STAT37710 — Homework 3

University of Chicago, Spring 2018 — due Monday, April 30 at 11:59pm

The files `faces.tar.gz` and `background.tar.gz` contain 2000 images of faces and non-face image patches, respectively. Your task in this assignment is to implement a Viola–Jones type boosting based detector cascade, as described in the lectures, to distinguish between these two classes, and hence detect faces in larger images. We provided a standard test image to test your algorithm. To detect the faces in the image, use a sliding window to evaluate your trained detector on each 64×64 patch. Report your results either by giving the coordinates of each patch where you detected a face, or, better yet, by drawing a square around the patch. You might find that you get many overlapping detections around each face. To avoid this problem use an exclusion rule to avoid overlaps. Please include a detailed writeup in your submission explaining any design choices that you have made, how many classifiers you have in your cascade, what the training error of each classifier is, and how long it took to train each stage of the cascade.

You have a lot of freedom in how exactly to implement the algorithm. However, here are a few tips:

1. We suggest that you convert the images to grayscale rather than worrying about color.
2. Use the Haar-like features described in the Viola–Jones paper and simple decision stumps (with varying threshold) as weak learners. Each round of boosting will select a single (i, p, θ) combination (where i indexes the feature, $p \in \{-1, +1\}$ is the polarity, and θ is the threshold) to add to your classifier.
3. The original Viola–Jones paper used three distinct types of Haar–like features. For the purposes of this assignment using just the simplest "two-rectangle" features is probably sufficient. However, for extra credit you might implement the other features as well.
4. The hypothesis returned by regular AdaBoost is $h(x) = \operatorname{sgn}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$. However, in a classifier cascade it is critical that each classifier have a low false negative rate. Therefore, we suggest that you instead use $h(x) = \operatorname{sgn}\big(\sum_{t=1}^{T} \alpha_t h_t(x) - \Theta\big)$, where Θ is set so that there are no missed detections (false negatives) on your training date at all.
5. A single booster with such an artificially low false negative rate will have a fairly high false positive rate. We suggest that you keep adding new features until the false positive rate falls below about 30%. You will probably find that less than a dozen rounds a boosting are sufficient for this. Chaining several such boosters together will push the false detection rate of the entire cascade down to acceptable levels.
6. One key idea in the Viola–Jones paper is that it is feasible to have a large number of base learners (Haar features) as long as the base learners are very fast to compute. These fetures should therefore always be computed on the fly (from the integral image representations) rather than computed once and then stored.
7. Speed might nonetheless be an issue in your implementation. We suggest that in the development stage you work with smaller data, and then scale up to the entire dataset on the final "production run". In addition, don't be afraid to (a) reduce the number of features by using a stride of size 2 or 4 as opposed to trying every Haar-like feature in every possible pixel location (b) reduce the training set, by training on just 1000 or even less images from each class. Try and write code that is relative efficient — that can also make a big difference.
8. Depending on the efficiency of your implementation you might want to start with smaller number of features than currently used in the provided code. One way to achieve that is to use Haar features of larger size (i.e. window size larger than 2×1 and 1×2) and/or larger strides.

Your code can be written in any language, but most students in past years used Python. We suggest that you precompute and store the following arrays as global variables:

- **iimages**: an $N_{\text{images}} \times 64^2$ array whose i'th rows stores the sum of all pixel values in image i in the rectangle extending from $(0,0)$ to (x,y).
- **featuretbl**: an $N_{\text{features}} \times 8$ array whose i'th row stores $(x_1, y_1), (x_1', y_1'), (x_2, y_2), (x_2', y_2')$ defining the two rectangles involved.

We suggest that your code define the following functions (this is of course a non-exhaustive list):

- **computeFeature(i,f)**: return the value of feature number f in image number i.
- **bestLearner(w)**: return the best weak learner, threshold and polarity over the training set for the current weight vector w.