Automatic Identification of Red-Eye in Photos

Paul Cuff
Department of Electrical Engineering
Stanford University
Stanford, CA, 94305, USA
E-mail: pcuff@stanford.edu;

***Abstract*— A simple algorithm is proposed for detecting the red-eye defect in photographs on a pixel-by-pixel basis. The algorithm implements a support vector machine using the degree of redness in the surrounding context as features. Results are sub-par, chiming in at 9% false positives and 15% false negatives.**

I. Introduction

From personal experience with photos, I realize that red-eye removal is still a serious annoyance. Many software applications require the user to manually identify red-eye in order to remove it. For this project I use a support vector machine (SVM) to identify which pixels in a picture have red-eye. This can serve as the first step to automatically replacing the red with appropriate colors. After red-eye is successfully identified in photos, a number of things can be done to automatically correct it. That is beyond the scope of this project.

Algorithms do exist which purport to solve this problem. Due to their often non-free nature, I am under the impression that they are complicated and proprietary. One goal of this project is not only to solve the problem but to do so in a simple manner, demonstrating the capabilities of the SVM.

A group at the University of Dublin/Trinity College [1] attempts to solve this same problem. However, their algorithm is very hand-tuned and doesn't take advantage of the machine learning techniques that we have studied in this course. Their algorithm also misclassified the eyes in various specific settings; in my opinion this is because it was so carefully designed for the typical face in an ideal environment.

I do the entire identification on a pixel-by-pixel basis, without first locating the eyes or using any heuristics. Because the classification considers each pixel individually, I can train the algorithm quite well with only a few photographs because each supplies many training pixels of data.

The features that will be passed into the SVM will come from the "context" of the pixel in question. By context I mean a small section of the photograph centered around the pixel. It seems reasonable that this could work well since the "context" of a red-eye pixel should have red in the center surrounded by some darker color which is surrounded by some white. Patterns like this should be picked up automatically during the training of the SVM.

Finding an appropriate inner-product for the SVM in this problem provides some difficulty. The difference in photographs is not measured well by the difference in color intensities of the pixels for several reasons. One is that the photos might not have the same temporal scale; one might be larger than the other, having different resolution, but still be a very similar picture. Another problem is that two identical pictures with different brightness will not cancel each other through subtraction. The approach that I take to resolve these concerns is discussed in section III.

The exact feature calculation is explained in section IV, and the collection of training data and SVM parameters are explained in section V. Finally, results are summarized in section VI, and conclusions are drawn in section VII.

II. Description of Problem

When a photograph is taken of a face sometimes the flash causes the eyes to glow red. The pupil appears red instead of black and is often larger than normal. We call this effect (and the affected area) red-eye. The automatic algorithm must correctly identify all pixels included in the red-eye while not

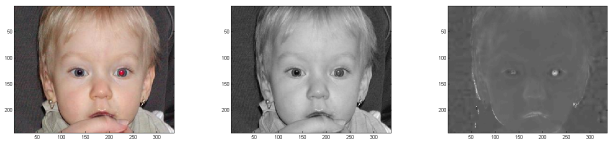

Fig. 1. Data matrices: Original photograph (left), data matrix I (center), and data matrix C (right)

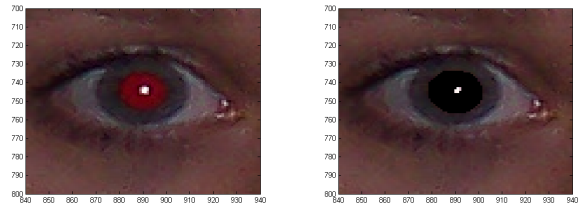

Fig. 2. Manual selection of red-eye: Original photograph (left) and manually selected red-eye marked in black (right)

incorrectly classifying other regions as red-eye, such as lips or a red coat.

III. Approach

I train a support vector machine to classify a pixel as red-eye or not based on the context of the photograph around the pixel. Because each photograph is taken in different lighting and at different resolution, I do a couple of things to normalize the contexts so that they can be reasonably compared with the Gaussian kernel. The degree of redness of each pixel is weighted more heavily in the feature set (see section IV) than the overall intensity. Degree of redness is invariant to intensity, so that helps normalize the lighting condition. In addition, I use several different scalings of each context (by downsampling) when collecting the training samples (see section V). This way, if the resolution of two photographs are not exactly the same they might at least be comparable for one of the combinations of scalings.

IV. Features

A photograph can be described using three matrices of equal size, R, G, B, that specify the brightness of red, green, and blue in each pixel, respectively. The element in the ith row and the jth column of R will be designated R_{ij}, and likewise for the other colors.

I form two data matrices, I and C (illustrated in figure 1), from which I will take the features for the algorithm. I and C are the same size as the original photo, and I (intensity) represents the brightness of each pixel while C (color) represents the redness.

$$\begin{aligned} I_{ij} &= R_{ij} + G_{ij} + B_{ij}, \\ C_{ij} &= \frac{Rij}{I_{ij}}. \end{aligned}$$

The features that I use for the support vector machine for a given pixel are all the entrees of I and C in a 41×41 square centered around the pixel in question. However, for the features that come from I, the set is first normalized to have zero mean and a standard deviation of 0.03; this way the overall intensity of the photograph is ignored and the intensity features play a less important role in the inner product than the color features.

V. Training

I took four photographs that contained a total of seven faces affected by red-eye and four photographs that did not contain red-eye. I first manually identified the red-eye in photos where it was present, as illustrated in figure 2. The top image in the figure shows a section of the original photo, and the bottom image shows the same section with the pixels designated as red-eye indicated in black.

I randomly selected 100 red-eye pixels and 100 non-red-eye pixels from each of the affected photos, as well as 100 pixels from each of the photos that didn't contain red-eye. For each selected pixel I collected up to three training samples by gathering the features described in section IV at three different zoom scales. In other words, I gathered one sample using features exactly as described, one after downsampling the photo by two, and one after downsampling the photo by four. In cases where the selected pixel was too close to the edge of the photo to retrieve the full context (set of features), that data sample was omitted. In all there were about 3,600 training samples.

The support vector machine used a Gaussian kernel with $\sigma^2 = 176.4$ and the ℓ_1 norm regularization scaling factor was $C = 10$.

Fig. 3. Automatic Classification: Original photograph (left) and red-eye marked in black (right)

Fig. 4. Automatic Classification: Original photograph (left) and red-eye marked in black (right)

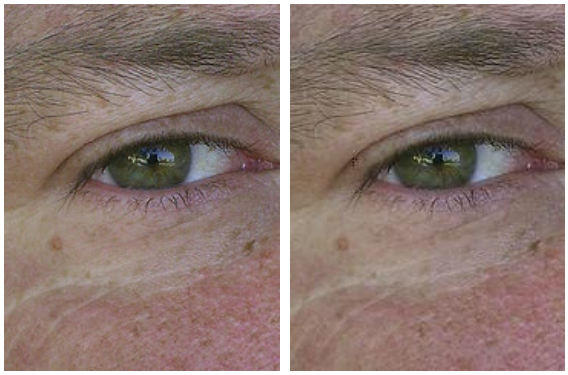

Fig. 5. Automatic Classification: Original photograph (left) and red-eye marked in black (right)

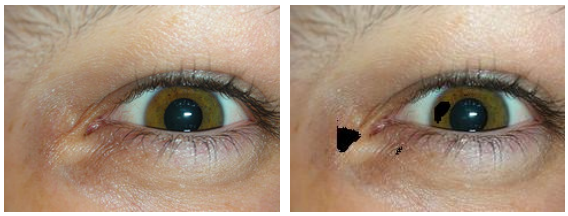

Fig. 6. Automatic Classification: Original photograph (left) and red-eye marked in black (right)

VI. Results

I tested the algorithm on eight photos that weren't used for training. In these photos, each pixel that wasn't too close to an edge was classified by gathering the features in the context around it after down-sampling by a factor of two. Overall misclassification was around 9.5%, with false positives at 9% and false negatives at 15%.

In figure 3, figure 4, figure 5, and figure 6 the pixels classified as red-eye are marked in black.

Figure 3 shows an example where the red-eye is correctly identified, but also some skin locations are incorrectly classified as red-eye. Figure 4 nicely illustrates the fact that the algorithm is not only searching for red color. Notice that the lips are red, but they are not falsely identified as red-eye. On the other hand, some stray marks on the forehead and ear are made, and not all of the red-eye is completely identified. Figure 5 shows a case where no red-eye exists and the algorithm correctly handled it. Figure 6 also shows a case with no red-eye, yet the algorithm has some false positives in the skin region again.

VII. Conclusion

The red-eye detection algorithm presented in this paper takes a simple approach to detect red-eye in a photograph using only a support vector machine and information from the context of each pixel. Other tools could be used in conjunction with this, such as first identifying faces and eyes before searching for red-eye, but these would also be at the expense of the simplicity of the algorithm.

The current performance of this algorithm is too poor to be useful for practical purposes. The false positive error rate is 9% on a pixel-by-pixel basis, which means that most photographs will contain errors. Perhaps with more training data this algorithm could perform acceptably.

References

[1] H. Y. Hui, "Automated Red-Eye Detection & Correction," Final Year Project, The University of Dublin — Trinity College, April 2004.