RED EYE DETECTION WITH MACHINE LEARNING

Sergey Ioffe

FUJIFILM Software (California)
1740 Technology Dr., Suite 490, San Jose, CA 95110
sioffe@fujifilmsoft.com

ABSTRACT

Red-Eye is a problem in photography that occurs when a photograph is taken with a flash, and the bright flash light is reflected from the blood vessels in the eye, giving the eye an unnatural red hue. Most red-eye reduction systems need the user to outline the red eyes by hand, but this approach doesn't scale up. Instead, we propose an Automatic Red-Eye Detection System. The system contains a red-eye detector that finds red eye-like candidate image patches; a state of the art face detector used to eliminate most false positives (image regions that look but red eyes but are not); and a red-eye outline detector. All three detectors are automatically learned from data, using Boosting. Our system can be combined with a red-eye reduction module to yield a fully automatic red eye corrector.

1. INTRODUCTION

Red-Eye is a commonly occurring problem in photography. The problem occurs when a photograph is taken with a flash, and the bright flash light is reflected from the blood vessels in the eye, giving the eye an unnatural red hue. The quality of a picture can be significantly improved if the red eyes can be fixed. Much research has been done on fixing the red eye, replacing the red pixels with a more natural eye color, once the image area to be fixed has been indicated. However, if many images need to be processed, for example in a photo lab, having a human examine each image and outline the red eyes is unreasonable. We need a method to automatically identify the red eyes in consumer images. Such a method can be combined with a red-eye reduction algorithm to automatically reduce this artifact.

We have designed a system that automatically detects red eyes. The output of the system is a binary mask indicating, for each image pixel, whether or not this pixel needs to be passed to the correction algorithm. In this paper, we describe the system's components. In section 2 we describe how to use computer vision and machine learning techniques to train a detector that outputs a list of possible red eyes for an image. In section 3, we describe how to eliminate many false positives, using the fact that a red eye must occur on a human face. We describe our state of the art face detector which is both accurate and efficient, and leaves our system with almost no false positives. In section 4, we show how to produce a binary mask for the image. This involves finding the boundaries of the detected red eyes, which we, again, accomplish with a learned classifier. In section 5, we give an outline of the machine learning method we used to train all modules of our system, and show some results in section 6.

2. LEARNING A RED-EYE DETECTOR

As the first step of our automatic red-eye reduction system, we need an automatic red-eye *detection* module. Red eyes are, essentially, small red circular blobs, but formalizing this observation may be difficult. Furthermore, if we design a red-eye detector by hand, it is hard to ensure that the detector is robust, applicable to many types of red eye. Therefore, we choose the alternative: *learning* the detector automatically from data. In particular, let us suppose that we have sufficient quantities of (*positive*) example images of red eyes, and (*negative*) examples of non-red eye images. We *normalize* the training examples to have a fixed size (re-scaling the red eyes as necessary). Some examples are shown in figure 1(a). We can now apply machine learning methods to train a classifier that distinguishes between positive and negative data. Such a classifier looks at an image patch of a fixed size (in our case 11×11) and determines whether or not it represents a red eye. To detect red eyes in an image, we scan the image, and apply the classifier to the 11×11 image patches at all possible positions. Furthermore, since red eyes may appear in a variety of size, we also apply the detector to scaled versions of the image.

We represent each image patch with a set of features, each with a finite set of possible values. This representation lends itself to a training method (Boosting) outlined in section 5. To obtain our representation, we apply the following steps to each image:

Change the color-space: For red-eye detection, we are interested in the color properties such as the brightness and redness, and the individual RGB color channels do not cap-

0-7803-7750-8/03/$17.00

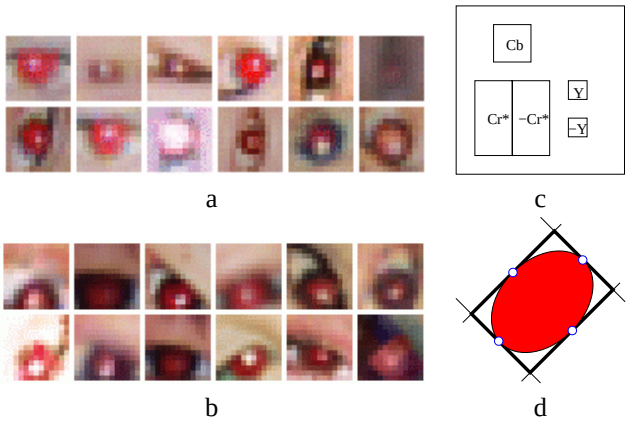


Fig. 1. *(a) Examples of red eyes used to train the red eye detector. (b) Examples used to train the detector for the top edges of red eyes. These were obtained by placing the top edge point of the red eye, rather than its center, at the center of the image patch. (c) Some possible features. Each looks at one color channel (from YCbCrCr*) and computes either the average within a block, or a difference of two such averages. (d) To detect the red eye outline, the top-edge detector is applied to 8 rotated versions of the red eye (only 4 detected edges are shown for clarity). The dots show the detected positions, and the area outlined by the thick polygon is passed to the red-eye corrector.*

ture this information. Instead, we use the YCbCr color-space, which captures the luminance, blueness and redness measurements of each pixel. Because skin often appears as red as some red eyes, we augment the color-space with a channel (Cr*) which has been designed (using Linear Discriminant Analysis) as the linear function of RGB that maximizes the difference between red-eye and skin pixels.

Combine multiple pixels: Each feature uses pixels from a particular color channel. Each feature is either an average value of a channel over a rectangular region, or a difference of such averages in two separate rectangular areas. The averages can be computed efficiently, in a constant time per pixel (time is independent of the rectangular block size) using a technique similar to Integral Images (see e.g. [1]).

Quantize: Finally, the features are quantized (into 64 levels), allowing us to store them in histograms.

Now that we can represent both positive and negative data with discrete features as described above, we can train a classifier using Boosting [2, 3, 4], as described in section 5. During training, we select a subset of features and a (64-bin) histogram for each feature. To detect red eyes, we consider the image as well as it scaled versions (we scale by factors $2, 4, 8, \ldots$). We scan the image and classify each 11×11 patch as either a red eye or not by computing the discrete value of each selected feature, looking it up in the corresponding histogram, and adding up the results. The patch is a red eye if the cumulative response exceeds a threshold. Non-maximum suppression is finally applied to keep only local maxima of the classifier response.

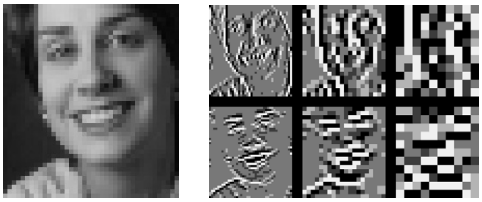


Fig. 2. *Representing a candidate face image for face detection. Given an image (left), a discrete wavelet transform is computed and quantized (right); we keep the HL and LH bands at 3 scales. Each feature used for detection is a combination of 4 quantized DWT coefficients.*

3. USING FACES TO IMPROVE RED-EYE DETECTION

The red eye detector looks for small image patches that look like red eyes. While this allows us to find the red eyes, many False Positives – images that look like red eyes but are not – result. To eliminate these false positives, we use the fact that a red eye must be a part of a face. Therefore, we can eliminate the spurious red eye candidates by examining the image regions around each one and eliminating the candidate if a face is not found.

To determine whether an image region contains a human face, we train a *Face Detector*, using a set of *positive* images containing faces, geometrically normalized using hand-label landmarks, and *negative* images without faces. Our classifier distinguishes faces from non-faces based on some image features, similar to those used in [5]. We represent images for face detection with quantized wavelet coefficients of the image luminance. In particular, we compute the Discrete Wavelet Transform of the luminance using a 5/3 filter bank, and keep only the HL and LH bands at 3 levels of decomposition (that is, we use the responses to horizontal and vertical bar filters at 3 scales). The wavelet coefficients are contrast-normalized using local image variance and quantized to 7 levels each, as shown in figure 2. Each feature used for face detection is a combination of 4 quantized DWT coefficients. Therefore, each candidate face image is represented with a set of discrete features, and the training procedure of section 5 can be used to obtain a face detector.

A combination of the red-eye detector combined with a face detector allows us to find image patches that look like red eyes each of which is a part of a face-like image region, thus ensuring that most of the detections are in fact red eyes.

In practice, the red eye detector is faster than the face detector. Therefore, we apply the red-eye detector *before* the face detector, and consider a candidate face only if a red eye was detected nearby.

4. DETECTING THE EYE OUTLINE

To automatically correct the red eyes in photos, we need to know which pixels constitute the red eyes. The method described above allows us to find the red eyes, but it gives only the position of the red eye, whereas what we need is the outline of the image area corresponding to the red eye. One way to detect such an outline would be using a form of edge detection. However, to make sure the outline detector is robust and works for many types of red eye, we choose to *learn* it automatically.

We train a classifier to respond to the edges of the red-eye region. In particular, it will respond to the *top edge* of the red eye; the edge can be traced by applying this classifier to multiple rotated versions of the image.

The edge detector is trained in a way identical to the red-eye detector, except for the geometric normalization of positive training examples. When we trained the red eye detector, the training red-eye examples were centered at the center of the 11×11 image patch. Training the edge detector, rotate each red eye by a random angle and place the upper-most point of the rotated red eye at the center of the image patch (as in figure 1(b)), thus ensuring that the detector is anchored to the edge rather than the center of the red eye. Once the red eye (i.e. its center) is detected, the entire outline is computed by applying the top-edge detector to the 8 rotations of the image (by multiples of 45°) and intersecting the 8 half-planes induced by each edge, as shown in figure 1(c). In addition to telling the corrector which pixels to change, the outline helps us reject some false positives. We keep an eye outline only if its edges have sufficiently strong detector responses and its aspect ratio is close to 1, since we expect the red eyes to be roughly circular.

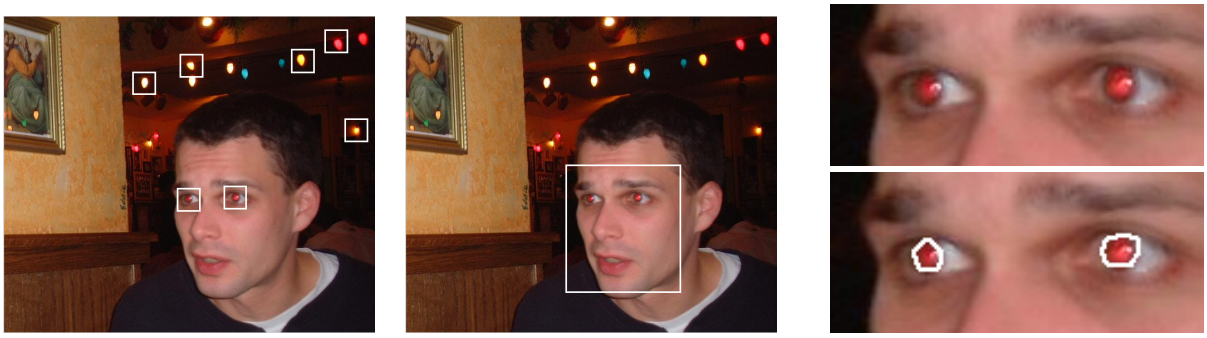
5. TRAINING A CLASSIFIER USING BOOSTING

Boosting [2, 3, 4] is a family of learning algorithms that produce a classifier that is a weighted combination of very simple, "weak" classifiers. Any boosting algorithm takes a set of positive (for example, faces) and negative (non-face) examples and learns a classifier $f(X) = f_1(X) + f_2(X) + \ldots$ such that $f(X) > 0$ for positive examples X and $f(X) < 0$ for negative examples. Each $f_k(X)$ is a *weak classifier*. To train the classifier, we learn the weak classifiers f_k in a sequence, and update the weights of the training examples so that they are higher for the examples that are more difficult, i.e., those which are misclassified by the combination of the weak classifiers learned so far.

We use the method of [4] to maximize the objective function $\Phi = \prod_{i=1}^{N}(1 + e^{-y^i f(X^i)})^{-1}$ where $X^1 \ldots X^N$ are the training examples, and $y^i = 1$ for positive examples and -1 for negative ones. Thus, the classifier should not only correctly classify the training data but also have a high margin between the two classes. To optimize Φ, we update the weights of the training examples after each iteration, so that, when training the $(i+1)$th weak classifier f_{i+1}, a training example $X = (X_1, \ldots, X_K)$ with label $y = \pm 1$ receives the weight $\frac{1}{1+e^{yf(X)}}$. The image features we use for red-eye and face detection can take on a finite number of values, and it is natural to consider weak classifiers that look at a single feature. For a specific feature X_k, we can learn the probabilities of observing each value x_k in an object of interest $P(X_k = x_k \mid y = 1)$ and in a random image of a background $P(X_k = x_k \mid y = -1)$. Each distribution is learned from a set of *weighted* examples and is represented with a *histogram* – with a bin for each feature value x. It can be shown that the optimal weak classifier based on the feature X_k must output $f_{i+1}(X) = f_{i+1}(X_k) = \frac{1}{2}\log\frac{P(X_k=x_k \mid y=+1)}{P(X_k=x_k \mid y=-1)}$. At each iteration of boosting, we try each possible feature, compute the corresponding weak classifier, evaluate the resulting value of the objective function Φ and select the feature for which Φ is the highest.

Our detector must have a very low false positive rate, since each photograph will contain a huge number of potential faces or red eyes, at different positions and scales. Thus, we cannot have a reasonably-sized set of representative negative examples. Instead, we use *bootstrapping*: after every few weak classifiers, we apply the detector to background images and extract the patterns falsely classified as positive. These false positives become the negative training set for the next several training iterations.

We make our detector fast by avoiding the computation of the entire sum of the weak classifier responses $f(X) = f_1(X) + f_2(X) + \ldots + f_N(X)$ at each location. Many negative patterns can be rejected early, from just a few leading terms in that sum. To implement this *early rejection*, we compute a set of thresholds $\theta_1, \theta_2, \ldots, \theta_N = 0$, such that a pattern X is classified as positive if not only $f(X) > 0$ but also, for each i, the ith partial sum $f_1(X) + \ldots + f_i(X) \geq \theta_i$. We compute these intermediate thresholds by considering a large number of positive examples, selecting those that are correctly classified (i.e. those for which $f(X) > 0$), and computing $\theta_i = \min_X(f_1(X) + \ldots + f_i(X)$. Thus, if a face example X is correctly classified, then for each i we have $f_1(X) + \ldots + f_i(X) \geq \theta_i$. Thus, for each candidate image window X, the algorithm computes $f_1(X)$, $f_2(X), \ldots$ in order, maintaining the cumulative response $s = f_1(X) + f_2(X) + \ldots + f_i(X)$, for $i = 1 \ldots N$. If $s \geq \theta_i$, proceed with the evaluation of $f(X)$; otherwise, abandon this candidate and move on to the next one. This



a b c

Fig. 3. *Red-Eye and Face Detection results. (a) A red-eye detector is applied to the image; the squares indicate where the detections occurred. Several false positives result. (b) To combat the false positives, a face detector is used. The rectangle shows the face found in the image. (c) The outline detector is applied to the image regions where both a red eye and a face were detected. The top image shows an enlarged image area where both detectors fired, and the bottom shows the outlines detected with the learned red-eye edge detector.*

way, we quickly eliminate most negative examples.

6. RESULTS

We have trained the red-eye detector, the red-eye outline detector, and the face detector. The positive training examples for the first two detectors were manually selected from color images containing red eyes, and then normalized by being rescaled to a certain range of sizes, rotated by an arbitrary angle, and centered. The face detector was trained on grey-scale images from the CMU frontal face data set (collected by Rowley, Schneiderman, Baluja and Kanade) and our own face database.

The overall detection system first uses the detector of section 2 to obtain candidate red eyes. Then, it eliminates false positives by applying the face detector of section 3 to the image areas around the red eye candidates. Finally, the detector of section 4 is applied to the remaining red eyes to obtain the outline that is passed to the correction module, and helps further eliminate false detections.

The accuracy of the face detector has been evaluated on the standard CMU frontal face database (117 images with 511 faces). We achieve a 91% detection rate with the average of 0.33 false positives per image, or an 80% detection rate with 0 false positives. This compares favorably to the state-of-the-art face detection algorithms such as [1, 5].

Our system has a very low false detection rate, which is necessary for bulk image processing. In fact, on 300 images containing red eyes, only one false positive is found. In spite of this low false positive rate, we obtained a detection rate of 74% on a collection of images containing red eyes of varying severity (some severe, some completely inoffensive); the detection rate for the severe red eyes alone was considerably higher. Some examples of the results produced by the red-eye detector and the face detector separately and in conjunction are shown in figure 3.

Our system is fast, taking the average of 1.2 seconds per 1200×900 image on a 2.2GHz Pentium, where we consider multiple orientations of the image since faces may not be upright (for example, if the head is tilted or a negative is scanned upside-down). Due to the system's extremely low false positive rate, high detection rate, and speed, it is practical enough to be used in a product. Indeed, combined with a red-eye corrector, it will reduce most severe red eyes and many non-severe ones, without compromising the image quality by incorrectly changing the color of the image regions not containing red eyes.

Acknowledgements

Thanks to Thomas Leung, Troy Chinen, Ken Brown and Nobu Nakajima for helpful discussions.

7. REFERENCES

[1] Paul Viola and Michael Jones, "Robust real-time object detection," *International Journal of Computer Vision - to appear*, 2002.

[2] Yoav Freund and Robert E. Schapire, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, 1996, pp. 148–156.

[3] R. Schapire, "The boosting approach to machine learning: An overview," in *MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA*, 2001.

[4] Michael Collins, Robert E. Schapire, and Yoram Singer, "Logistic regression, adaboost and bregman distances," in *Computational Learing Theory*, 2000, pp. 158–169.

[5] H. Schneiderman and T. Kanade, "A statistical method for 3d object detection applied to faces and cars," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2000, pp. 746–51.