An Efficient Automatic Redeye Detection and Correction Algorithm

Huitao Luo, Jonathan Yen and Dan Tretter
Hewlett-Packard Labs,
1501 Page Mill Road, MS 1203, Palo Alto, CA 94304

Abstract

A fully automatic redeye detection and correction algorithm is presented to address the redeye artifacts in digital photos. The algorithm contains a redeye detection part and a correction part. The detection part is modeled as a feature based object detection problem. Adaboost is used to simultaneously select features and train the classifier. A new feature set is designed to address the orientation-dependency problem associated with the Haar-like features commonly used for object detection design. For each detected redeye, a correction algorithm is applied to do adaptive desaturation and darkening over the redeye region.

1 Introduction

Redeye is a common problem in consumer photography. Currently, many image processing software applications in the market offer redeye removal solutions. However, the majority of them are semi-automatic or manual solutions in that the user has to either click on the redeye or draw a box containing the redeye before the redeye removal algorithms can find the redeye pixels and correct them. It is desirable to design a fully automatic redeye removal algorithm such that no user intervention is needed. Recently, two such algorithms [2, 4] have been proposed. Both algorithms primarily based their redeye detection scheme on general face detection and eye detection algorithms, though other heuristics and features were also discussed.

This paper discusses the design of an automatic redeye detection and correction algorithm that requires no user intervention. Technically, a redeye removal algorithm can be decomposed into a redeye detection step and a redeye correction step, where the detection step is the more challenging part. Unlike the available algorithms [2, 4], we propose a redeye detection algorithm that is less strongly tied to face detection and eye detection. Instead of starting with a strong, but difficult classifier such as face detection, the proposed algorithm starts with a weak, but much easier classifier that identifies red oval regions. The detected red regions are regarded as initial candidates, and they are further verified by a number of classifiers that separate false alarms from real redeyes, before the final redeye detection results are obtained. In this architecture, although the initial detection module and each individual verification classifier may be weak by itself, the final detection algorithm could still be very strong as long as the proper training procedure is used.

In our work, the verification classifiers are trained in two stages: a single eye verification stage and a pairing verification stage. Adaboost [1] is used to train the classifiers because of its ability to select relevant features from a large number of object features. This is partially motivated by the face detection work of Viola and Jones [5]. However, in comparison to their work, our contributions come in three aspects. First, in addition to grayscale features, our detection algorithm utilizes color information by exploring effective color space projection and space conversion in designing object features. Second, we design a set of non-orientation-sensitive features to address the orientation-sensitivity problem associated with the Haar-like features used in Viola and Jones' work. Third, our algorithm uses not only Haar-like rectangle features, but also features of different semantics such as object aspect ratio, percentage of skin tone pixels, etc.

2 System Overview

The proposed redeye removal system contains two steps: the redeye detection and the redeye correction (see Fig. 1). The detection step contains three modules: initial candidate detection, single eye verification and pairing verification. Among them, the initial candidate detection is a fast processing module designed to find all the red oval regions that are possibly redeyes. The single eye verification module verifies the redeye candidates using various object features, and eliminates many candidate regions corresponding to false alarms. Pairing verification further verifies the remaining redeye candidates by grouping them into pairs. Conceptually, single eye verification module plays the role of eye detection and pairing verification corresponds to face detection. However, they are different from general eye and

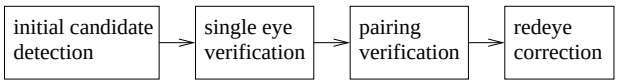

Figure 1. System flowchart

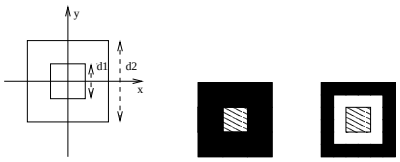

Figure 2. (a) Initial candidate detection filter design, (b) Concentric templates

face detection because the classifiers involved are trained specifically on redeye samples. As will be shown soon, many object features that are not appropriate for general eye or face detection can be used to improve the detection performance.

3 Initial Candidate Detection

This module detects red oval regions that are potentially redeyes. It utilizes two object features: the redness and the shape. To capture the redness feature, we define a relative redness measure $f_r(I)$ for each color pixel I. This forms a redness map for each input image. A contrast filter is then designed to detect red oval regions as follows. At each pixel location, the filter is defined using two concentric squares centered at this pixel (see Fig 2(a)). If we denote the average redness of the inner and outer squares as AR_1 and AR_2 respectively, the filter output is defined as $AR_1 + w * (\frac{AR_1}{AR_2})^n$, where w is a weighting factor, and n is a predefined constant. Note the first term here represents the absolute redness and the second term represents the redness contrast between the inner square and its surrounding margins.

Applying this contrast filter to a redness map followed by a thresholding process yields a binary image, with "1"s represent red areas and "0"s represent background. Note the output of the filter is influenced by the size of the squares. Normally we fix the ratio between $d1$ and $d2$ and refer to $d1$ as the kernel size of the filter. The output of the filter will peak when the kernel size is close to the size of the redeye. To detect redeye of different sizes, filters of multiple kernel sizes are used to process the image, and the output binary images are merged into one image by "or"ing them. The candidate objects are then generated by a standard blob labeling operation, and the final detected candidate is each represented by a containing rectangle (referred to as redness box thereafter), and some basic features such as the aspect ratio and the size of the candidate can be easily computed for false alarm elimination.

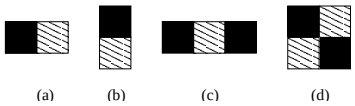

Figure 3. Features used by Viola and Jones

To design redness measure function $f_r(I)$, we manually labeled about 1000 red eyes samples and analyzed their statistical color distributions under different color spaces, from which two measures are designed: one is based on a linear projection of RGB vector as $f_r(I(R,G,B)) = w(4R - 3G + B)$ (where w is a weighting factor), and the other is based on a nonlinear projection to CIE $L^*a^*b^*$ space: $f_r(I(R,G,B)) = I(a^*) + a_0$ where $I(a^*)$ is the a^* component and a_0 is a shifting constant. Overall the L*a*b* measure produced slightly better performance.

4 Single Eye Verification

This module verifies that the redeye candidates detected by the initial candidate detection module are actually red eyes rather than false alarms such as red flowers. We use *feature based object detection* do design the verification classifier because feature computation helps compress and select relevant data and feature design can be controlled to represent ad hoc domain knowledge that otherwise is difficult to learn from a limited quantity of training data.

4.1 Feature Design

For object recognition, simple Haar-like features have been used by Papageorgiou [3]. Viola and Jones also used three kinds of Haar-like features in their face detection work [5] (see Fig. 3): a *two-rectangle feature* (a-b), a *three-rectangle feature*(c), and a *four-rectangle feature* (d). Fundamentally, this set of features corresponds to projecting the image to a set of over-complete Wavelet basis functions. It has been proven quite successful in detecting upright, frontal human faces [5]. However, it is not quite applicable to this work because it is orientation sensitive. In order to detect a single eye in arbitrary orientation, we design a different set of rectangle features that is motivated by [3, 5], but extends their works to non-orientation-sensitive, color object recognition.

The proposed design defines a feature by two factors: a feature plane and a feature template. A feature plane is a scalar image on which the rectangle features are computed. Note in both [3, 5], only the grayscale feature is used and the grayscale image works as the default feature plane. In

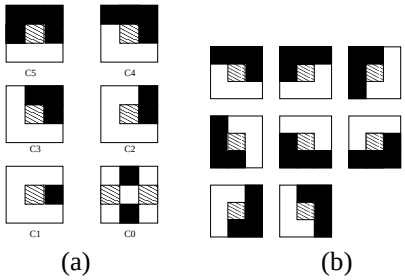

Figure 4. (a) Circular templates, (b) Eight rotated positions of a circular (C4) template

redeye detection work, color is used in addition to grayscale information.

A feature template specifies how a feature is computed over a feature plane. For example, under this definition, Fig. 3 serves as a set of feature templates for Viola and Jones' work. In our approach, the proposed feature templates are composed of a number of concentric rectangles of the same shape grouped in two classes: concentric templates, and circular templates, illustrated in Fig. 2(b) and Fig. 4(a) respectively. A feature based on a concentric template is obtained by subtracting the pixel average over an inner rectangle (dashed area) from the pixel average over another outer margin area (solid gray area). Note the solid gray area and the dashed area do not have to be immediately adjacent. To compute a feature using circular template (C5-C1), (see Fig. 4), the pixel average over a central rectangle (dashed area) is compared with the pixel average over its eight neighboring rectangles, which have the same size and shape. In this comparison, different circular template considers different number of neighboring rectangles (solid gray area). To make the final feature non-orientation-sensitive, each template is rotated circularly to eight different positions (see Fig. 4(b) for an example of template C4). The template value at each position is computed, and their maximal and minimal values are used as the two features yielded from this feature template. Note the template C0 is an exception to circular template definition, in that it compares two pairs of neighboring rectangles, rather than the center rectangle and its neighboring rectangles.

With the above feature definition, the whole feature set is populated by changing the size of the rectangles, changing the control parameters for each template, and using different feature planes for feature computation.

Grayscale Feature: Grayscale feature is computed by applying the feature templates to the grayscale feature plane. To make grayscale features comparable across candidates of different size, we first compute the iris area (modeled as a dark square) of each candidate, and normalize each against its iris square. The computing of the iris area is based on the grayscale information described as follows.

Given an initial redeye candidate, the iris square is assumed to share the same center with its redness box, and the iris size is determined through a searching procedure. First, a template is defined using two concentric squares, with the inner one having half the side length of the outer one. The output of the search template is defined to be the difference between the pixel average over the inner square and the average over the outer margin area. To determine the iris size, the size of this search template is searched over a fixed range with respect to the size of the redness box. The inner box of the search template producing the highest output is considered as the detected iris. In this search design, the search range is determined empirically to represent the flexible relation between the redness box size and the actual eyeball size. The red region of the redeye can occupy the whole iris or only a small fraction of the iris, depending on how much the pupil is dilated.

Color Feature: Color features are computed over two feature planes corresponding to the two redness measures defined in Section 3. For feature computation on both planes, each candidate is normalized with respect to its redness box yielded from initial candidate detection.

Other Features: Additional features used include skin pixels in the neighborhood, aspect ratio of the redness box, percentage of pixels within the redness box labeled as red pixels, etc. These features are quite different semantically, and are selected based on heuristics.

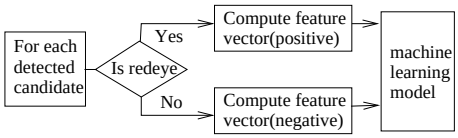

Figure 5. Machine learning design

4.2 Learning Classification Functions

Given the feature set design, a classifier can be trained using a machine learning framework. As illustrated by Fig. 5), each training image is processed by the initial detection module, and the candidates obtained are further manually labeled. Feature vectors are then computed for labeled candidates and the output is sent to a machine learning model to train a binary classifier.

In our system, Adaboost [1] is used as the learning model because the feature set we designed in Sec 4.1 are heuristics based. And if fully populated, the size of the feature set will be very big, representing fair amount of redundancy. Adaboost's ability to select relevant features (from a large feature set) during the classifier training process makes it the right choice for this application.

To include feature selection ability in Adaboost, a weak learner $H(x)$ is restricted to a set of simple classification functions $h_j(x)$ in which each classifier uses only one feature. The learning algorithm includes two steps. First, each classification function $h_j(x)$ is optimized individually by finding the optimal thresholding function that minimizes its classification error. Second, the classification function that produces the smallest classification error is selected as the final weak learner function. We show the boosting procedure using the pseudo code detailed in Table 1. As indicated, T simple classifiers are constructed each using one single feature, and the final strong classifier is a weighted linear combination of these T simple classifiers, where the assignment of weights is such that the higher the training error e_t, the smaller the corresponding weight α_t.

Practically, the total feature set (Section 4.1) is populated as follows. Each feature template is applied 4 times using 4 different scales. Since each circular template produces 2 features (maximal and minimal), each concentric template produces 1 feature, and there are 6 circular templates, 2 concentric templates and 3 feature planes, this creates a total of $(6 \times 4 \times 2 + 2 \times 4 \times 1) \times 3 = 168$ features. The feature set totals 178 when adding 10 features from *other features category*. The final classifier in our work utilizes 25 out of the 178 features.

Once the single eye verification classifier is designed, the pairing verification module can be designed similarly. The details are omitted due to space limits.

Input: Given training feature vectors $(x_1, y_1), \cdots, (x_n, y_n)$, where $y_i = 0, 1$ for negative and positive examples respectively.

Boosting Algorithm:

I. Initialize weights $w_i^{(0)} = 1/(2m), 1/(2l)$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive examples respectively.

II. For $t = 1, \cdots, T$:

1) Normalize weights: $w_i^{(t)} = w_i^{(t-1)} / (\sum_{k=1}^{n} w_k^{(t-1)})$.

2) For each feature j, train a classifier $h_j^{(t)}()$ that is restricted to using a single feature that minimizes the classification error $e_j^{(t)} = \sum_i w_j^{(t)} |h_j(x_i) - y_i|$.

3) Choose the weak learner function as the $h_j^{(t)}$ with the lowest error $e_j^{(t)}$, e.g., $H_t() = h_{j_0}^{(t)}()$, $e_t = e_{j_0}^{(t)}$, where $j_0 = \arg\min_j (e_j^{(t)})$.

4) Update the weights: $w_i^{(t)} = w_i^{(t)} \beta_t^{1-\epsilon_i}$, where $\epsilon_i = 0$ if example x_i is classified correctly; $\epsilon_i = 1$ otherwise, and $\beta_t = \frac{e_t}{1-e_t}$.

Output: The final classifier is:

$$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t H^{(t)}(x) > 0.5 \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}$$

where $\alpha_t = -\log \beta_t$.

Table 1. Adaboost Pseudo code.

5 Detection Experiments

The proposed redeye detection system is trained on a data set of 420 redeye images with 1070 redeyes manual labeled. The training images are collected from various sources, including scanned images, digital camera images, and Internet photos. The typical image size is 3M pixels. Redeye sizes range from 25 pixels to over 10K pixels.

On this training data set, the three detection modules are trained/adjusted sequentially (see Fig. 1) in three stages. Due to the cascade architecture of the system, the first few modules can afford more false alarms as long as the detection rate is high. In our experiment, the initial redeye detection module is adjusted to detect 98% of the redeyes, and generates 59800 false alarms. In the next stage, the single eye verification classifier detects 94% of the redeyes, and generates 2100 false alarms. In the final stage, pairing verification classifier further reduce the false alarm number to 150, and generates a detection rate of 87.1%.

6 Redeye Correction & Conclusion

Once the locations and sizes of the redeyes are determined, a correction algorithm is applied to each detected redeye. In this algorithm, a pixel mask is first generated, the mask is then refined and pixel values are desaturated. Though we do not have space to cover details, one interesting feature about redeye correction is that in most cases when a false alarm from redeye detection is desaturated, the final result is normally not perceivable by the end user [1]. This helps further reduce the *effective* false alarm artifacts of the system. Overall, the proposed algorithm is very efficient computationally and we have actually implemented it on a number of embedded platforms in HP products.

References

[1] Y. Freund and R. Schapire. A short introduction to boosting. *J. of Japanese Society for AI*, pages 771–780, 1999.

[2] M. Gaubatz and R. Ulichney. Automatic red-eye detection and correction. In *ICIP-2002*.

[3] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *ICCV*, 1998.

[4] J. Schildkraut and R. Gray. A fully automatic redeye detection and correction algorithm. In *ICIP-2002*.

[5] P. Viola and M. Jones. Robust real-time object detection. In *ICCV Workshop Statistical Comp. Theories of Vision*, 2002.

[1] As long as the false alarm is not located at the region of interest of known colors.