HW8: Object Detection via Viola and Jones

J. Zapf[†]

I. Problem Statement

In this homework, we consider the problem of object detection. That is, given an image, we wish to identify and locate instances of a general object class within the image. Specifically, we are interested in detecting cars. A complete object detection system consists of a sliding window and a classifier. As the sliding window scans the image, the classifier makes a decision regarding whether or not the portion of the image occupied by the sliding window is an instance of a particular class or not. Obviously, such a classifier must have a very low false positive rate as the number of negative samples will drastically outnumber the positive samples.

The complete object detection problem is not considered in this homework. Only the classification portion of the problem is treated. Following the example of the Viola and Jones face detector, we implement an Adaboost classifier with Haar-like features. Such a classifier consists of a weighted sum of many weak classifiers. To obtain a sufficiently low false positive rate, we cascade several Adaboost classifiers, where at each stage of the cascade only the samples classified as positive are allowed to pass through. Thus, increasingly difficult false positives are pruned away at each stage.

II. Computing Haar-like Features

The image patches considered in this homework are of size 20×40 pixels. From these image patches, we extract a subset of Haar-like features, which are computed using vertical and horizontal differencing operators of the form shown in Figure 1. Considering all possible template sizes and all possible locations within the image patch, we see that there are

$$(39 + 37 + \cdots + 1)(20 + 19 + \cdots + 1) = 84000 \tag{1}$$

[†] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907

possible vertical features for each image and

$$(19 + 17 + \cdots + 1)(40 + 39 + \cdots + 1) = 82000 \tag{2}$$

possible horizontal features for each image. Thus the total number of possible features is 166000. These features can be efficiently computed by first computing the integral image

$$S(i, j) = \sum_{x<i} \sum_{y<j} I(x, y). \tag{3}$$

Note that S should start with a row and column of zeros and thus should be 21×41. Each Haar-like feature can then be expressed as the wighted sum of 6 entries in S of the form

$$f = -S(x_1, y_1) + S(x_2, y_2) + 2S(x_3, y_3) - 2S(x_4, y_4) - S(x_5, y_5) + S(x_6, y_6), \tag{4}$$

with the locations of coordinates 1-6 for vertical and horizontal features shown Figure 2.

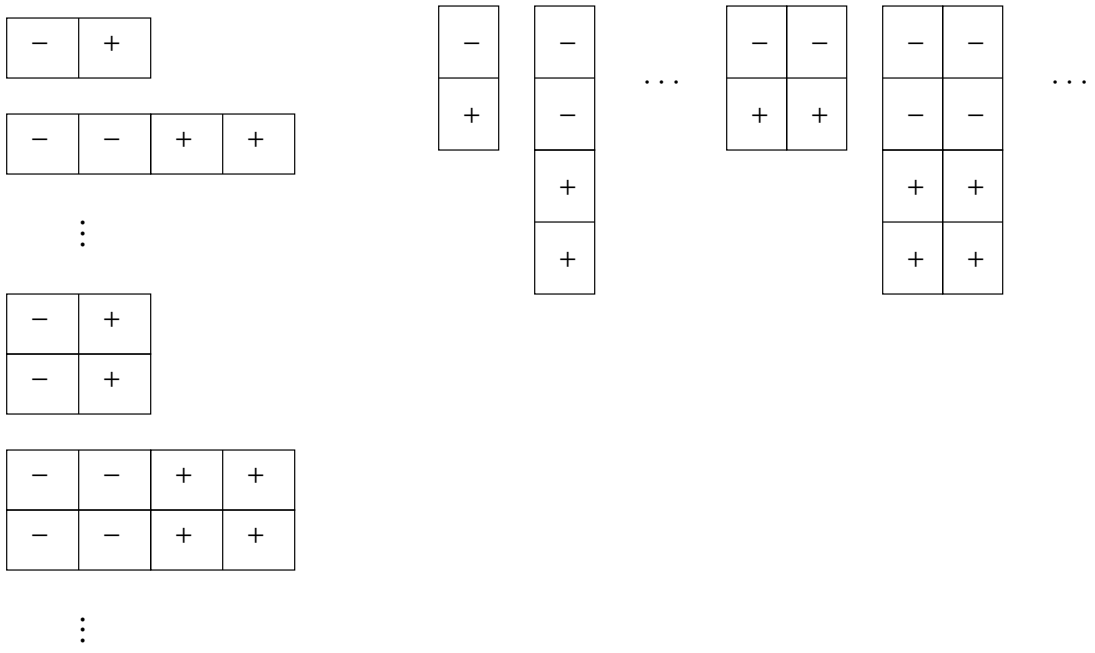

Fig. 1. Haar-like features.

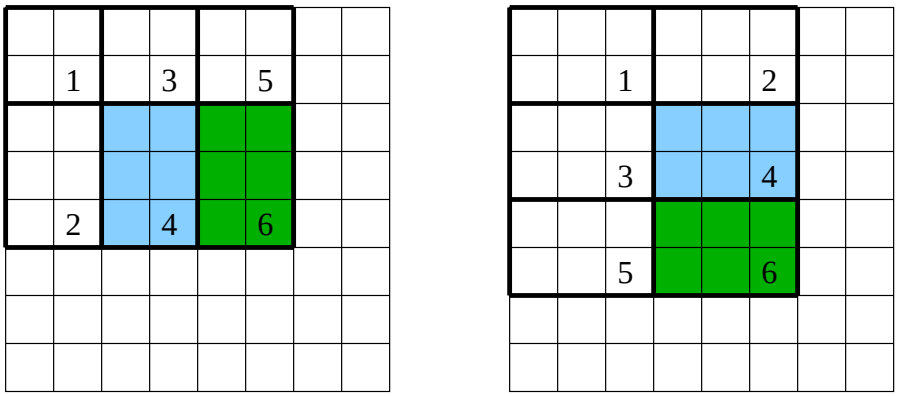

Fig. 2. Feature extraction via the integral image.

III. Training the Classifier

A single Adaboost classifier consists of a wighted sum of many weak classifiers, where each weak classifier is a threshold on a single feature. These features, i.e. weak classifiers, are selected one at a time according to how well they separate the data. The weak classifiers are trained on wighted data, meaning that not all of the data samples are treated the same. The weight associated with a given sample is adjusted based on whether or not the weak classifier correctly classifies the sample or not. Specifically, each Adaboost classifier is trained using a slight variant of the procedure listed in Table 1 on page 142 of the Viola and Jones paper. The procedure used in this homework is as follows.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$, where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize the classifier count $t = 0$ and the sample weights $w_i = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive samples.
- While the number of negative samples pruned is less than 50%:
  1) Increment $t = t + 1$.
  2) Normalize the weights, $w_i = \frac{w_i}{\sum_j w_j}$.
  3) Select the best weak classifier with respect to the weighted error

$$\epsilon_t = \min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|. \tag{5}$$

  4) Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.

5) Update the weights as

$$w_i = w_i \beta_t^{1-e_i}, \tag{6}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

6) Evaluate the strong classifier

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \gamma_t \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where $\alpha_t = \log \frac{1}{\beta_t}$ and γ_t is chosen such that all positive training samples are correctly classified.

In the above training procedure, the weak classifier $h(x, f, p, \theta)$ is defined as

$$h(x, f, p, \theta) = \begin{cases} 1 & pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

where f denotes the feature value, θ is the threshold, and p is the polarity indicating the direction of the inequality. Viola and Jones provide an efficient method for computing ϵ_t. To summarize, the minimum error can be found by searching over every possible feature for every training sample. For a given feature, however, we only need to pass through a sorted list of the training images once to find the optimal threshold. This is accomplished by maintaining four sums: the total sum of positive example wights T^+, the total sum of negative example weights T^-, the sum of positive weights below the current example S^+, and the sum of negative weights below the current example S^-. The error for a threshold which splits the range between the current and previous example in the sorted list is then

$$e = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)). \tag{9}$$

Note that the first error in the $\min$ function is the error associated with labeling all examples below the current example negative and labeling the examples above positive. In this case, it can be shown that the polarity of the weak classifier should be $p = -1$.

To decrease the false positive rate, we cascade several Adaboost classifiers together. In the cascaded approach, a sample is classified as positive only if every classifier in the cascade classifies it as positive. In terms of training, this implies that for each sequential Adaboost

classifier we only train on those samples that were classified as positive in all preceding stages. For this homework, we continued to add cascade stages until all of the samples were correctly classified.

IV. Results

Applying the above training procedure to the training data supplied on the course web page, a 9 stage classifier was learned. Printed below is a MATLAB print out displaying the number of negative samples pruned upon adding weak classifiers at each stage of the cascade.

```
>> AdaClass
cascade stage 1
negative samples pruned = 0 0 0 0 609 609 594 599 760 841 1118
cascade stage 2
negative samples pruned = 0 0 0 0 0 92 141 138 239 238 263 318 315 240 239 271
298 322
cascade stage 3
negative samples pruned = 0 0 0 0 0 0 0 1 41 51 44 43 46 42 66 86 81 133 130 158
141 127 140 127 145 109 130 175
cascade stage 4
negative samples pruned = 0 0 0 0 2 2 14 14 24 12 32 22 39 25 44 38 42 32 38 52
76
cascade stage 5
negative samples pruned = 0 0 0 0 0 0 0 0 1 4 1 13 17 19 33 41
cascade stage 6
negative samples pruned = 0 0 0 0 0 0 1 2 8 11 14
cascade stage 7
negative samples pruned = 0 0 0 0 0 3 3 6
cascade stage 8
negative samples pruned = 0 0 2 2 5
cascade stage 9
negative samples pruned = 0 1
>>
```

Once the classifer was learned, we then applied it to the testing data set provided on the course web page. The false positive and false negative rates after the first k stages of the cascade were calculated as follows:

$$fp = \frac{\text{\# of misclassified negative test image}}{\text{\# of negative test image}} \tag{10}$$

$$fn = \frac{\text{\# of misclassified positive test image}}{\text{\# of positive test image}}. \tag{11}$$

Figure 3 depicts the evolution of these rates on the testing data set.

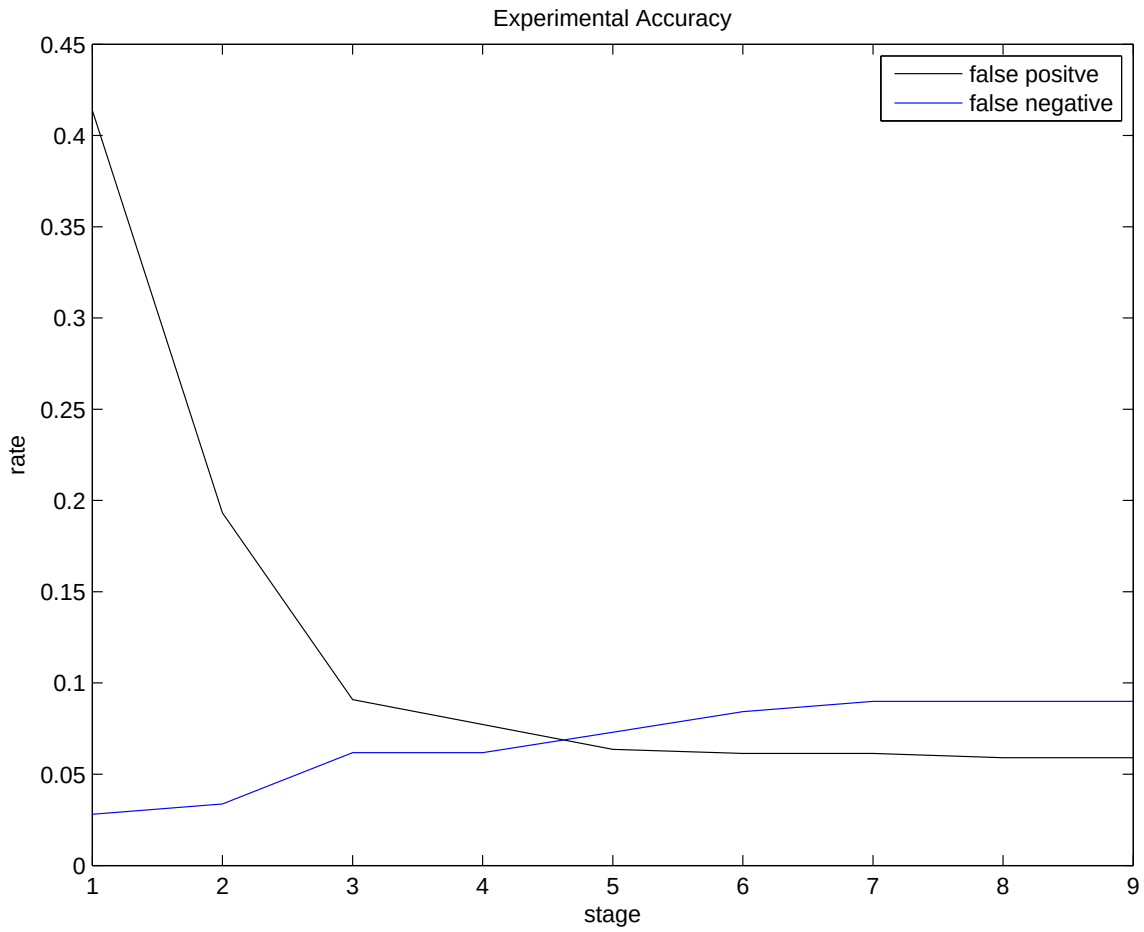

Fig. 3. Classification accuracy at each stage of the cascade.

V. MATLAB Code for Feature Extraction

```
%-------------------------------------------------------------------------
% GetFeat
%-------------------------------------------------------------------------
function GetFeat

clear all;

% filename = sprintf('images/train/positive/');
% ExtFeat(710,filename,'positive.mat',0);
% filename = sprintf('images/train/negative/');
% ExtFeat(879,filename,'negativeA.mat',0);
% ExtFeat(879,filename,'negativeB.mat',879);

filename = sprintf('images/test/positive/');
ExtFeat(178,filename,'positive_test.mat',710);
```

```
filename = sprintf('images/test/negative/');
ExtFeat(440,filename,'negative_test.mat',1758);

%--------------------------------------------------------------------------
% ExtFeat
%--------------------------------------------------------------------------
function ExtFeat(nImg,inFile,outFile,offset)

%>>> initialize the data matrices <<<%

T = zeros(20,1);
S = zeros(21,41);
f = zeros(166000,nImg);

%>>> display the progress <<<%

fprintf('%d training images\n', nImg);
fprintf('images processed =');

%>>> perform for each image in the training set <<<%

for k = 1:nImg

    %>>> load the training image <<<%

    imageName = sprintf('%s%06d.png',inFile,offset+k);
    image = imread(imageName);
    image = rgb2gray(image);

    %>>> compute the integral image <<<%

    for i = 2:21
        for j = 2:41
            T(1:i-1) = sum(image(1:i-1,1:j-1),2);
            S(i,j) = sum(T(1:i-1),1);
        end
    end

    %>>> extract the vertical Haar-like features <<<%

    cnt = 0;
    for h = 1:20
        for w = 1:20
            for i = 1:21-h
                for j = 1:41-2*w

                    x1 = j;
                    x2 = j;
                    x3 = j+w;
                    x4 = j+w;
                    x5 = j+2*w;
                    x6 = j+2*w;
```

```
                    y1 = i;
                    y3 = i;
                    y5 = i;
                    y2 = i+h;
                    y4 = i+h;
                    y6 = i+h;

                    cnt = cnt+1;
                    f(cnt,k) = -S(y1,x1)+S(y2,x2)+2*S(y3,x3)-...
                        2*S(y4,x4)-S(y5,x5)+S(y6,x6);

                end
            end
        end
    end

    %>>> extract the horizontal Haar-like features <<<%

    for h = 1:10
        for w = 1:40
            for i = 1:21-2*h
                for j = 1:41-w

                    x1 = j;
                    x3 = j;
                    x5 = j;
                    x2 = j+w;
                    x4 = j+w;
                    x6 = j+w;

                    y1 = i;
                    y2 = i;
                    y3 = i+h;
                    y4 = i+h;
                    y5 = i+2*h;
                    y6 = i+2*h;

                    cnt = cnt+1;
                    f(cnt,k) = -S(y1,x1)+S(y2,x2)+2*S(y3,x3)-...
                        2*S(y4,x4)-S(y5,x5)+S(y6,x6);

                end
            end
        end
    end

    %>>> display the progress <<<%

    if mod(k,50) == 0
        fprintf(' %d', k);
    end

end
```

```
fprintf(' %d\n', nImg);
fprintf('\n');

%>>> save features to a MAT file <<<%

save(outFile,'f','-mat','-v7.3');
```

VI. MATLAB Code for Feature Grouping

```
%--------------------------------------------------------------------------
% GroupFeat
%--------------------------------------------------------------------------

clear all;

% nPos = 710;
% nNeg = 879;
% nTot = nPos+2*nNeg;
nPos = 178;
nNeg = 440;
nTot = nPos+nNeg;
nFts = 166000;
nSet = 5000;
nGrp = ceil(nFts/nSet);

fprintf('number of groups = %d\n', nGrp);
fprintf('groups processed =');

feat = zeros(nSet,nTot);
for i = 1:floor(nFts/nSet)

    k0 = nSet*(i-1)+1;
    kf = nSet*i;

    % load('positive.mat','-mat','f');
    load('positive_test.mat','-mat','f');
    feat(:,1:nPos) = f(k0:kf,:);
    clear f;

    % load('negativeA.mat','-mat','f');
    load('negative_test.mat','-mat','f');
    feat(:,nPos+1:nPos+nNeg) = f(k0:kf,:);
    clear f;

    % load('negativeB.mat','-mat','f');
    % feat(:,nPos+nNeg+1:nTot) = f(k0:kf,:);
    % clear f;

    filename = sprintf('FEAT%02d.mat',i);
    save(filename,'feat','-mat','-v7.3');
```

```
    fprintf(' %d', i);

end

clear feat;
feat = zeros(mod(nFts,nSet),nTot);

k0 = nSet*floor(nFts/nSet)+1;
kf = nFts;

% load('positive.mat','-mat','f');
load('positive_test.mat','-mat','f');
feat(:,1:nPos) = f(k0:kf,:);
clear f;

% load('negativeA.mat','-mat','f');
load('negative_test.mat','-mat','f');
feat(:,nPos+1:nPos+nNeg) = f(k0:kf,:);
clear f;

% load('negativeB.mat','-mat','f');
% feat(:,nPos+nNeg+1:nTot) = f(k0:kf,:);
% clear f;

filename = sprintf('FEAT%02d.mat',nGrp);
save(filename,'feat','-mat','-v7.3');

fprintf(' %d\n', nGrp);
fprintf('\n');
```

VII. MATLAB Code for Creating the Classifier

```
%--------------------------------------------------------------------------
% AdaClass
%--------------------------------------------------------------------------

nPos = 710;                     % number of positive samples
nNeg = 879;                     % 1/2 number of negative samples
nTot = nPos+2*nNeg;             % total number of samples
nFts = 166000;                  % number of features
nSet = 5000;                    % number of features in a group
nGrp = ceil(nFts/nSet);         % nunmber of groups
%nGrp = 1;

nWClass = 50;       % maximum number of weak classifiers
nStages = 20;       % maximum number of stages in the cascade

%--------------------------------------------------------------------
%  initialize the cascaded classifier
%--------------------------------------------------------------------
```

```
h = zeros(3*nStages,nWClass);
a = zeros(nStages,nWClass);
s = zeros(1,nStages);
t = zeros(1,nStages);

accuracy = zeros(4,nStages);
positive = ones(1,nTot);
feat = zeros(nSet,nTot);
f = zeros(nWClass,nTot);

%-------------------------------------------------------------------
%  add stages until 100% accuracy obtained
%-------------------------------------------------------------------
stage_cnt = 0;
false_pos = inf;
false_neg = inf;
while false_pos+false_neg > 0

    %---------------------------------------------------------------
    %  increment cascade stage count
    %---------------------------------------------------------------
    stage_cnt = stage_cnt+1;

    %---------------------------------------------------------------
    %  initialize the weights
    %---------------------------------------------------------------
    mPos = sum(positive(1:nPos));
    mNeg = sum(positive(nPos+1:nTot));
    w = [ones(1,nPos)/(2*mPos),ones(1,2*nNeg)/(2*mNeg)];
    fprintf('cascade stage %d\n', stage_cnt);
    fprintf('negative samples pruned =');

    %---------------------------------------------------------------
    %  add weak classifiers until 50% of negative samples pruned
    %---------------------------------------------------------------
    cnt = 0;
    negCnt = 0;
    while negCnt < mNeg/2

        %-----------------------------------------------------------
        %  increment weak classifier count
        %-----------------------------------------------------------
        cnt = cnt+1;

        %-----------------------------------------------------------
        %  normalize the weights
        %-----------------------------------------------------------
        w = w/sum(positive.*w);

        %-----------------------------------------------------------
        %  search for the best weak classifier
        %-----------------------------------------------------------
        eMin = inf;
```

```
TPos = sum(positive(1:nPos).*w(1:nPos));
TNeg = sum(positive(nPos+1:nTot).*w(nPos+1:nTot));

for i = 1:nGrp              % for each group of features

    filename = sprintf('FEAT%02d.mat',i);
    load(filename,'-mat','feat');

    [sfeat,ifeat] = sort(feat,2);
    nFeats = size(feat,1);
    fNew = 0;

    for j = 1:nFeats        % for each feature in the group

        SPos = 0;
        SNeg = 0;

        for k = 1:nTot      % for each training sample

            if positive(ifeat(j,k)) ~= 1
                continue;
            end

            e1 = SPos+(TNeg-SNeg);
            e2 = SNeg+(TPos-SPos);

            if e2 < e1
                eTmp = e2;
                pTmp = 1;
            else
                eTmp = e1;
                pTmp = -1;
            end

            if eTmp < eMin
                fNew = 1;
                eMin = eTmp;
                h(3*(stage_cnt-1)+1,cnt) = nSet*(i-1)+j;
                h(3*(stage_cnt-1)+2,cnt) = pTmp;
                h(3*(stage_cnt-1)+3,cnt) = sfeat(j,k);
            end

            if ifeat(j,k) > nPos
                SNeg = SNeg+w(ifeat(j,k));
            else
                SPos = SPos+w(ifeat(j,k));
            end

        end

    end

    if fNew == 1
```

```
        jFeat = h(3*(stage_cnt-1)+1,cnt)-nSet*(i-1);
        f(cnt,:) = feat(jFeat,:);
    end

end

%-------------------------------------------------------------
%  update the weights
%-------------------------------------------------------------
beta = eMin/(1-eMin);

for i = 1:nPos
    if positive(i) ~= 1
        continue;
    end
    p = h(3*(stage_cnt-1)+2,cnt);
    theta = h(3*(stage_cnt-1)+3,cnt);
    if p*f(cnt,i) < p*theta
        w(i) = w(i)*beta;
    end
end

for i = nPos+1:nTot
    if positive(i) ~= 1
        continue;
    end
    p = h(3*(stage_cnt-1)+2,cnt);
    theta = h(3*(stage_cnt-1)+3,cnt);
    if ~(p*f(cnt,i) < p*theta)
        w(i) = w(i)*beta;
    end
end

%-------------------------------------------------------------
%  set threshold for strong classifier
%-------------------------------------------------------------
a(stage_cnt,cnt) = log(1/beta);

HSMin = inf;
for i = 1:nPos
    if positive(i) ~= 1
        continue;
    end
    HS = 0;
    for j = 1:cnt
        p = h(3*(stage_cnt-1)+2,j);
        theta = h(3*(stage_cnt-1)+3,j);
        if p*f(j,i) < p*theta
            HS = HS+a(stage_cnt,j);
        end
    end
    if HS < HSMin
        HSMin = HS;
```

```
        end
    end

    %-------------------------------------------------------------
    %  test strong classifier on negative samples
    %-------------------------------------------------------------
    negCnt = 0;
    for i = nPos+1:nTot
        if positive(i) ~= 1
            continue;
        end
        HS = 0;
        for j = 1:cnt
            p = h(3*(stage_cnt-1)+2,j);
            theta = h(3*(stage_cnt-1)+3,j);
            if p*f(j,i) < p*theta
                HS = HS+a(stage_cnt,j);
            end
        end
        if HS < HSMin
            negCnt = negCnt+1;
        end
    end

    %-------------------------------------------------------------
    %  display progress
    %-------------------------------------------------------------

    fprintf(' %d',negCnt);

end

fprintf('\n');

%-----------------------------------------------------------------
%  save the weak class count and the strong class threshold
%-----------------------------------------------------------------
s(stage_cnt) = cnt;
t(stage_cnt) = HSMin;

%-----------------------------------------------------------------
%  evaluate on samples classified as positive
%-----------------------------------------------------------------
false_neg = 0;
for i = 1:nPos
    if positive(i) ~= 1
        continue;
    end
    HS = 0;
    for j = 1:cnt
        p = h(3*(stage_cnt-1)+2,j);
        theta = h(3*(stage_cnt-1)+3,j);
        if p*f(j,i) < p*theta
```

```
                HS = HS+a(stage_cnt,j);
            end
        end
        if HS >= HSMin
            positive(i) = 1;
        else
            false_neg = false_neg+1;
            positive(i) = 0;
        end
    end

    false_pos = 0;
    for i = nPos+1:nTot
        if positive(i) ~= 1
            continue;
        end
        HS = 0;
        for j = 1:cnt
            p = h(3*(stage_cnt-1)+2,j);
            theta = h(3*(stage_cnt-1)+3,j);
            if p*f(j,i) < p*theta
                HS = HS+a(stage_cnt,j);
            end
        end
        if HS >= HSMin
            false_pos = false_pos+1;
            positive(i) = 1;
        else
            positive(i) = 0;
        end
    end

    accuracy(1,cnt) = false_pos;
    accuracy(2,cnt) = mPos;
    accuracy(3,cnt) = false_neg;
    accuracy(4,cnt) = mNeg;

end

%-------------------------------------------------------------------
%  save the results to a MAT file
%-------------------------------------------------------------------
save('results.mat','h','a','s','t','accuracy','-mat','-v7.3');
```

VIII. MATLAB Code for Testing the Classifier

```
%-------------------------------------------------------------------------
% AdaTest
%-------------------------------------------------------------------------

nPos = 178;                    % number of positive samples
```

```
nNeg = 440;                     % 1/2 number of negative samples
nTot = nPos+nNeg;               % total number of samples
nFts = 166000;                  % number of features
nSet = 5000;                    % number of features in a group
nGrp = ceil(nFts/nSet);         % nunmber of groups

%-------------------------------------------------------------------
%  load results from the training stage
%-------------------------------------------------------------------
load('results.mat','-mat','h','a','s','t','accuracy');

%-------------------------------------------------------------------
%  concatenate feature indices for all weak classifiers
%-------------------------------------------------------------------
nStages = find(s == 0,1)-1;
FeatInd = zeros(1,sum(s(1:nStages)));

for i = 1:nStages
    k0 = sum(s(1:i-1))+1;
    kf = sum(s(1:i));
    FeatInd(1,k0:kf) = h(3*(i-1)+1,1:s(i));
end

%-------------------------------------------------------------------
%  stack test feature vectors for all weak classifiers
%-------------------------------------------------------------------
[sFeatInd,iFeatInd] = sort(FeatInd,2);
f = zeros(sum(s(1:nStages)),nTot);

for i = 1:nGrp

    filename = sprintf('FEAT%02d.mat',i);
    load(filename,'-mat','feat');

    k0 = find(sFeatInd > (i-1)*nSet, 1, 'first');
    kf = find(sFeatInd < i*nSet+1, 1, 'last');

    for j = k0:kf
        f(iFeatInd(j),:) = feat(mod(sFeatInd(j),nSet),:);
    end

end

%-------------------------------------------------------------------
%  evaluate the classifier on the test data set
%-------------------------------------------------------------------
positive = ones(1,nTot);
accuracy = zeros(4,nStages);

false_neg = 0;
for i = 1:nStages           % for each stage of the cascade

    accuracy(2,i) = sum(positive(1:nPos));
```

```
accuracy(4,i) = sum(positive(nPos+1:nTot));

for j = 1:nPos          % for each positve test sample

    if positive(j) ~= 1
        continue;
    end

    HS = 0;
    for k = 1:s(i)      % for each weak classifier

        p = h(3*(i-1)+2,k);
        theta = h(3*(i-1)+3,k);
        if p*f(sum(s(1:i-1))+k,j) < p*theta
            HS = HS+a(i,k);
        end

    end

    if HS >= t(i)
        positive(j) = 1;
    else
        false_neg = false_neg+1;
        positive(j) = 0;
    end

end

false_pos = 0;
for j = nPos+1:nTot     % for each negative test sample

    if positive(j) ~= 1
        continue;
    end

    HS = 0;
    for k = 1:s(i)      % for each weak classifier

        p = h(3*(i-1)+2,k);
        theta = h(3*(i-1)+3,k);

        if p*f(sum(s(1:i-1))+k,j) < p*theta
            HS = HS+a(i,k);
        end

    end

    if HS >= t(i)
        false_pos = false_pos+1;
        positive(j) = 1;
    else
        positive(j) = 0;
    end
```

```
    end

    accuracy(1,i) = false_pos;
    accuracy(3,i) = false_neg;

end

%--------------------------------------------------------------------
%  plot the results
%--------------------------------------------------------------------
figure(1)
plot(1:nStages,accuracy(1,:)/nNeg,'-k')
hold on
plot(1:nStages,accuracy(3,:)/nPos,'-b')
hold off
title('Experimental Accuracy')
ylabel('rate')
xlabel('stage')
legend('false positve','false negative','location','northeast');

print -depsc 'accuracy.eps'
```