CSC2503 – Computational Vision – Fall 2011

Assignment 4: Detecting Human Eyes

Due: 8:00 pm., Friday, December 2
This assignment is worth 15 marks towards your grade in this course.

This assignment uses a learning approach for view-based object detection. To get started, first download `A4Handout.zip` from the course web page.

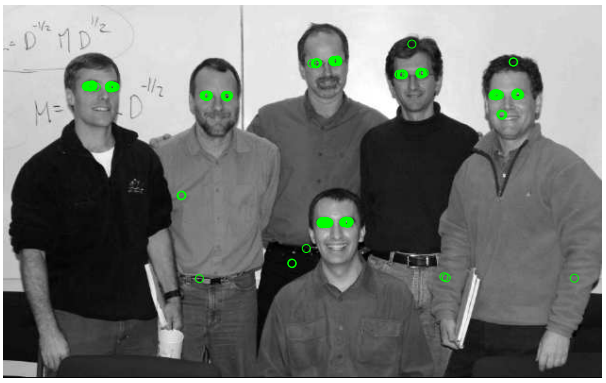

Figure 1: Results of an eye detector roughly similar to the detector constructed in this assignment. Green circles indicate detected eyes. Your results will vary.

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 programs in your report. Also, please email `odonovan@dgp.toronto.edu` a gzipped tar-file that contained all the Matlab files that you altered or created.

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 various eye detectors and, once the training is complete, we will use the latter to test their performance.

The images of eyes 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. There is also a set of non-eye images in the training and test sets, which are used to tune and test the detectors.

1. **Weak Classifiers.** We demonstrate an eye classifier built using the AdaBoost learning algorithm. This is a boosted classifier, based on many weak classifiers. Your task in this question is to complete `trainStump.m`, which sets some of the parameters for a weak classifier.

   The input for this M-function is a projection vector $\vec{f}$, a flag `useAbs`, an $N \times K$ dimensional data matrix X, a $1 \times K$ class label vector $\vec{y}$ (with y_k equal to 0 or 1, indicating the k^{th} column of X corresponds to a non-target or target, respectively), and a $1 \times K$ vector of non-negative weights

$\vec{w}$. We seek a "weak-classifier" of the form:

$$h(\vec{x}, \vec{\theta}) \;=\; \begin{cases} I(u(\vec{f}^T\vec{x}) \le \theta_1), & \text{for } \theta_2 = 1, \\ I(u(\vec{f}^T\vec{x}) > \theta_1), & \text{for } \theta_2 = -1, \end{cases} \tag{1}$$

where $u(z)$ is either just z or $|z|$, depending on whether or not the flag *useAbs* is false or true. Also, $I(b)$ in (1) is equal to 1 (one) when the boolean value b is true, and 0 otherwise. Note, this weak classifier $h(\vec{x}, \vec{\theta})$ is similar to the weak classifier $f(\vec{x}, \vec{\theta})$ on pp. 14-17 of the Object Recognition lecture notes, except there the classifier outputs values $+1$ or -1, while here it outputs $+1$ or 0.

The only unknowns for `trainStump.m` are the threshold value θ_1 and the parity θ_2. Complete the M-function `trainStump.m` so that it returns the values described in the function comment, including the optimum parameters $\vec{\theta}$. Denoting the k^{th} column of X by $\vec{x}_k$, the optimum $\vec{\theta}$ should minimize the weighted error

$$err \;=\; \frac{\sum_k w_k I(y_k \neq h(\vec{x}_k, \vec{\theta}))}{\sum_k w_k} \tag{2}$$

Hint: You only need to consider discrete values of θ_1, namely the values $u(\vec{f}^T\vec{x}_k)$ where $\vec{x}_k$ is a positive or negative training example. You can then find θ_1 and θ_2 in Matlab without writing your own loop by using the built-in functions `cumsum`, `sum`, `max`, and/or `min`.

Complete a short Matlab script, `testStump.m`, to demonstrate your M-function `trainStump.m`. In particular, build several weak classifiers based on projection vectors $\vec{f}$ obtained from derivatives of a Gaussian kernel (i.e., generated by `buildGaussFeat.m` in the `util` directory of the handout code). You can assume (for now) that the weights in $\vec{w}$ are all equal (although it is important that your implementation of `trainStump.m` works correctly for any non-negative weights with $\sum_k w_k \neq 0$). In your report show histograms of the continuous-valued function $u(\vec{f}^T\vec{x})$ for both target $\vec{x}$'s and non-targets. Report the true positive rate and the false positive rate of each of your weak classifiers using the optimal $\vec{\theta}$.

2. **AdaBoost.** Browse the script file `trainAdaGauss.m`. The next thing you need to do to complete the training part of this script is to write `evalBoosted.m` as described in the handout code's comments. Given a list of weak classifiers $\{h_m(\vec{x}, \vec{\theta}_m)\}_{m=1}^M$, which have been trained as in question 5 above and found to have errors err_m, for $m = 1, \ldots, M$, then the strong classifier is given by

$$S_M(\vec{x}) = \sum_{m=1}^{M} \alpha_m (2h_m(\vec{x}, \vec{\theta}_m) - 1), \tag{3}$$

here $\alpha_m = \log((1 - err_m)/err_m)$. (This expression is different from the one on p.16 of the lecture notes, since our weak classifiers have discrete values in $\{0, 1\}$ instead of $\{-1, 1\}$.)

The first part of the script file `trainAdaGauss.m` can then be run to train a boosted classifier for eyes. This iteratively adds one weak classifier to the additive linear model (3), trying many different weak classifiers to determine which one to add. This search over many weak classifiers may take a minute or so per classifier so you might try using only `nFeatures = 20` weak classifiers. If you have more CPU resources, try training `nFeatures = 50` or `100`.

As each weak classifier is selected, it is particularly instructive to observe the histograms of the feature before thresholding, that is, histograms of $u(\vec{f}^T\vec{x})$, for both the targets and the non targets. (The M-file `trainStump.m` in the handout code is set up to make these plots.) If there is an edge in most of the training target images at a particular scale, sign, and orientation, what might you expect a selected feature to be? If the sign of the edge varies (i.e., either light to dark

or dark to light), what might you expect a selected feature to be? If the training target images are all relatively smooth (i.e., no edges) in a particular region, what might you expect a selected feature to be?

Add code to the end of `trainAdaGauss.m` to plot the DET curves (see p.7 of the Object Recognition lecture notes) obtained by thresholding the strong classifier in (3), that is, $S_M(\vec{x}) > \tau$, for different values of τ. Draw separate DET curves for various values of M. Do this for both the training set of eyes and non-eyes, and the test set. To evaluate $S_M(\vec{x})$ for some M less than the maximum number of features you trained, you do not need to rerun the training. Why? Simply limit the sum in (3) to the first M terms. Comment on the results. Are the testing and training errors for a given strong classifier (i.e., the same M) similar? If not, explain what the cause for this might be.

3. **Application to an Image Patch.** The script file `tryEyeDetector.m` reads in an image of several people, none of whom appear in the previous training or test sets. Write Matlab code to select a rectangle from this image (using ginput(2)) and run an eye detector centered at every pixel within the selected rectangle. Show the resulting detections overlayed on top of the original image.

   The reason for not running the eye detector over the whole image is that these Matlab implementations are slow. You can write a relatively efficient Matlab implementation by storing the image as one long column vector I, and then forming $X = I(J)$, where J is a $500 \times K$ matrix of indices into I. If you choose the indices in J correctly, each column of X will be the pixels from a 20×25 patch within I. The detector can then be applied to all the columns in X quite efficiently. This trick won't work for the whole image, since you will run out of memory, but it will be able to process many patches at a time (i.e., K patches).

   Small Print. The eye detection results in Figure 1 were obtained using a similar positive set to the one used here, but with a much larger negative set. The main point of showing these results here is to demonstrate that a plausibly useful result can be obtained with the techniques described in this assignment.

4. **Synthetic Visual Cortex.** Suppose the first stage of our computational vision system always computed an image pyramid. And say this pyramid included Gaussian filtered images (say at σ equal to 4, 2, and 1), along with the first and second Gaussian derivatives (i.e., g_x, g_y, g_{xx}, g_{xy} and g_{yy}) at each of these scales. Completely specify the operations required to compute any of the weak classifiers (1) actually used in Question 3 above. (We'd like a written answer, you do not need to program anything.) Here you can assume you are given $\vec{n} = (\cos(\theta), \sin(\theta)$ corresponding to the steering orientation, θ, of any directional derivative filter required. Note, some of the filters used in `buildGaussFeat.m` were truncated to fit within the 20×25 image patch. You can ignore this spatial truncation in describing how the weak classifiers are computed from these pyramids. Also, roughly how many arithmetic operations are required to compute the strong classifier $S_M(\vec{x})$ above (ignoring the costs of forming and indexing into the image pyramids, and the effects of spatial filter truncation)?