ENEE 6582, CSCI 6633
Computer Vision
Project 5
Adaboost: Face Detection

Database

- CBCL MIT dataset: `http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html`
- 19 x 19 Grayscale PGM format images
- Training set: 2429 cropped faces; 4548 non-faces

Reading Images

All training images are 19x19, as opposed to 24x24 image size specified by J&V. This will make computing faster, but will affect the overall accuracy of the images. Luckily the training database used has cropped faces, unlike J&V database. You may use the following code to batch read many images

```
H=19; W=19;                          %size of training images
Path = "c:\path\faces\";             %directory where faces are located
fname = dir([Path, '*.pgm']);        %get file names. Data structure.
fnum = length(fname);                %number of files
Imgs = zeros(fnum, W*H);
for i=1:num
    im = im2double(imread([Path, fname(i).name]));
    Imgs(i,:) = im(:)';              %vectorize each image
end
```

It is a good idea to modify the code so that intensity of each image is normalized, by subtract the mean of each image and divide by its standard deviation. However, take care to see that standard deviation is not 0 (especially for non-faces).

It's also a good idea to compute the integral sum of each image using the function `cumsum()` before saving it in `Imgs`.

Creating Haar Features

There are 4 types of features: A. Horizontal , B. Vertical, C. 3-Horizontal, D. 4-Corners. You must create features that vary in size and location inside the 19x19 training image dimensions. The following code generates feature (x,y) coordinates (location) and (hx2w) size for the horizontal Haar feature:

```
fvec = [];                           %grow the feature vector matrix
W = 19; H=19;                        %dimensions of training images
% Horizontal (Type A)                %whitebar, blackbar horizontal
for w = 1:floor(W/2)-1               %w=1:8 => 2w=2:16
    for h = 1:H-2
        for x = 2: H-h                          %skip first rows
            for y = 2:W-2*w                     %skip first columns
                feat = zeros(H,W);              %initialize to 0
                feat(x:x+h-1,y:y+w-1)=1;        %white bar
                feat(x:x+h-1,y+w:y+2*w-1)=-1;   %black bar
                fvec=[fvec feat(:)'];           %vectorize feature
end;end;end;end
save fvec.mat fvec
```

Note that equation determining white bar and black bar will vary based on the feature type (A,B,C, or D). As specified by the J&V, the first and columns of the image are not used in calculating the integral sum. The width for a horizontal feature will vary from 2,4,6,8,…,16; while the height will vary from 1,2,3,…,17 (even picture sizes would have worked slightly better). Also, `feat` is 19x19 image which is reshaped into 391x1 vector for easy storage and subsequent multiplication. It's always better to define the feature vector matrix size instead of growing it as shown above. It's a good idea to save the features into memory when done so that you don't have to recomputed them every time.

The above code is for feature type A. The code can be modified for the calculation of features B, C, & D.

Computing Feature Integral Sum

To compute the integral sum for an image: multiply the vectorized feature matrix with transpose of the vectorized image integrals, e.g.: `Imgs' * fvec`. Repeat the calculation for the non-faces.

Adaboost

Number of learning iterations is comparable to the number of Haar features used.

```
H=19; W=19;
h = [posInt negInt]'*fvec;     %posInt=cumsum posdata; negInt=cumsum negdata
                               %posdata = faces; negdata = nonfaces;

pn = size(posInt,2);     %number of columns = num of pos images
nn = size(negInt,2);     %number of columns = num of neg images

alpha = zeros(T,1);            %T=number of features used
Ds = ones(pn+nn,1);            %init the probab distrib to 1 (normalize later)
Y  = [ones(pn,1) ones(nn,1)]; %Y=1 for face; Y=0 for nonfaces
```

The classification will be based on, `h`, the cumulative sum of (image.*haar feature). Each column of `h` represents a single Haar feature applied to all the images (rows). The number of face training samples and non-face training samples are stored in `pn` and `nn`. The vector alpha represents the weight of each classifier `h(i,j)`. Vector `Ds` is the probability associated with each image. Y is a binary vector with 1 and 0 representing face and non-face image (0 is used instead of -1 to facilitate and simplify the calculations).

Minimizing Error

We wish to learn a threshold for the cumulative sum that produces a classification error less than chance (50%).

Assume:
h(:,i)' = [**4 5 6 7** 8 9 1 2 3 0];
y' = [**1 1 1 1** 0 0 0 0 0 0];
D' = [**1 1 1 1** 1 1 1 1 1 1]/10.

In the above example there are only 10 images in the database (10 rows). (h(:,i) = cumulative sum of (each image x i^{th} Haar function). Note that the very first 4 images are face (y=1) and the last 6 are non-face images (y=0). Initially the probability of each image is D = 1/10.

We wish to pick a threshold that minimizes the error. If we pick $\theta \leq 4$, then the images considered faces are highlighted in blue: h = [**4 5 6 0** 7 8 9 1 2 3]. The error = [**0 1 1 1** 0 0 0 1 1 1].*D = 0.6.
If $\theta \leq 5$, then h = [**4 5 6 7** 8 9 1 2 3 0], e = [**0 0 1 1** 0 0 1 1 1 1].*D = 0.6
If $\theta \leq 6$, then h = [**4 5 6 7** 8 9 1 2 3 0], e = [**0 0 0 1** 0 0 1 1 1 1].*D = 0.5
If $\theta \leq 7$, then h = [**4 5 6 7** 8 9 1 2 3 0], e = [**0 0 0 0** 0 0 1 1 1 1].*D = 0.4

From the above we can see that the threshold that minimizes the error is 7.

Note that if search for the threshold is iterative, as shown above, the algorithm will be very slow. Think of ways to speed-up the search for the threshold without using a for-loops. (Hint: sort the numbers in ascending order, then apply cumsum on indexes belonging to y=1 and y=0.)

Updating D

With the error, ε_t, the weights of classifiers are: $\alpha_t = \frac{1}{2} Ln \frac{1-\varepsilon_t}{\varepsilon_t}$, and the updated probabilities are:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x))}{Z_t}$$

where, Z_t is just the sum (D_t). The iterations, T, will be equal to the number of features used.

Analysis

1. Number of Haar Features: Identify 200 of the most successful Haar features in detecting faces and rejecting non-faces. Of these identify 10 of the most successful Haar features.

   The classification of an image as a face is determined by

   $$\sum_{\tau=1}^{T} \alpha_\tau h_\tau \geq 0.5$$

   Calculate the detection error rate as starting with 1 Haar features until all the top 200 Haar features are simultaneously. Plot the error rate vs number of Haar features. Repeat the plot but for the rejection rate.

2. Effect of scale: Take a number of graylevel test images (face and non-face) and scale them down into the following different sizes: 19x19, 24x24, 29x29, 36x36, 45x45, 56x56, 70x70, 88x88, 110x110. The images don't have to be many. Note that the each size is 1.25x the previous size rounded to the nearest integer. The images don't have to be many, e.g. 5 face images and 5 non-face images.

   You may use the Matlab function imresize() in resize the images. Always start with cropped square images that are greater than 110x110. And always scale down from the original cropped image.

   Scale the Haar features by the same scale (you can use imresize() but ensure that nearest neighbor is used for interpolating the data). Without changing the weights test the effectiveness of the classifier in detecting the faces and rejecting the non-faces.

   The goal is to study if the weights are adequate under different scales. If the weights are not adequate determine how should they be scaled for different scales.