CSC487/2503 – Computational Vision – Fall 2007

Assignment 3: Detecting Eigen-Eyes

Due: at the beginning of the lecture, 1:10 pm., Thurs., Nov. 15
This assignment is worth 15 marks towards your grade in this course.

This assignment explores principal component analysis (PCA) for representing images. Our objective in this assignment is to gain some experience in using PCA and in evaluating the characteristics of detectors built using PCA models. The general idea is that a fairly weak detector can be useful given further constraints on the desired solutions. Here the additional information we lean on is that eyes appear in skin coloured blobs, and two eyes typically appear together with a known separation.

Familiarize yourself with the material in the `utvisToolbox/tutorials/eigenTut` tutorial.

Download `eyeFinderHandout.zip` from the course web page.

What to hand in. Write a short report addressing each of the itemized questions below (hand-written reports are fine). You can assume that the marker knows the context of the questions, so do not spend time repeating material in the hand-out or in class notes. Include print outs of the output from your program in your report. Also, please email `mbrubake@cs.toronto.edu` each of the Matlab files that you altered or created. Undergrads do not need to do problems 1 and 3 below (they will be given extra credit, according to the grad marking scheme, if they do).

Training Data. One large set of eye and non-eye images has been split into two disjoint sets, one in `trainSet.mat`, the other in `testSet.mat`. We will use the former to train our eye detector and, once the training is complete, we will use the latter to test its performance on a new data set.

The eye images in `trainSet.mat`, have been warped to be roughly of constant position, orientation, and scale (the scale was set by specifying an inter-eye distance of about 40 pixels). These warped images were then cropped to be of size 20×25. They are represented as 500 dimensional columns in the Matlab matrix `eyeIm`. (There is also a set of non-eye images in `nonIm`, which we will use to tune the detector.)

The handout code `trainPCA.m` loads the training set of images. It then massages the training eye images to account for variations in image contrast. This is done by dividing the image by a blurred version of itself. This preserves local contrast changes, but largely suppresses global shading variations. Then the resulting images are rescaled and the mean image is projected out. This results in a training set, say $D_k(\vec{x})$ for $k = 1, \ldots, K$ of normalized, projected eye images for which the PCA components are extracted.

The SVD then produces the basis images $B_j(\vec{x})$ along with the singular values σ_j, for $j = 1, \ldots, 500$. In order to match the principal covariances of the data set, the singular values have

been rescaled by $\frac{1}{\sqrt{K}}$. These quantities are saved in `trainDivEyes.mat` (do not overwrite this file).

There is nothing you need to hand in for this section.

EigenEye Detector. The second script file, namely `trainDetector.m`, uses the above PCA results to develop an eye detector. The detector is different from the one described in `eigenTut`. Again there is nothing you to hand in for this section.

In particular, suppose we have extracted a 20×25 test image patch from a brightness normalized image. Suppose $I(\vec{x})$ is that patch. In order to detect whether or not $I(\vec{x})$ is the image of an eye (at the right position, orientation, and scale) we first project out the mean eye image. The result is the image $D(\vec{x})$. We expand this using the PCA basis $\{B_k(\vec{x})\}_{k=1}^{n_B}$, giving the model image

$$M(\vec{x}) = \sum_{k=1}^{n_B} B_k(\vec{x}) a_k. \tag{1}$$

Here the coefficients a_k are $a_k = \sum_{\vec{x}} B_k(\vec{x}) D(\vec{x})$. Finally the error in this model image is

$$E(\vec{x}) = D(\vec{x}) - M(\vec{x}). \tag{2}$$

We detect unmodeled pixels as points for which the error is larger than a simple threshold involving the magnitude of the projected image $D(\vec{x})$, namely

$$|E(\vec{x})| > c_1 + c_2 |D(\vec{x})|, \tag{3}$$

where $\vec{c}^T = (c_1, c_2)$ is a 2-vector of adjustable coefficients. In general, the threshold coefficients $\vec{c}$ will depend on n_B, the number of basis vectors used. The detector is based simply on the number of pixels which satisfy (3).

Specific questions to be answered are listed below:

1. **[5 pts] Residual Variance.** Suppose we consider approximating the training images $D_k(\vec{x})$ with just the first n PCA basis images, $B_j(\vec{x})$ for $j = 1, \ldots, n$. Define the residual variance:

$$V_n(\vec{x}) \equiv \frac{1}{K} \sum_{k=1}^{K} \left[D_k(\vec{x}) - \sum_{j=1}^{n} b_{k,j} B_j(\vec{x}) \right]^2, \tag{4}$$

where the expansion coefficient $b_{k,j}$ is given by $b_{k,j} = \langle D_k(\vec{x}), B_j(\vec{x}) \rangle$.

Show mathematically that this residual variance $V_n(\vec{x})$ satisfies

$$V_n(\vec{x}) = \sum_{j=n+1}^{500} \left[B_j(\vec{x}) \sigma_j \right]^2, \tag{5}$$

where σ_j^2 is the j^{th} eigenvalue of the data covariance matrix associated with the eigenvector $B_j(\vec{x})$. (This exercise is not meant as a hint for question 3 below.)

2. **[5 pts] Detector performance.** Use `trainDetector.m` to determine the best coefficients $\vec{c}$ (to only one significant digit) for each of the three choices for the basis dimension: $n_B = 10, 20, 50$. Here we wish to adjust the $\vec{c}$ to roughly minimize the false positive rate when the false negative rate is close to 10%.

   The code currently reports the false positive rate for the closest false negative rate it can find below the 10% limit. I want you to minimize this reported false positive rate by adjusting the $\vec{c}$. You can use different $\vec{c}$'s for different values of n_B.

   Modify `trainDetector.m` so that it also generates ROC plot with the three detectors (for $n_B = 10, 20, 50$). (Feel free to copy the relevant code from `eigenTut`.)

   Copy the modified `trainDetector.m` to `testDetector.m`, and run the detectors you tuned above (for each $n_B = 10, 20, 50$) on the data from the test set in `testSet.mat`. (Do NOT refit your detectors on this test set!) This allows us to check that we have not overfit the detector to the training data. The idea is that the results on the test set should be similar to those you get on the training set.

   Do you get similar ROC plots to the ones for the training set? Your report should include the ROC plots for both the training and test results. Also, report your detector results (for both training and test results) in terms of the false positive rate at a threshold where the false negative rate is close to 10%.

3. **[5 pts] Alternative detector.** Try a different approach to do this detection. Implement this approach in two M-files, `trainMyDetector.m` and `testMyDetector.m`. I am particularly interested in smaller false positive rates, while keeping the false negative rate around 10%.

   Your alternative approach could be the detector described in the lecture notes, or a variation on the one implemented in the handout code `trainDetector.m`, or anything else you would like to try. (Although, note that this question is only worth 5 points.)

   In your report, first describe what motivated you to try this detector. Why should it work well? Clearly describe the algorithm, and show ROC plots for both the training and test data. I will not base the mark on the performance of the detector. Just try something reasonable, and report your results.

4. **[10 pts] Skin Coloured Blob Tracker.** Here we consider massaging a given image sequence, such as the one displayed in the beginning of the script file `eyeFinderHandout.m`, so that it can then be processed to find eyes. In order to use the PCA-based eye detector developed above we will need to locate the region around the face, and rescale it to have an inter-eye distance of roughly 40 pixels. The image extent of the face does not change very much in the given sequence, and therefore the code simply uses a single rescaling constant (you don't need to change this). As a result, we only need consider the identification and tracking of the center of a skin-coloured region from frame to frame. In question 5 we will then apply the eigen-eye detector to the skin coloured regions we extract here.

   (a) **[1 pt] Skin Colour Direction**. We follow an approach similar to the one described in Assignment 1 for segmenting a diffusely reflecting surface. In particular, let $\vec{r}$ be the 3-vector denoting the mean of the R, G, B pixel responses in a small hand-selected, skin-coloured region of an image. Let $\vec{d}$ be the "skin-colour" direction defined by normalizing this mean response, that is $\vec{d} \equiv \vec{r}/||\vec{r}||$. Fill in the computation of $\vec{d}$ below the line indicated in `eyeFinderHandout.m`.

(b) **[2 pts] Skin Coloured Pixels**. The general idea is to identify skin-coloured pixels within a test image by testing to see if their R, G, B values fall within a cone whose axis is the skin-coloured direction $\vec{d}$ identified in part 2a.

To do this, we define $s(\vec{x}) = \vec{d}^T \vec{im}(\vec{x})$ to be the component of the R, G, B responses at pixel $\vec{x}$ along the skin-coloured direction $\vec{d}$. Here $\vec{im}(\vec{x})$ is a 3-vector denoting the R, G, and B responses at pixel $\vec{x}$. The first restriction on skin-coloured pixels is that $s(\vec{x})$ must be bigger than some minimum value, say $s(\vec{x}) > m_s$, for some constant threshold $m_s > 0$ (otherwise, as we saw in Assignment 1, many dark pixels will be accepted as being skin-coloured). The second condition is that we need the radial distance to the cone's axis to be sufficiently small. That is, for a given RGB value $\vec{im}(\vec{x})$, define $r(\vec{x})$ to be the perpendicular distance between the axis of the cone and $\vec{im}(\vec{x})$, namely

$$r(\vec{x}) = ||\vec{im}(\vec{x}) - s(\vec{x})\vec{d}||.$$

Finally, for any pixel with $s(\vec{x}) > m_s$, define $skinIm(\vec{x}) = f(r(\vec{x})/s(\vec{x}))$. Here $f(z) \in [0, 1]$ is a non-increasing function of z for $z \geq 0$, with $f(0) = 1$ and $\lim_{z\to\infty} f(z) = 0$. Choose any suitable such function $f(z)$. For pixels which are too dark, that is $s(\vec{x}) \leq m_s$, set $skinIm(\vec{x}) = 0$.

After you provide the implementation for $skinIm(\vec{x})$ in the code `eyeFinderHandout.m`, the code will display your computed skin-colour response image. Adjust thresholds m_s and f_0 such that $skinIm(\vec{x}) > f_0$ selects most of the skin-coloured pixels, but as few of the pixels from the background as possible.

(c) **[2 pts] Skin Coloured Blobs**. We wish to remove small regions of isolated responses from the skin-coloured pixel response image $skinIm(\vec{x})$, and to fill in undetected pixels in skin-coloured regions. These properties will be important in Question 3 below, where we will only consider pixels identified to be within skin coloured blobs when we search for eyes. We therefore do not wish to miss too many pixels around the eyes! Also, to save computation time, we do not want to have too many extraneous pixels within these skin-coloured blobs.

To get the skin-coloured blob image, simply blur the thresholded image $skinIm(\vec{x}) > f_0$ by convolving it with a Gaussian filter, and threshold the response image, say $blurSkinIm(\vec{x}) > b_0$. (I found $\sigma = 7$ for the Gaussian filter worked well given the overall sizes of the faces in the current test set.) The code `eyeFinderHandout.m` will display your computed skin-colour blob image, $blurSkinIm(\vec{x})$. Adjust threshold b_0 such that $blurSkinIm(\vec{x}) > b_0$ selects almost all of the skin-coloured pixels, especially in the regions around the eyes.

(d) **[5 pts] Skin Coloured Blob Tracking**. We wish to rescale the image region around the face so that the inter-eye distance roughly matches that used to train the PCA model. To do that we first need to locate the face region in each of the images. Given the rough (x, y) image location of the face (as specified by `meanLoc` in `eyeFinderHandout.m`), the code already provided rescales and crops the region around this location. Here the rescaling value is simply taken to be a constant, and no rotation is used in the image warp. All you need to set is the center location of the face region, namely the value `meanLoc`.

Your task is to set the 2D vector `meanLoc` in `eyeFinderHandout.m` to provide the rough (x, y) image coordinates for face region. These coordinates do not need to be very accurate, so long as the face region ends up within the rescaled image. Use iteratively reweighted least squares to find the robust mean of the (x, y) pixel locations for which $blurSkinIm(\vec{x}) > b_0$.


Figure 1: Results of eigen-eye pair detector for the first and last frames. White regions indicate the detected eyes.

Given an appropriate choice for the weights, the algorithm should be able to track the face region through-out the sequence, using the location from the previous frame as the initial guess for the next frame. The initial guess for the first frame is already set to be the center of the image.

Once you have completed this modification of `eyeFinderHandout.m`, the revised code should be able to track the skin coloured blob corresponding to the face. If you didn't use loops over image pixels, the entire algorithm should run reasonably quickly (my implementation requires less than a second per frame).

In your report, provide a detailed description of this iteratively reweighted least squares algorithm to determine `meanLoc`, along with a specification for the particular distributions of weights you used.

5. **[10 pts] Eye Finder.** Here we apply the eigen-eye detector discussed above (or yours from question 3) to 20×25 subimages cut out of the warped subimage $wIm(\vec{x})$ produced in question 4 above. Check the handout file `eyeFinderHandout.m` for the locations you need to add code.

   First preprocess $wIm(\vec{x})$ by doing the local brightness division (see question 2 above), and overwrite $wIm(\vec{x})$ with the result. Use a loop (yes, a loop!) over pixels $\vec{x}_0 = (x_0, y_0)$ in the warped image at which a skin coloured blob was identified (i.e. $blurSkinIm(\vec{x}_0) > b_0$). For each such pixel $\vec{x}_0$, crop a 20×25 subimage from the normalized warped image wIm. In order to center the eye of the PCA model at this pixel $\vec{x}_0 = (x_0, y_0)$, use the ranges x `=` x_0 `+ rangeEyeX` and y `=` y_0 `+ rangeEyeY`, where `rangeEyeX` and `rangeEyeY` are provided in `eyeFinderHandout.m`. Store all these cropped subimages in one $500 \times P$ matrix of test patches, where P is the total number of patches. (This is the same format as the training and test data used in question 2 above.) You will also need to store the center position (x_0, y_0) for each of these P patches. (Option: Only sample every other pixel both horizontally and vertically within the skin coloured patch.)

   (a) **[7 pts] Single Eye Detector** Apply one of the eigen-eye detectors discussed above to this set of P cropped images. Note that this part of your Matlab code will be slow, perhaps half a minute a frame, depending on how large the skin coloured blob was in the warped frame and on n_B, the dimension of the basis used in the detector. Generate an overlay image (like the ones in Fig. 1) with the positions of the detected eyes marked in white.

Report on the overall performance of this detection scheme (include printed outputs of the eye detection overlays). In particular, how common are false positives? Also, in what situations do false negatives occur?

(b) **[3 pts] Eye Pair Detector** Finally, given the results of your individual eye detector in part 5a, check that there is a corresponding eye response roughly 40 pixels to either the right or left of it. This distance is the inter-eye separation used in the training images. (The same distance was also used in selecting the rough size of the rescaling factor (i.e. 0.65) for computing the warped image $wIm(\vec{x})$.) In order to check for the matching eye in a pair, allow for several pixels of error in both the vertical and horizontal directions.

Implement this verification step, and show only the pixels that were both identified as eyes according to step 5b and have a corresponding companion response in roughly the right location to either the left or right. Include printed copies of the overlayed detection results in your report (similar to Fig. 1). How common are false positives now? In what situations do false negatives occur?