Real Time Face detection System Using Adaboost and Haar-like Features

Jie Zhu
College of Electronics and Information Engineering
Nanjing University of Technology
Nanjing, China
zhu.jie.ergee@gmail.com

Zhiqian Chen
Department of Software Engineering
Peking University
Beijing, China
imczq@pku.edu.cn

***Abstract*—Face detection is widely used in interactive user interfaces and plays a very important role in the field of computer vision. In order to build a fully automated system that can analyze the information in face image, there is a need for robust and efficient face detection algorithms. One of the fastest and most successful approaches in this field is to use Haar-like features for facial appearance and learning these features by AdaBoost algorithm. The key advantage of a Haar-like feature over most other features is its calculation speed. Due to the use of integral images, a Haar-like feature of any size can be calculated in constant time, which greatly accelerates the detection speed, while AdaBoost algorithm is a good way to select a good set of weak learners to construct a strong classifier. In this paper, a real time face detection system using framework of Adaboost and Haar-like feature is developed. In the end, the experiments show high performance in both accuracy and speed of the developed system.**

Keywords-face detection; Adaboost; Haar-like feature; integral image; computer vision

I. INTRODUCTION

Face detection is widely used in interactive user interfaces, advertising industry, entertainment services, and video coding. It is also necessary first stage for all face recognition systems, etc. Face detection issue is about how to find, based on visual information, all the occurrences of faces regardless of who the person is. Face detection has been one of the most challenging problems in computer vision. So far there is no solution to achieve performance comparable to humans no matter in precision or speed, however, some best published results are achieved in papers [1] [2] [3] [4].

Face detection can be viewed as a two-class classification problem in which an image region is classified as either a "face" or a "non-face". There are over 170 approaches to face detection. All of them can be classified into four categories: Knowledge-based methods, Feature invariant methods, Template matching methods, and Appearance-based methods [5]. In the following, a brief review of these four categories is given. (1) Knowledge-based methods are rule-based methods, which encode human knowledge about what a face is. For example, in our human mind, symmetric eyes, ears, nose and mouth are key face feature. Developing these methods in different situations is sometimes difficult because not all states are countable [6] [7]. (2) Feature invariant approaches are regrouping methods with aim to find robust structural features which are invariant to pose, lighting, etc. This method is one of the most important methods for face detection. (3) Template matching methods compute the correlation between patterns of a face and an input image in order to detection. In these methods, the correlation of several patterns of face in different poses and the input images are stored to be a criterion for face validation. (4) Appearance-based methods use models learned from training sets to represent the variability of facial appearance. Actually in these methods the templates are learned from face image samples. Generally, appearance-based methods utilize statistical analysis and machine learning to find characteristics related to face or non-face images.

Nowadays the field of face detection has made significant progress in the real world applications. In particular, the work by Viola and Jones [8] has made face detection feasible in digital cameras and photo organization software. In this paper, an effective real time face detection system based on Viola approach is presented. The framework of our face detection system consist of: (1) Collecting faces and non-faces dataset for training; (2) Constructing bank of weak classifiers based on Haar-like features; (3) Building strong classifier (face detector) by boosting technique; (4) Set up digital camera to capture input images and develop a software to connect the hardware.

The rest of this article is organized as follows. In Section 2, we will give brief overviews on feature selection for our face detection task, especially the Haar-like feature and its fast computational framework. In Section 3, we focus on Adaboost [9] learning algorithm to train a strong classifier for face detection. In Section 4, face detection procedure is presented. In Section 5, the performance of our real time face detection system is analyzed. Finally, Section 6 presents a summary of the paper.

II. FEATURE SELECTION

A. Haar-like features

Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. Historically, working with only image intensities (i.e., the RGB pixel values at each pixel of image) lead to expensive computation in feature calculation.

978-1-4673-6850-6/15 $31.00 © 2015 IEEE
DOI 10.1109/ICISCE.2015.95

Papageorgiou et al. [10] proposed to working with a simple feature set based on Haar wavelets instead of the usual image intensities in order to make the recognition task more efficient. Viola and Jones [8] developed Haar-like features by adapting the idea of using Haar wavelets. The idea of Haar-like feature is to consider the adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and compute the difference between these sums. This difference is then used as a feature response to categorize subsections of an image.

Figure 1 shows three kinds of rectangle Haar-like features. The value of a two-rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent. A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles.

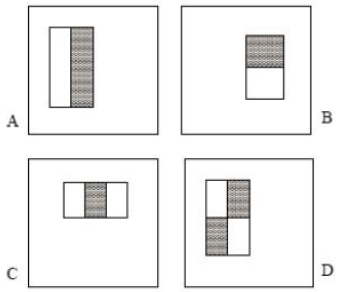


Figure 1: Example rectangle features. The sum of the pixels which lie within the white rectangles is subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four rectangle feature.

Why Haar-like features are useful? For example, let us say we have an image database with human faces. We can easily observe that among all faces the region of the eyes is darker than the region of the cheeks. Therefore, a common Haar-like feature for face detection, which is a set of two adjacent rectangles that lie above the eye and the cheek region, can be useful to capture the key characteristics of human faces.

B. Fast computation of Haar-like features by integral image

Rectangle features can be computed very rapidly using the integral image [8]. The integral image at location x, y contains the sum of the pixels above and to the left of x, y inclusive:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y'),$$

where $ii(x,y)$ is the integral image and $i(x,y)$ is the original image. Using the following pair of recurrences:

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

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

where $s(x,y)$ is the cumulative row sum, $s(x,-1) = 0$, and $ii(-1,y) = 0$, the integral image can be computed in one pass over the original image.

Using the integral image any rectangular sum can be computed in four array references (see Figure 2). We can easily see that the difference between two rectangular sums can be computed in eight references. Since the two-rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features.

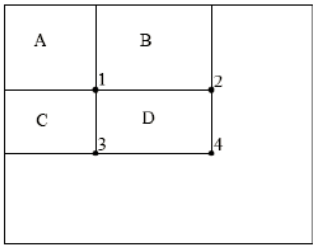


Figure 2. Integral images. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A+B, at location 3 is A+C, and at location 4 is A+B+C+D. The sum within D can be computed as 4+1-(2+3).

III. ADABOOST LEARNING

In its original form, the AdaBoost learning algorithm is used to boost the classification performance of a weak learning algorithm. Viola and Jones [8] provided a variant of AdaBoost both to select a small set of features and train the classifier. In our system, we have over 180,000 rectangle features associated with each image sub-window, which is much larger than the number of pixels. Even though each feature can be computed very efficiently, computing the complete set is still expensive. How to select a small set of these features is the main challenge.

In order to use the Adaboost learning method, we have to define the weak classifiers for this goal. Each rectangle feature can be considered as a weak classifier if we assign a threshold to each feature. For each feature, the weak learner determines the optimal threshold classification function, such that the minimum number of examples is misclassified.

A weak classifier $h_j(x)$ consists of a feature f_j, a threshold θ_j and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1 & if\ p_j f_j(x) < p_j \theta_j \\ 0 & otherwise \end{cases}$$

where x is a 24x24 pixel sub-window of an image. No single weak classifier can achieve high performance in the detection task. However, the boosting procedure can select some of them to build a strong classifier so that it can perform the classification with low error. Figure 3 gives us

a sense of how to use boosting technique in our bank of weak classifiers:

- Given training images $(x_1, y_1), \dots, (x_n, y_n)$ where x represents features and y represents class label. $y = 0,1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}$ for $y_i = 0$, and $w_{1,i} = \frac{1}{2l}$ for $y_i = 1$, where m and l are the number of negative and positive respectively.
- For $t = 1, \dots, T$:

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 with respect to w_t is measured by $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
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
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}$$
where $\alpha_t = log \frac{1}{\beta_t}$

Figure 3. Pseudocode for the AdaBoost algorithm adapted from [4]

The algorithm is first given a set of positive and negative examples. Each of the examples need to be converted to grayscale, scaled to the base resolution of the detector and annotated with a 1 or 0 for positive or negative examples. Next, the algorithm initially assigns equal weights to the examples. The boosting algorithm performs a series of weak classifier selection and weights update, each time selecting a new weak classifier that produces the smallest misclassification error with respect to the weight vector. This step requires classifying all training examples with the bank of weak classifiers, which is computationally expensive. The weights of the correctly classified examples are multiplied by β, while the weights of misclassified examples do not change. Every time after the adjustment of the weights, as well as combing with normalization, this update lead to most of the weight being placed on examples that are hard to classify. Therefore, as the number of trials increases, the error rates also increase, thus leading to smaller α values for weak learners selected later in the training process. The final classification function is the sum of the predictions of the selected weak learners multiplied by the corresponding α values.

IV. Face Detection

After the training of the face detector by Adaboost, we can perform face detection given an input image from camera. The system detects objects by exhaustively scanning images by the detector, window, or so-called template. For each input image, we first convert the image to grayscale and compute the integral image. For multi-scale detection, we use different scales of input images instead of scaling the detector to find the best scale of the object. Generally, we start with an initial scale of 1.0 and evaluates every sub-window with the fixed size of strong classifier. The scale is then increased or decreased and all sub-windows are evaluated. The scale is increased until the detection window is too small compared to the image, and is decreased until the detection window is larger than the image. A sub-window is marked with bounding box as an object occurrence if the strong classifier returns a value of 1. In order to capture some objects that are rotated in the images, instead of rotating the detector, we can also rotate the images and scanning the rotated images with the detector, which is a similar way in multi-scale detection.

V. Implementation and Experiments

Building the face detection system required an annotated dataset of faces, finding optimal threshold values for the features, and training the classifiers. Additionally, the system need a camera to detect faces in real-time. The system is implemented in Java and detects faces at a rate of 10-20 frames per second for images with a resolution of 500x400. The detector window size is 40x40 pixels. We collect a face dataset of 5175 face images and a non-face dataset of 10000 samples. Figure 4 shows a part of face images in our dataset for classifier training.

Figure 4. Faces dataset

We compute the value of each rectangle feature for each example. For easy computation we compute the feature value by using integral image. Each feature corresponds to a weak learner. We approximately determine the threshold between face sample and non-face sample for all the weak learners by assuming the distribution of face sample and non-face sample are normal distribution and set the threshold as the middle between mean of face sample and mean of non-face sample. Figure 5 shows the best 20 features in terms of low classification error before Adaboost. We may find that all the top features are vertical 2-rectangle features. Most of these features measure the difference in intensities between the eye region of the eyes and a region

across the upper cheeks. These features capitalize on the observation that the eye region is often darker than the cheeks. After boosting, some other types of feature are included in the top 20 features, which are shown in Figure 6.

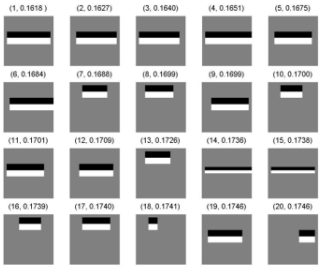
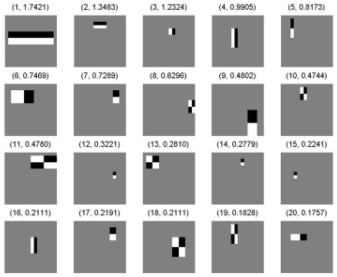


Figure 5. Top 20 features before boosting



. Figure 6. Top 20 features after boosting

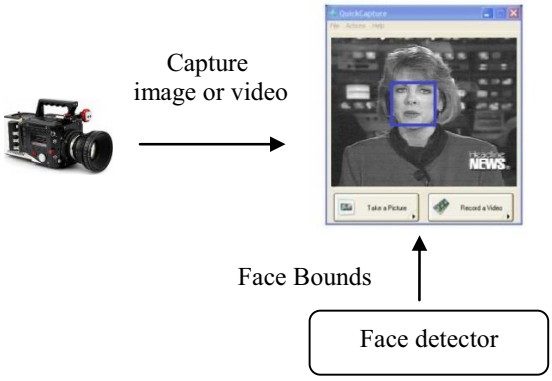


Figure 7. Real time face detection system

The structure of real time face detection system is shown in Figure 7. The accuracy of the face detector was analyzed on images files from the CMU face dataset (available at http://vasc.ri.cmu.edu/idb/html/face/). The dataset contains a variety of faces in different lighting conditions and is considered to be a difficult dataset. Figure 7 shows some face detection results by our system.

I. CONCLUSIONS

Face detection is one of the main challenges of computer vision. Boosting has been shown to be an effective technique for face detection. This paper builds a real time face detection system by using AdaBoost algorithm and Haar-like features. The experiments show high performance in both accuracy and speed of the developed system.

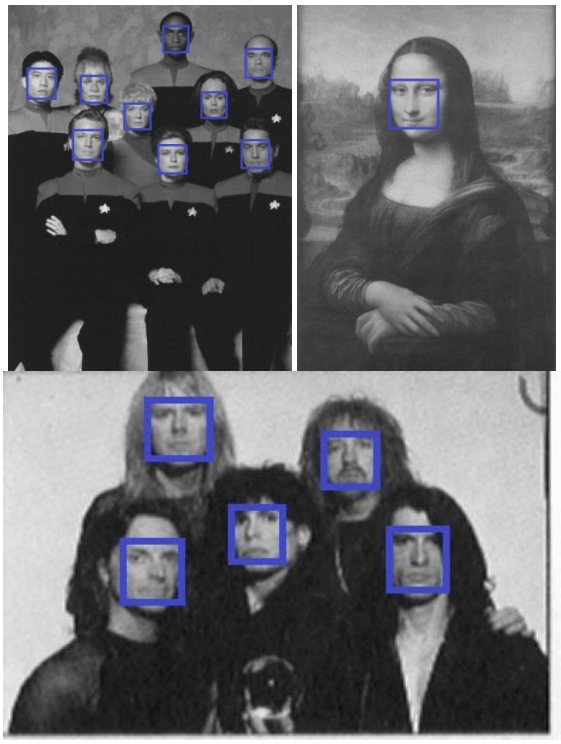


Figure 8. Detection results

REFERENCES

[1] K. Sung and T. Poggio. Example-based learning for viewbased face detection. In IEEE Patt. Anal. Mach. Intell., volume 20, pages 39–51, 1998.

[2] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In IEEE Patt. Anal. Mach. Intell., volume 20, pages 22–38, 1998.

[3] D. Roth, M. Yang, and N. Ahuja. A snowbased face detector. In Neural Information Processing 12, 2000.

[4] H. Schneiderman and T. Kanade. A statistical method for 3D object detection applied to faces and cars. In International Conference on Computer Vision, 2000.

[5] M. Yang and D. J. Kriegman, "Detecting faces in images: a survey "IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no.1, january 2002.

[6] T.K. Leung, M.C. Burl, and P. Perona, “Finding faces in cluttered scenes using random labeled graph matching,” Proc. Fifth IEEE Int. Conf. Computer Vision, pp. 637-644, 1995.

[7] K.C. Yow and R. Cipolla, “A probabilistic framework for perceptual grouping of features for human face detection,” Proc. Second Int. Conf. Automatic Face and Gesture Recognition, pp. 16-21, 1996.

[8] P. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features", IEEE Conference on Computer Vision and Pattern Recognition, 2001.

[9] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt ’95, pages 23–37. Springer-Verlag, 1995.

[10] C. Papageorgiou, M. Oren, and T. Poggio. “A general framework for object detection”. In International Conference on Computer Vision, 1998.