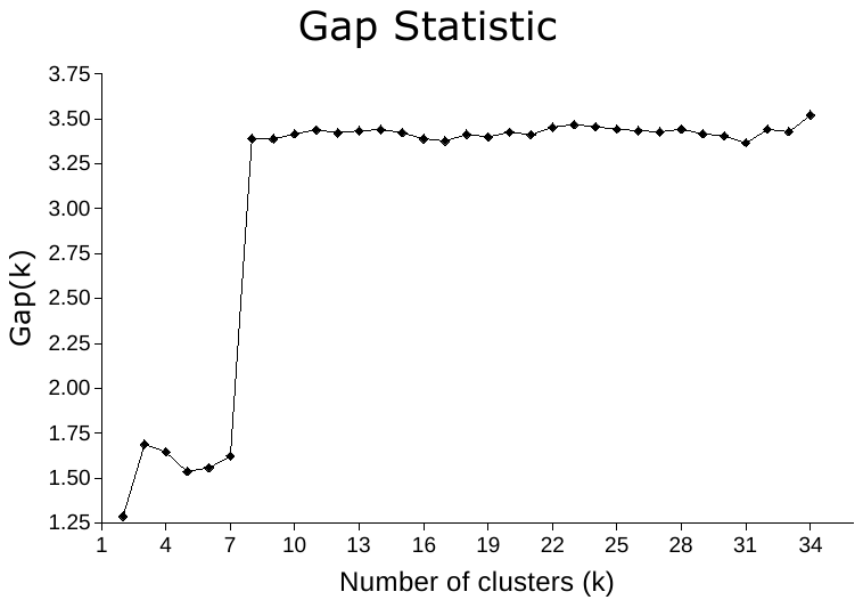


Figure 2.6: The gap statistic.

where E_n^* denotes expectation under a sample of size n from the reference distribution.

However, simply looking for the maximum value of $Gap_n(k)$ will only work for well separated data. More complex data distributions can have several local maxima and the optimal maximum can only be determined by looking at the plot of $Gap_n(k)$. This is the approach followed in this work.

In order to determine the optimal number of labels, 29279 descriptors, generated from 200 images (positive and negative examples), were clustered. The clustering algorithm used for each k was k-means, where k ranged from 2 to 35. The stopping criterion is the absolute change in cluster centers. When this value drops below $0.001 * k * d$, where d is the dimension of each data point, clustering stops. In the calculation of the gap statistic, the expectation was estimated by an average of 3 reference data sets. The samples in each reference dataset were generated from a uniform distribution over a box aligned with the principal components of the data. The resulting gap statistic is plotted in Figure 2.6.

It is clear from the plot that the optimal number of labels is between 8 and 33, because that is the region of the plot with a maximum gap value. It is interesting to note that there is

Figure 2.7: Image patches corresponding to descriptors in several clusters.

a local maximum at $k = 3$, but that one is clearly not the optimal global maximum. Therefore, using a simple search for the first local maximum would yield this suboptimal value. The gap begins to rise again at $k = 34$, but that number of labels is too high, resulting in a very fragmented descriptor space, which is not good for pattern recognition. The beginning of the maximum at $k = 8$ is not surprising, because this corresponds to the eight primary directions of our curves. However, only using eight would assign descriptors containing multiple directions the same label as descriptors with one primary direction. Therefore, the optimal k for this application is somewhere in between. In this work, $k = 18$ was used. With this value of k, 97% of descriptors fall into one of the eight labels corresponding to primary curve directions.

This number can be verified by looking at the set of image patches that generated the descriptors belonging in each cluster. A selection of image patches in each cluster can be seen in Figure 2.7. It is clear that patches in a cluster have common appearance.

2.4 Triplets

In this work, groups of three curves, called triplets, are used as mid-level features. Once a codebook of feature descriptors is constructed, it is easy to define these. For example, one triplet can be defined as 3-4-7, which means it is a group of curve-3, curve-4, and curve-7. This simple definition, however, is still not discriminative enough to be used directly. A triplet $a-b-c$ occurs in a multitude of geometric configurations in images. Therefore, the definition of a triplet is augmented by several geometric properties. These are calculated from the triangle, where a vertex corresponds to the location of a curve. The properties are:

- Two angles defining the triangle orientation (α_1, α_2). One angle is not enough, due to a mirror image ambiguity.
- Internal angles of a triangle (β, γ).
- Normalized size of one side of a triangle.
- Normalized coordinates of each vertex of a triangle.

These are summarized in Figure 2.8. While this set of properties is redundant in describing the geometry of a triplet, it provides flexibility in calculating the geometric similarity of two triplets.

In order to achieve correct matching of geometric properties between triplets, for any triplet defined by $a-b-c$, $a < b < c$. This constraint ensures that, for example, the internal angle β in one triplet is compared to the corresponding internal angle in another triplet. One implication of this constraint is that a triplet must be defined by 3 different curves – triplet $3-5-5$ is not allowed, for instance.

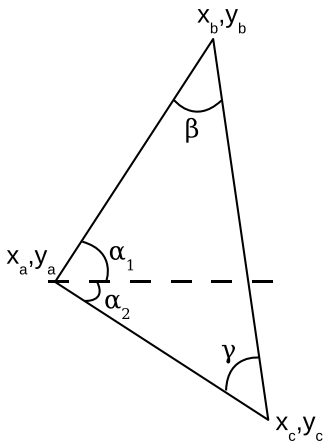


Figure 2.8: Geometric properties of a triangle formed by a triplet.

2.5 Boosting

Triplets are easily transformed into weak classifiers. The feature response of the weak classifier is a minimum distance of a triplet to an image. This distance measures the geometric similarity between the triplet of the weak classifier to triplets in an image with the same label. If there is no such triplet in an image, the distance returned is infinity. Formally, geometric similarity between two triplets, $\tau_{a1-b1-c1}$ and $\tau_{a2-b2-c2}$, is defined as

$$d(\tau_{a1-b1-c1}, \tau_{a2-b2-c2}) = \begin{cases} \|\mathbf{x}_{a1} - \mathbf{x}_{a2}\| + \|\mathbf{x}_{b1} - \mathbf{x}_{b2}\| + \|\mathbf{x}_{c1} - \mathbf{x}_{c2}\| & a1 = a2, b1 = b2, c1 = c2 \\ \infty & \text{otherwise} \end{cases} \tag{2.4}$$

where $\mathbf{x}$ denotes the location of a curve in normalized coordinates ($[0,1]$). Using curve locations in similarity calculations implies that classification is done in a detection window. This is intentional, because the geometric location of a triplet is significant. It helps to avoid false matches between triplets. For efficiency reasons, some geometric properties of a pair of triplets are checked before even calculating the formal geometric similarity. If any of these properties are not satisfied, the distance returned is automatically infinity. These properties are:

- Scale difference between corresponding curves is ≤ 1.
- α_1 difference $\leq \frac{\pi}{8}$.
- α_2 difference $\leq \frac{\pi}{8}$.
- β difference $\leq \frac{\pi}{16}$.
- γ difference $\leq \frac{\pi}{16}$.
- Normalized size difference ≤ 0.10.

Given the triplet distance to an image as a feature response, a weak classifier can be trained to discriminate between two classes of examples, with an accuracy of 51-75%. In training the weak classifier, examples where the feature response is infinity are ignored. This weak classifier is of the form:

$$h_t(x) = \begin{cases} 1 & \text{if } p_t f_t(x) < p_t \theta_t \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where p_t indicates the polarity of the inequality sign, θ_t is the classification threshold of the classifier, and $f_t(x)$ is the feature response. Here x is a detection window in an image.

Using the AdaBoost algorithm, a strong classifier can be built from a set of weak classifiers. After T iterations of the algorithm, the strong classifier is of the form:

$$H(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq c \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where α_t is the selected weight for classifier $h_t(x)$, and c is a threshold.

The AdaBoost algorithm works best with a very large feature pool. Ideally, all possible triplets are considered as weak classifiers. However, since the number of possible triplets is theoretically infinite, and practically very large, not all triplets can be included in the

feature pool. Therefore, a careful triplet selection process is employed. Initially, the set of available triplets is generated dynamically from the training set. For each image in the training set, curves are extracted, and all valid combinations of 3 curves in the image are put in the feature pool. Valid combinations of curves must have 3 different labels, and the scale difference between the curves must be ≤ 1. In order to consider as many training images as possible and achieve a diverse set of triplets, a triplet is included in the feature pool only if its geometric similarity to other triplets with the same label already in the feature pool exceeds a threshold.

The resulting feature pool is still relatively large and computationally intensive for AdaBoost. It also contains many triplets that have a response on only one or two images. In general, AdaBoost can work with these features, but in practice, these present serious overfitting problems. Therefore, the pool is filtered as follows. For each triplet, its responses on P positive and N negative examples in the training set are calculated and sorted in ascending order. Then, a strength score of a triplet τ, $S(\tau)$, is calculated using

$$\begin{aligned} S(\tau) &= \sum_{j=0}^{P+N-1} s(j) \\ s(j) &= \begin{cases} p*(P-j) & j < P \\ -p*(j-P+1) & \text{otherwise} \end{cases} \end{aligned}$$

where p is 1 for positive examples and -1 for negative examples. The lower the score, the weaker a triplet is. The score can be negative. Those triplets that have a score less than 0.25 times the maximum possible score are removed from the feature pool. Similar feature pool optimization has been done in [LKW06]. This final feature pool is used in AdaBoost to train a strong classifier.

Once AdaBoost completes, the resulting strong classifier can be applied on any detection window. First, the low-level features are extracted. Then, each triplet in the weak

classifier is matched to possible triplets in the window. The response of a triplet is the minimum geometric distance to matched triplets. Based on the weak classifier threshold and polarity, the response is classified as positive or negative. The strong classifier makes a decision based on the sum of the responses of all triplets.

In a 320x240 image, thousands of detection windows are considered. As a result, multiple overlapping windows can be classified as a positive around the object of interest. These detections are cleaned up by sorting the windows by the strength of the classifier response, and calculating the overlaps of each window. If the area of the overlap is greater than 50% of the detection window, the overlapping window is removed.

3. RESULTS

The algorithm was evaluated on a head and shoulders detection task. The positive training set consisted of 527 images of people from the front view, cropped to contain the head and shoulders, and centered in the image. The negative training set consisted of 7,335 images of anything but people. Triplets in the feature pool were generated from a random subset of 14 images from the positive training set. This feature pool contained 172,000 triplets. After feature pool optimization using the strength threshold, the feature pool was reduced to 25,886 triplets. This optimization was based on a random set of 57 positive and 57 negative images.

Only one AdaBoost cascade stage was trained for the purposes of algorithm evaluation. The training performance was monitored on a small validation set of 57 positive and 57 negative images. Training was stopped after the strong classifier achieved at least 90% true detection rate and less than 4% false positive rate on this set. As a result, the final strong classifier contained 85 weak classifiers (triplets). The behavior of the first few triplets chosen by AdaBoost is shown in Figure 3.5.

The testing set consisted of 500 positive images and 500 negative images. In a typical 183x145 test image, there were about 330 detection windows tested. A positive image was classified correctly if at least one detection window over the head was positive. Similarly, a negative image was classified correctly if no detection window was positive. The resulting true positive detection rate was 90% while the false positive rate was 2%. Example detections are illustrated in Figure 3.2. Examples of detection on images with multiple people are shown in Figures 3.3 and 3.4. False positives and false negatives can be seen

Figure 3.1: Examples of correct detection from an opposite viewpoint (not trained on).

in Figure 3.6. Post-processing of overlapping detection windows was turned on except for detection in images with multiple people. This was because in this task the post-processing algorithm did not work well, and removed the correct detection windows. The number of scales tested was reduced as well in this task.

In order to verify that the object detection algorithm is using object's curves to make a decision, the algorithm was also run on images with people from the back view. This viewpoint was not present at all in the training set. However, the set of curves from the back-view is roughly the same as the set of curves in the front view. Therefore, the algorithm should be able to work here as well. These results are shown in Figure 3.1.

Figure 3.2: Examples of correct detection in a testing set.

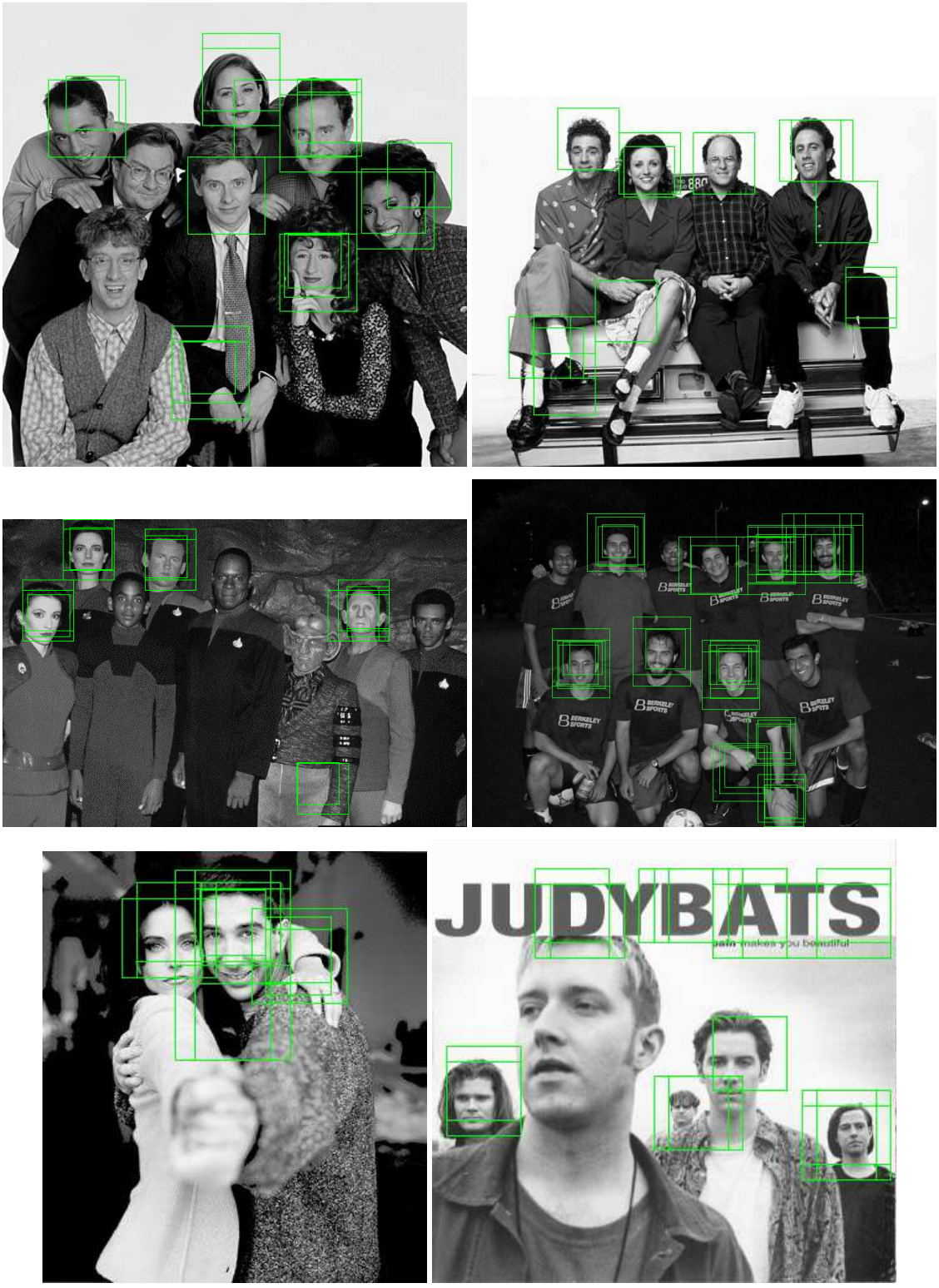


Figure 3.3: Examples of detection on images with multiple people.

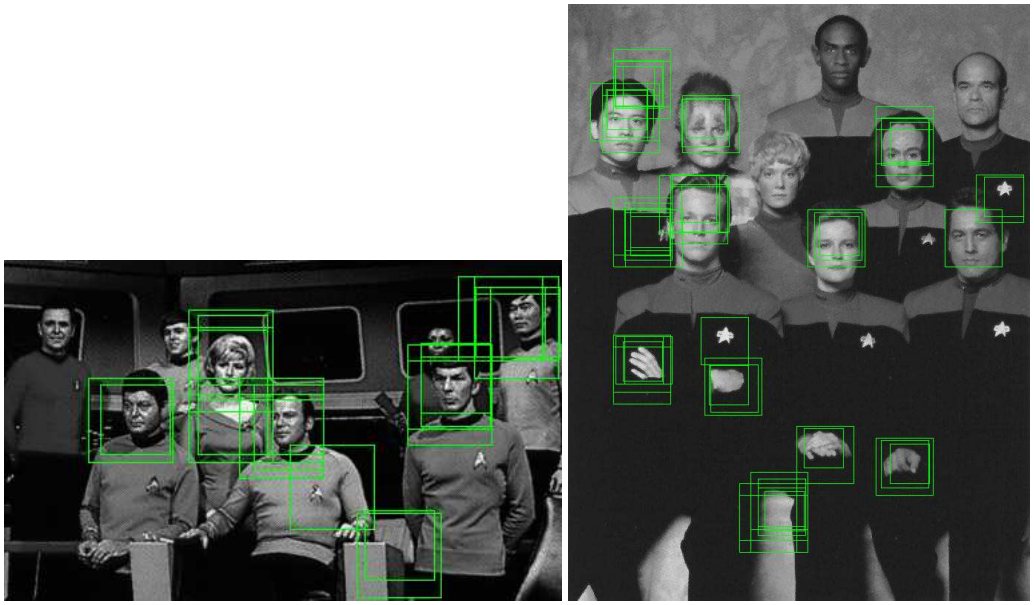


Figure 3.4: Examples of detection on images with multiple people.

Figure 3.5: The first few triplets selected by AdaBoost.

Figure 3.6: Examples of false negatives and false positives.

4. DISCUSSION

It is encouraging that significant performance was achieved with less than 100 triplets. This is compared to hundreds of Haar wavelets necessary to accurately detect a face. It is clear that features which do not heavily depend on intensity differences are useful.

It is interesting to see that the first triplets selected by AdaBoost correspond to the natural curves on the boundary of a head. The first two triplets capture the top curve, which stretches from the the left ear to the right ear. The third triplet makes the connection between a head curve and a shoulder curve. The fourth triplet captures a curve on the right side of a head.

The true positive rate was 90% and the false positive rate was 2%. This result is very encouraging, considering that the images tested had a diverse set of backgrounds, and the algorithm is using only very simple features. It does not take into account local brightness variations at all. This is confirmed further by the detections on a viewpoint not present in training. Learning an object's shape rather an appearance gives the algorithm a little bit of viewpoint invariance.

It is clear from the correct detection examples, that the variation in appearance is huge. There are people with light/dark skin, with/without glasses, with/without hats, with long/short hair, with light/dark hair, even in a small variety of configurations (facing left/right). Haar wavelets, or other features that use brightness information directly are not able to capture this variation in a compact form. The features introduced in this algorithm can, because they look at object's structure rather than appearance.

The failures of this algorithm are often a result of contrast problems, where curves in the image are not clearly evident. Also, some of these images are significantly blurred, which results in a very weak structure that is not captured by the algorithm. Loosening the thresholds in the algorithm can help alleviate this problem, but it inevitably results in more false positives. Solving these problems is an opportunity for future work.

5. CONCLUSIONS

New low-level and mid-level features designed for curved object detection were presented. These features capture the object's structure rather than appearance and thus do not suffer from the background clutter problem. The low-level features are fast to compute, which makes them especially useful in real-time applications. The mid-level features are built from low-level features, and are optimized for curved object detection.

Additionally, an object detection algorithm using these features was designed to evaluate the features' usefulness. This was accomplished by transforming the mid-level features into weak classifiers. The results on head and shoulders detection show a promising direction for detecting curved objects against cluttered background, where the features on the object's boundary are important.

REFERENCES

[AR02] Shivani Agarwal and Dan Roth. Learning a sparse representation for object detection. In *Proceedings of the 7th European Conference on Computer Vision*, volume 4, pages 113–130, 2002.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[FPZ03] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2003.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988.

[LKW06] Fayin Li, Jana Košecká, and Harry Wechsler. Strangeness based feature selection for part based recognition. *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, page 22, 2006.

[LLS04] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *ECCV'04 Workshop on Statistical Learning in Computer Vision*, pages 17–32, 2004.

[LM02] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *International Conference on Image Processing*, volume 1, pages 900–903, 2002.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LW04] K. Levi and Y. Weiss. Learning object detection from a small number of examples: the importance of good features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 53–60, 2004.

[MPP01] Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio. Example-based object detection in images by components. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(4):349–361, 2001.

[MS01] Krystian Mikolajczyk and Cordelia Schmid. Indexing based on scale invariant interest points. In *International Conference on Computer Vision*, volume 1, page 525, 2001.

[MS02] Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. In *Proceedings of the European Conference on Computer Vision*, pages 128–142, 2002.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[MSZ04] K. Mikolajczyk, C. Schmid, and A. Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In *Proceedings of the European Conference on Computer Vision*, volume 1, pages 69–82, 2004.

[MZS03] K. Mikolajczyk, A. Zisserman, and C. Schmid. Shape recognition with edge-based features. In *Proceedings of the British Machine Vision Conference*, volume 2, pages 779–788, 2003.

[OPZ06] A. Opelt, A. Pinz, and A. Zisserman. A boundary-fragment-model for object detection. In *Proceedings of the European Conference on Computer Vision*, pages 575–588, 2006.

[POP98] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer Vision*, pages 555–562, 1998.

[SK00] Henry Schneiderman and Takeo Kanade. A statistical method for 3d object detection applied to faces and cars. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, page 1746, 2000.

[SM07] Payam Sabzmeydani and Greg Mori. Detecting pedestrians by learning shapelet features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, page 511, 2001.

[ZYCA06] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1491–1498, 2006.