COMPUTER VISION

ECE 661

HOMEWORK 11

Tanzeel U. Rehman

Email: *trehman@purdue.edu*

1. Face Recognition:

1.1. Description of Methods:

A. Principal Component Analysis (PCA):

In this homework, we used PCA to reduce the dimensionality of the face data provided as part of the homework. MATLAB was used as programming language for this homework. To compute the principal components, images having a size of 128 ×128 pixels were converted into grayscale. These grayscale images were vectorized ($v_i = 16348 \times 1$) and were normalized (x_i) as shown in Eq. 1. The mean of all the images (*N*) in training set were then calculated as per Eq. 2.

$$x_i = \frac{v_i}{||v||} \tag{Eq.1}$$

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{Eq.2}$$

A matrix X was then calculated by subtracting the mean from each normalized image vector as given in Eq. 3. Instead of covariance matrix ($C = \frac{1}{N}XX^T$) , we computed the eigenvectors (*ti*) of the matrix XX^T, which were then used for computing eigenvectors (*ui*) of covariance matrix as shown in Eq. 4. The eigenvectors were normalized and were arranged in the descending order of their eigen values. Now we select the eigenvectors corresponding to *p* largest eigenvalues of the normalized *ui*. The matrix containing the dimensionally reduced principal components can be given Eq. 5. The feature vector (y_i) corresponding to training or test image was then computed by projecting the vectors in *X* matrix onto this subspace as shown in Eq. 6. These features can then be used to train a K-Nearest Neighbor classifier with K =1 using training features.

$$X = [x_1 - m, x_2 - m, x_3 - m, \ldots, x_N - m] \tag{Eq.3}$$

$$u_i = Xt_i \tag{Eq.4}$$

$$W_p = [w_1, w_2, w_3, \dots, w_p] \quad \text{(Eq.5)}$$

$$y_i = {W_p}^T (x_i - m) \quad \text{(Eq.6)}$$

B. Linear Discriminant Analysis (LDA):

The LDA tries to find the eigenvectors by minimizing the Fisher Discriminant function given by Eq. 7.

$$J(w_i) = \frac{{w_j}^T S_B w_j}{{w_j}^T S_W w_j} \quad \text{(Eq.7)}$$

Where S_B and S_W are the between-class and within-class scatters. In order to make sure that the S_W is not singular, the Yu and Yang's algorithm was applied. The images were first converted to the grayscale and were later vectorized and normalized as mentioned in previous section (Eq. 1). The mean of all the images in the training set was computed as per Eq. 2. The class means of images ($||C_k||$=number of images in *kth* class) in the *K* individual classes were calculated using Eq. 8. The computed means were then used to form Matrix *M* as given by Eq. 9. Instead of between-class ($S_B = \frac{1}{C} MM^T$) , we computed the eigenvectors (*ti*) of the matrix MM^T, which were then used for computing eigenvectors (*ui*) of S_B matrix as shown in Eq. 10. The eigenvectors were then normalized to obtain *Vi* as given by Eq. 11. Now the matrix Y ($Y = [V_1, V_2, \dots]$) was formed and the diagonal eigen value (D_B) of matrix S_B was used to compute the *Z* vectors as per Eq. 12. Now, we computed the eigenvectors $Z^T SwZ$ by rewriting it as Eq. 13. If U represents the normalized eigenvectors of $Z^T SwZ$, we sorted them in ascending order and selected the eigenvectors corresponding to *p* smallest eigenvalues. Now the projection vector (W_p) was created using the U_p as shown in Eq. 14. The normalized W_p was then used to compute the features for training and test set by projecting the respective images as shown in Eq. 15. These features can then be used to train a K-Nearest Neighbor classifier with K =1 using training features.

$$m_k = \frac{1}{||C_k||} \sum_{i=1}^{i=||C_k||} x_i \quad \text{(Eq.8)}$$

$$X = [m_1 - m, m_2 - m, m_3 - m, \dots, x_C - m] \quad \text{(Eq.9)}$$

$$u_i = Mt_i \quad \text{(Eq.10)}$$

$$V_i = \frac{u_i}{||u||} \quad \text{(Eq.11)}$$

$$Z = YD_B^{-1/2} \quad \text{(Eq.12)}$$

$$Z^T SwZ = (Z^T X)(Z^T \mathrm{X})^T \quad \text{(Eq.13)}$$

Where,

$$X = [x_{11} - m_1, x_{21} - m_2, x_{31} - m_3, \dots, x_{C1} - m_C, \dots x_{Ck} - m_C]$$

$$W_p = ZU_p \quad \text{(Eq.14)}$$

$$y_i = {W_p}^T (x_i - m) \quad \text{(Eq.15)}$$

1.2. Results:

The accuracy of the KNN classifier as a function of number of PCA and LDA reduced features was shown in Fig. 1 for the test dataset. For the latent space dimensions less than 2, the PCA performed better. Both dimensionally reduction techniques resulted in 100% accuracy eventually with LDA reaching earlier than PCA. LDA achieved 100% accuracy with latent space having dimensions of 7, while PCA reached 100% accuracy with latent space having dimensions of 13.

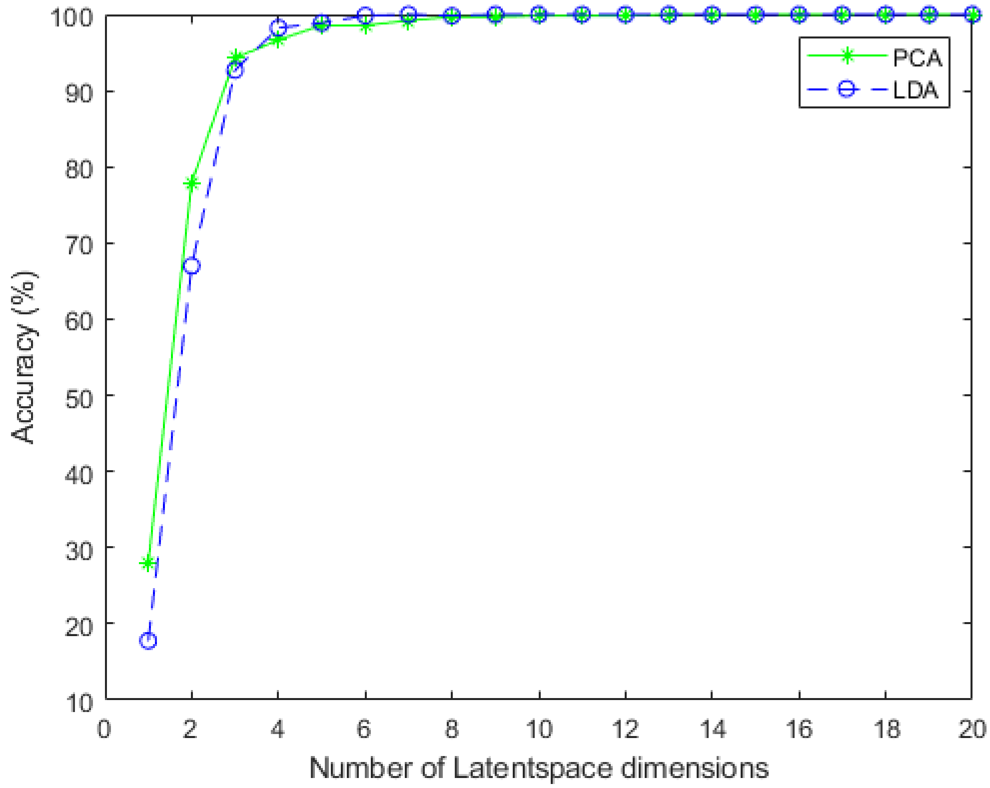

Fig. 1: Accuracy of K-NN classifier with K=1 trained using different number of PCA and LDA reduced features on test dataset

2. Object Detection with Cascaded Classifier using AdaBoost Algorithm :

2.1.Description of Methods:

2.1.1. Haar Feature extraction:

First of all, we extracted the features from all the positive and negative instances in both training and test datasets. For this purpose, we used Haar rectangular filter with multiple sizes and two orientations to extract the large number of features to build the weak classifiers. In the horizontal direction, we used $1 \times 2, 1 \times 4, \cdots, 1 \times 40$ and in vertical direction we used $2 \times 2, 4 \times 2, \cdots, 20 \times 2$ filters. All the pixels falling within the bounds of the rectangles defined by these filters summed using integral image representation. The integral image representation was used to reduce the computational cost.

2.1.2. Build Classifier:

In order to generate the strongest classifier by AdaBoosting, we first obtained the best weak classifier as follows:

1) Assign the weights to both positive and negative classes for all the samples in the training set by *1/2p* and *1/2n*, where *p* and *n* are total number of positive and negative instances. Normalize these weights.

2) For each feature, all the images were sorted based on the feature value in ascending order. For each image in the sorted list, we determined the total sum of positive and negative sample weights, which are represented as T^{+} and T^{-}, respectively. We also computed the sum of positive and negative sample weights below the current sample, which are represented as S^{+} and S^{-}, respectively. We finally computed the error as follows:

$$\epsilon = \min\left(S^{+} + (T^{-} - S^{-}), S^{-} + (T^{+} - S^{+})\right)$$

3) We then find the weak classifier for the current iteration as follows:

$$ht = h(x, f, p, \theta) = \begin{cases} 1 & , if\ pf(x) < \theta \\ 0 & , else \end{cases}$$

   Where, *f*, *p* and θ are the feature, polarity and threshold value that minimizes the error.

4) Now we update the weights for the next iteration as follows:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-\epsilon i}$$

Where, $\epsilon i = 0$ for correctly classified samples, otherwise, it is 1.

5) We checked, if the aggregated classifier achieved the stopping criteria of true detection rate of 1 and false positive rate of 0.5. If we don't satisfy the criteria, we repeated from 2.

6) The final classifier is given as:

$$C(x) = \begin{cases} 1 \quad , if \ \sum_{t=1}^{T} \alpha_t h_t(x) \geq threshold \\ \quad 0 \quad , else \end{cases}$$

Where, $\alpha_t = \log(1/\beta_t)$. The threshold is adjusted to the minimum value of positive samples only in training process to achieve true detection of 1.

7) We finally, checked if the accumulated false positive rate is zero or not. If not, we only selected the misclassified negative samples with all positive samples for the next run to begin from step 1.

2.2.Results:

Table 1: The false positive and number of weak classifiers in each stage for the training data

Stage	**Number of weak classifiers**	**Stage FPR rate**
1	6	0.45108
2	10	0.37305
3	15	0.44378
4	12	0.4823
5	12	0.2876
6	8	0.3
7	5	0

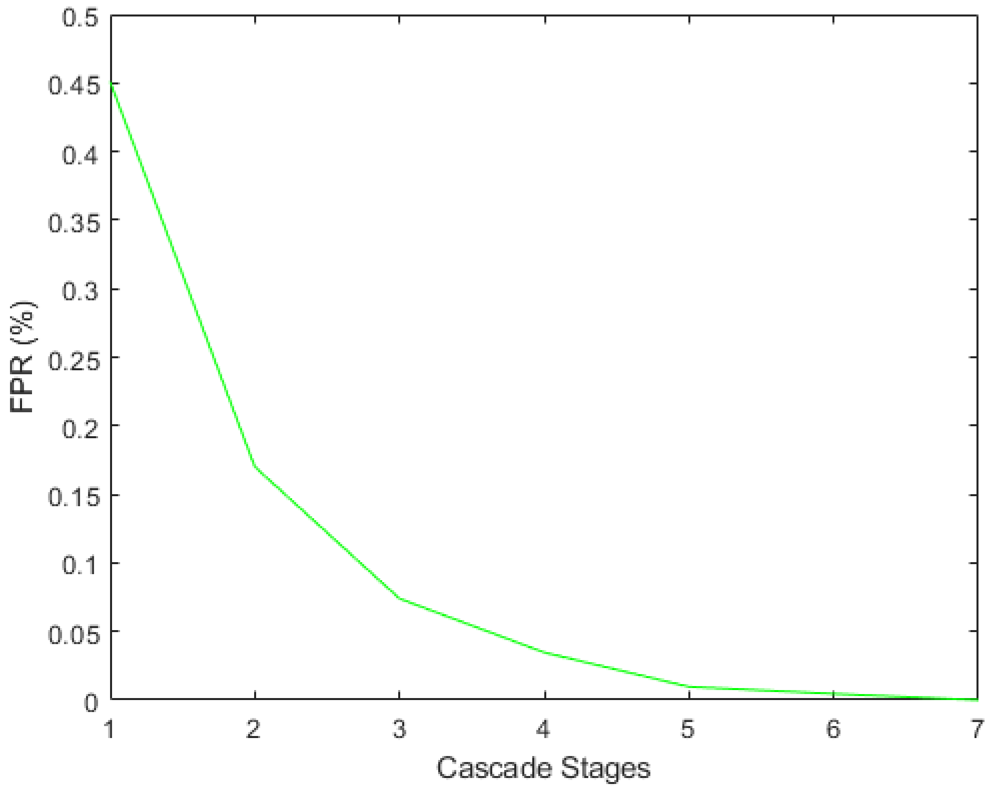

Fig. 2: The false positive rate with the cascade stage for the training data.

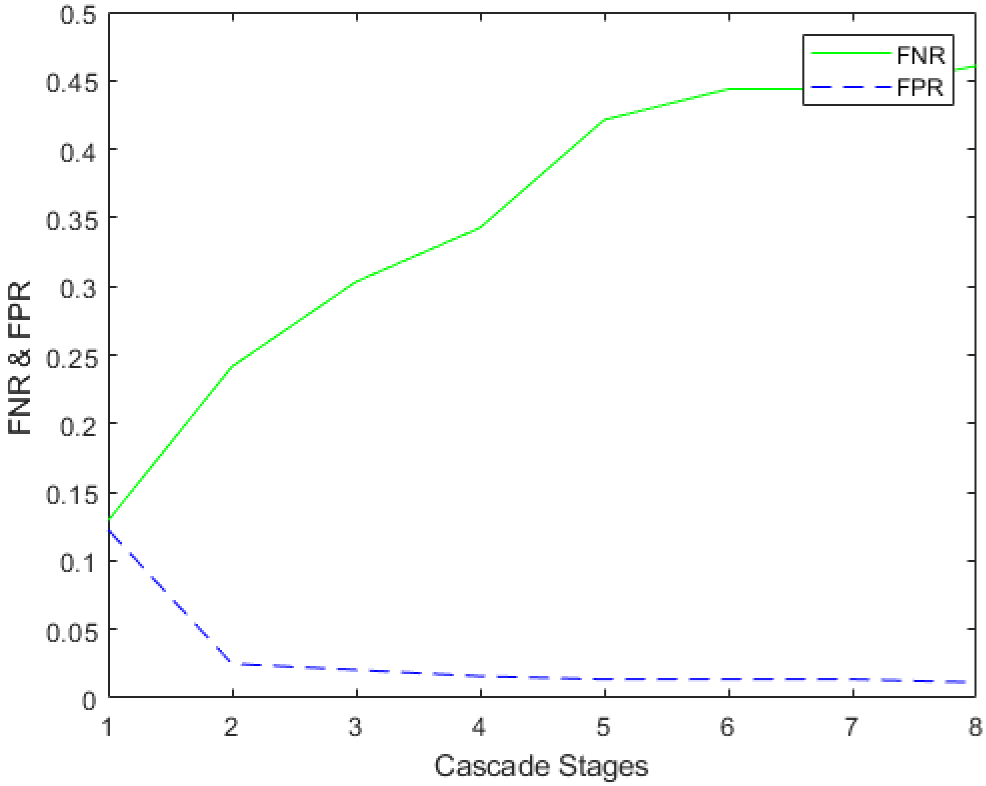

Fig. 2: The false positive and false negative rate with the cascade stage for the test data.

The false positive rate is decreasing as the cascade stage increases, while an opposite trend was observed for the false negative stage. The FPR at the end was 0.0114 and FNR was 0.4607.

3. Source Code:

3.1.Function calls for Task 1:

```
%------This is main function for PCA and LDA face classification ---------
%
close all; warning off
%-----Function calls for the PCA classification-----------%
%number of people in images
n_people=30;
% number of samples per person
n_samples=21;
% number of PCs to be extracted
n_PCs = 20;
%Directories containing the images for train and test datasets
```

```
train_imgs = 'ECE661_2020_hw11_DB1/train/';
test_imgs = 'ECE661_2020_hw11_DB1/test/';
%Load and pre-process the training images
[norm_vec_train, Vec_train, m_train] = PreProcess_images(train_imgs, n_people, n_samples);
%Lod and pre-process the test images
[norm_vec_test, Vec_test, m_test] = PreProcess_images(test_imgs, n_people, n_samples);
%Get the normalized weight vector from the training data
norm_w = PCA_Custom(norm_vec_train);

%Create the class labels
Class_labels=zeros(n_people*n_samples,1);
for i=1:n_people
    Class_labels((i-1)*n_samples+1:(i-1)*n_samples+n_samples,1)=i;
end

Accuracy_PCA = zeros(1, n_PCs);
for i=1: n_PCs
    latent_PCs=norm_w(:,1:i);
    %Reproject the image vectors from training and test data onto
    %sub-space defined by number of columns of weight matrix (cols grow sequentially)
    training_feat_PCA = latent_PCs' * norm_vec_train;
    test_feat_PCA = latent_PCs' * norm_vec_test;
    %Fit the K-NN classifier on training data with K = 1 and Eucledian distance
    Mdl_PCA=fitcknn(training_feat_PCA', Class_labels, 'distance', 'euclidean','NumNeighbors',1,...
        'NSMethod','exhaustive','BreakTies','smallest');
    %Apply the KNN classifier on test features to get predictions
    pred_test_PCA=Mdl_PCA.predict(test_feat_PCA');
    %Count the total correct predicitions
    Correct_total_PCA = sum(pred_test_PCA==Class_labels);
    Accuracy_PCA(1,i) =(Correct_total_PCA/(n_people*n_samples))*100;
end

% %Plot the accuracy against the n_PCs
% plot((1:n_PCs),Accuracy_PCA);

%-------Function Calls for the LDA---------------%
Accuracy_LDA = zeros(1, n_PCs);

[uw, Z] = LDA_custom(Vec_train,m_train,n_people,n_samples);
w = Z * uw;
normw = w./vecnorm(w);
for i = 1:n_PCs
    latent_LDs = normw(:,1:i);
    %Reproject the image vectors from training and test data onto
```

```
        %sub-space defined by number of columns of weight matrix (cols grow sequentially)
        training_features_LDA = latent_LDs' * (Vec_train - m_train);
        test_features_LDA = latent_LDs' * (Vec_test -m_test);
        %Fit the K-NN classifier on training data with K = 1 and Eucledian distance
        MDL_LDA=fitcknn(training_features_LDA', Class_labels, 'distance', 'euclidean','NumNeighbors',1,...
            'NSMethod','exhaustive','BreakTies','smallest');

        %Apply the KNN classifier on test features to get predictions
        pred_test_LDA=MDL_LDA.predict(test_features_LDA');
        %Count the total correct predicitions
        Correct_total_LDA = sum(pred_test_LDA==Class_labels);
        Accuracy_LDA(1,i) =(Correct_total_LDA/(n_people*n_samples))*100;
    end
    %Plot the accuracy against the n_PCs
    plot((1:n_PCs),Accuracy_PCA,'g-*','DisplayName','PCA');
    hold on;
    plot((1:n_PCs),Accuracy_LDA,'b--o','DisplayName','LDA');
    legend;
    hold off;
    xlabel('Number of Latentspace dimensions');
    ylabel('Accuracy (%)');
```

A. Different functions for PCA and LDA classification:

```
 function [norm_vec_all,Vec_imgs_all,m_vec] =
PreProcess_images(filePath,n_people,n_samples)
%%Function responsible for pre-process the images to vectorize them and
compute mean.
%Normalize the vector of images.
%%
%Image dimensions
height = 128; width = 128;
%Size of dataset:
dataset_size = n_people*n_samples;
%Output vector for the entire dataset
Vec_imgs_all = zeros(height*width,dataset_size);
%Read all images one by one, convert them to grayscale and reshape them
%to 1D vector
for i = 1:n_people
 for j = 1:n_samples
     Color_image = imread([filePath,num2file(i),'_',num2file(j),'.png']);
     gray_img = rgb2gray(Color_image);
     Vec_img = double(reshape(gray_img,height*width,1));
     %Normalize the image vector
     Vec_img = Vec_img/norm(Vec_img);
     %Accumulate the vectors from all the images
     Vec_imgs_all(:,(i-1)*n_samples+j) = Vec_img;
 end
end
%Compute the mean of a vector representing all image in dataset
```

```
m_vec = mean(Vec_imgs_all,2);
%Subtract mean
norm_vec_all = Vec_imgs_all - m_vec;
end

function filename = num2file(n)
%If the number passed is less than 10, add precceeding 0
if n <10
    filename = num2str(n,'%02.f');
else
    %If passed is >10, procced as it is
    filename = num2str(n);
end
end
function normW = PCA_Custom(norm_vec_all)
%%
%This is the PCA based decomposition. Makesure don't name it as PCA as it
%is default name used by MAtlab's builtin function.

% Decompose the square matrix of size 630x630 into eig vectors and eg vals
[U,D]= eig(norm_vec_all'*norm_vec_all);
%Sort the diagonalized eigenvalues
[~,idx] = sort(-1 .* diag(D));
U = U(:,idx);
%Compute weights and normalize them using norm of column vectors
w=norm_vec_all*U;
normW = w./vecnorm(w);
%
end
function [dW,Z] = LDA_custom( Vec_train, m_train, n_people, n_samples )
%Image dimensions
height = 128; width = 128;
%Get Class_wise mean
mean_class = zeros(height*width,n_people);
V_mean = zeros(height*width,n_people*n_samples);
for i = 1:n_people
  mean_class(:,i)=mean(Vec_train(:,(i-1)*n_samples+1:(i-
1)*n_samples+n_samples),2);
  V_mean(:,(i-1)*n_samples+1:(i-1)*n_samples+n_samples)= Vec_train(:,(i-
1)*n_samples+1:(i-1)*n_samples+n_samples) - mean_class(:,i);
end
%Compute the difference between class and entire training set mean
Diff_of_Means = mean_class - m_train;
%Decompose between class scatter matrix
[dB,uB] = eig(Diff_of_Means' * Diff_of_Means);
[~,idx] = sort(-1 .* diag(uB));
%dB = dB(:,idx);
%uB = uB(idx);

%Compute the vector V
V = Diff_of_Means * dB;
DB = diag(diag(uB.^(-0.5)));
Z = V*DB;

%Within_Class_scatter
SW = Z' * V_mean;
```

```
[dW,uW] = eig(SW*SW');
[~,idx] = sort(diag(uW));
dW = dW(:,idx);
end
```

3.2.Function calls for Task 2:

3.2.1. Function calls for Task 2:

```
%----This is main function for training and testing AdaBoost Classifier--%

%First of all compute Features
%Training features
% fprintf('Now Extracting features\n');
% Features_data =
ComputeDatasetFeatures('ECE661_2020_hw11_DB2\train\positive\','Train_Positive
.mat');
% Features_data =
ComputeDatasetFeatures('ECE661_2020_hw11_DB2\train\negative\','Train_Negative
.mat');
% %Test Features
% Features_data =
ComputeDatasetFeatures('ECE661_2020_hw11_DB2\test\positive\','Test_Positive.m
at');
% Features_data =
ComputeDatasetFeatures('ECE661_2020_hw11_DB2\test\negative\','Test_Negative.m
at');
%Train the network
Num_FP =
train_AdaBoost('Train_Positive.mat','Train_Negative.mat','Trained_Classifier.
mat');
%Get the FPR and FNR from test dataset
[FPR_test,FNR_test] =
test_AdaBoost('Test_Positive.mat','Test_Negative.mat','Trained_Classifier.mat
');
```

B. Different functions for AdaBoost:

```
function Features_data = ComputeDatasetFeatures(filepath,savename)
%Struct contining all the png files in a directory
img_set = dir([filepath,'*.png']);

n_imgs = length(img_set);
Features_data = zeros(8000+3900,n_imgs);
for i = 1:n_imgs
    Color_img = imread([filepath, img_set(i).name]);
    Gray_img = double(rgb2gray(Color_img));
    Haar_img = integralImage(Gray_img);
    Features_data(:,i) = ComputeFeatures(Haar_img);
end
%Dataset_Features.features = Features_data;

%Save the Computed features of dataset
save(savename,'Features_data','-v7.3');
```

```
end
function All_Features = ComputeFeatures(Haar_img)
%%Function responsible for computing the features from an input Haar image
%Get the size of Haar image. This will be +1 pixel compared to original
%image size
[img_height,img_width] = size(Haar_img);
Horizontal_Features =
ComputeHorizontalFeatures(Haar_img,img_width,img_height);
Vertical_Features = ComputeVerticalFeatures(Haar_img,img_width,img_height);
All_Features = [Horizontal_Features;Vertical_Features];
end

function Features=ComputeHorizontalFeatures(Haar_img,img_width,img_height)
Features = [];
kernel_sizes = 2:2:img_width-1;
%Control the number of kernels
for n  = 1:length(kernel_sizes)
    kernel = kernel_sizes(n);
    for i = 1:img_height-1 %Height
        for j = 1:(img_width-1 - kernel + 1)%Width
            kh = kernel/2;  %Half_kernel
            %Compute the two sums
            Sum_1 = ComputeSum([i;j; i;(j+kh);(i + 1); j;(i +
1);(j+kh)],Haar_img);
            Sum_2 =
ComputeSum([i;(j+kh);i;(j+kernel);i+1;(j+kh);i+1;j+kernel],Haar_img);
            Features = [Features; (Sum_2 - Sum_1)];
        end
    end
end
end
function Features = ComputeVerticalFeatures(Haar_img,img_width,img_height)
Features =[];
%Control the number of kernels
kernel_size = 2:2:img_height-1;
for n  = 1:length(kernel_size)
    kernel = kernel_size(n);
    for i = 1:(img_height -1 - kernel + 1)%Height
        for j = 1:img_width - 2 %Width
            kh = kernel/2;  %Half_kernel
            %Compute the two sums
            Sum_1 = ComputeSum([i;j;i;j+2;i+kh;j;i+kh;j+2],Haar_img);
            Sum_2 =
ComputeSum([i+kh;j;i+kh;j+2;i+kernel;j;i+kernel;j+2],Haar_img);
            Features = [Features; (Sum_1 - Sum_2)];
        end
    end
end
end
function Pixel = ComputeSum(Corners,Harr_Img)
    cor1 = Harr_Img(Corners(1), Corners(2));
    cor2 = Harr_Img(Corners(3), Corners(4));
    cor3 = Harr_Img(Corners(5), Corners(6));
    cor4 = Harr_Img(Corners(7), Corners(8));
    Pixel = cor4 + cor1 - cor2 - cor3;
end
```

```matlab
function
Num_FP=train_AdaBoost(Positive_Feat_file,negative_Feat_file,savename)
% Load the stored features for positive and negative examples
Features_positive_train = load(Positive_Feat_file);
Features_negative_train=load(negative_Feat_file);
Feat_positive = Features_positive_train.Features_data;
Feat_neagative = Features_negative_train.Features_data;
%Num of positive and negative samples in the data
Positive_samples = size(Feat_positive,2);
Negative_samples = size(Feat_neagative,2);

%Concatenate all the features
Combined_Features = [Feat_positive,Feat_neagative];
%Build Bossted classifiers
for i = 1:10
    Clss_stages(i,1)=
AdaBoost_Classifier(Combined_Features,Positive_samples,1,0.5);
    % Get the new updated features for next iterations
    Combined_Features = Clss_stages(i,1).NewFeatures;
    %Save the fsalse psotitive observations for plotting
    Num_FP(i,1) = size(Combined_Features,2) - Positive_samples;
    fprintf('Now Running at Stage %s\n', num2str(i));
    fprintf(['Fasle Positives = ', num2str(Num_FP(i,1))]);
    fprintf('\n');
    %Stop if only positive features are left in Combined features
    if (size(Combined_Features,2) == Positive_samples)
        break;
    end
end
plot (1:length(Num_FP),Num_FP/Negative_samples,'g-');
xlabel('Cascade Stages');
ylabel('FPR (%)');
save(savename,'Clss_stages','-v7.3');
end
function
[FPR_test,FNR_test]=test_AdaBoost(Positive_Feat_file,negative_Feat_file,saven
ame)
% Load the stored features for positive and negative examples
Feat_positive = load(Positive_Feat_file);
Feat_neagative=load(negative_Feat_file);
Features_positive_test = Feat_positive.Features_data;
Features_negative_test = Feat_neagative.Features_data;
%Num of positive and negative samples in the data
Positive_samples = size(Features_positive_test,2);
Negative_samples = size(Features_negative_test,2);
%Load the models
load(savename);
FPR_test = zeros(length(Strong_Clf_train),1);
FNR_test = zeros(length(Strong_Clf_train),1);
FN_test = 0;
TN_test = 0;
for i = 1:length(Strong_Clf_train)
    Class_Stage = Strong_Clf_train(i);
    %Get the combined features
    Combined_Features = [Features_positive_test,Features_negative_test];
```

```
    Pred =
ClassifyTest(Combined_Features,size(Features_positive_test,2),size(Features_n
egative_test,2),Class_Stage);
    Positive_samples_satge = Pred.PositiveSamples;
    Pred_Results = Pred.ClassificationResults;
    %Measure FNR anf FPR for current stage
    FN_test = FN_test + length(find(Pred_Results(1:Positive_samples_satge) ==
0));
    TN_test = TN_test +
length(find(Pred_Results(Positive_samples_satge+1:end) == 0));
    FNR_test(i) = FN_test / Positive_samples;
    FPR_test(i) = (Negative_samples - TN_test) / Negative_samples;
    %Only those observations which are misclassified for next stage
    idx_positive = find(Pred_Results(1:Positive_samples_satge) == 1);
    idx_negative = find(Pred_Results(Positive_samples_satge+1:end) == 1);
    Features_positive_test = Features_positive_test(:,idx_positive);
    Features_negative_test = Features_negative_test(:,idx_negative);
end
plot(1:1:length(Strong_Clf_train),FNR_test,'g-')
hold on
plot(1:1:length(Strong_Clf_train),FPR_test,'b--')
hold off
legend('FNR','FPR');
xlabel('Cascade Stages');
ylabel('FNR & FPR ');
end

function Ada_Classifier =
AdaBoost_Classifier(Combined_Features,n_Positive,tpr_thresh,fpr_thresh)
%Function returning the AdaBoost Classifier structure
% Maximum iterations for the Ada-Boosting
T = 20;
%Total and negative samples in current iteration
total_samples = size(Combined_Features,2);
n_Negative = total_samples - n_Positive;

%Compute the weights based on the num of oservations (prior probability)
weight_pos(1:n_Positive,1) = 0.5/n_Positive;
weight_neg(1:n_Negative,1) = 0.5/n_Negative;
%Concatenate weights
Weight_comb = [weight_pos;weight_neg];
%Define the true labels for psoitive and negative classes
label_pos(1:n_Positive,1) = 1;
label_neg(1:n_Negative,1) = 0;
%Concatenate the labels
lbls_comb = [label_pos;label_neg];
%Parameters of AdaBosst Classifier
Clf_params = zeros(T,4);
cascade_results = [];
alphas = [];
TPR = [];
FPR = [];

%Get new features after removing the negative examples with correct
%classification
%Features_new = Combined_Features(:,img_idx);
```

```
for i = 1:T
    %Normalize the updated weights at every iteration
    Weight_comb = Weight_comb./sum(Weight_comb);
    %Get the cascade classifier results
    Classifier =
Cascade_Classifier(Combined_Features,n_Positive,Weight_comb,lbls_comb);
    %Calculating the parameters for updating the weights
    beta = Classifier.min_Err / (1-Classifier.min_Err);
    a = log(1/beta);
    %Updating the weights for next cascade classifier
    Weight_comb = Weight_comb.*beta.^(1-abs(lbls_comb-
Classifier.classification_results));
    %Append the lists for making comaprison
    alphas = [alphas;a];
    cascade_results = [cascade_results,Classifier.classification_results];

    %Find the threshold alpha by finding min of class_results*alpha
    C = cascade_results(:,1:i) * alphas(1:i,1);
    threshold_alpha = min(C(1:n_Positive));
    %Find all the observations which are above threshold
    Cx = C >= threshold_alpha;

    %Compute the TPR and FNR for current cascade classfier
    tpr = sum(Cx(1:n_Positive))/n_Positive;
    fpr = sum(Cx(n_Positive+1:end))/n_Negative;
    TPR = [TPR;tpr];
    FPR = [FPR;fpr];
    %Terminate the search if current TPR and FPR meets the requirement
    if ((tpr >= tpr_thresh) && (fpr <= fpr_thresh))
        break;
    end
    %Keep the copy of best parameters
    Clf_params(i,:) =
[Classifier.Features,Classifier.theta,Classifier.polarity,a];
end

%Now pick only those negative examples which are miscalssified for new
%cascade
[negative_sorted,sorted_idx] = sort(Cx(n_Positive+1:end));
for j = 1:n_Negative
    if negative_sorted(j)>0
    neg_misclassified = sorted_idx(j:end);
    break;
    end
end
%Index containing all positive and miscalssified negative observations
if sum(negative_sorted)>0
    new_idx = [1:n_Positive,neg_misclassified'+n_Positive];
else
    new_idx = 1:n_Positive;
end
new_Features = Combined_Features(:,new_idx);

%Update the classifier structure.
```

```
Ada_Classifier.ClassifierParams = Clf_params;
Ada_Classifier.NewIdx = new_idx;
Ada_Classifier.Iterations = i;
Ada_Classifier.FPR = FPR(i);
Ada_Classifier.NewFeatures = new_Features;
end

function BestClassifier =
Cascade_Classifier(Combined_Features,n_Positive,Weights,lbls)
%% Function returns a classifier structure.
%Total samples obtained from combined psotive and negative samples
total_samples = size(Combined_Features,2);
%Sum of weights for positive and negative examples
Total_positive = sum(Weights(1:n_Positive,1));
Total_negative =sum(Weights(n_Positive+1:end));
BestClassifier.min_Err = inf;

for i = 1:length(Combined_Features)
    %Initialize the clasisification results
    Class_results = zeros(total_samples,1);
    %Get idx of sorting all the samples with respect to a feature.
    [sorted_feats,idx] = sort(Combined_Features(i,:));
    sorted_weight = Weights(idx);
    sorted_lbl = lbls(idx);
    %Compute the cummulative sume of the sorted weights
    Sum_positive = cumsum(sorted_weight.*sorted_lbl);
    Sum_negative = cumsum(sorted_weight) - Sum_positive;
    %Compute two types of error
    Err_1 = Sum_positive + (Total_negative - Sum_negative);
    Err_2 = Sum_negative + (Total_positive - Sum_positive);
    %Find the minimum of two errorz
    minErr = min(Err_1,Err_2);
    %The minimum value and index of minimum of two erros
    [min_min_Err, Err_idx] = min(minErr);
    %Calssify the samples.
    if Err_1(Err_idx) > Err_2(Err_idx)
       %All values are correctly classified upto error index, rest are
       %falsely classified, leaving them zero.
       Polarity = 1;
       Class_results(1:Err_idx,1) = 1;
       Class_results(idx) = Class_results;
    else
       %All values are correctly classified after error index, before that
       %index all values are falsely classfiied, leaving them zeo.
       Polarity = -1;
       Class_results(Err_idx + 1:end,1) = 1;
       Class_results(idx) = Class_results;
    end

    if Err_idx == 1
         BestClassifier.theta = sorted_feats(1) - 0.5;
    elseif Err_idx == size(Combined_Features,1)
         BestClassifier.theta = sorted_feats(size(Combined_Features,1)) +
0.5;
    else
```

```
         BestClassifier.theta = mean([sorted_feats(Err_idx),
sorted_feats(Err_idx-1)]);
    end
     %Check if the current error is lower than earlier error to find new
best.
     if min_min_Err < BestClassifier.min_Err
         %Update the classifier structure.
         BestClassifier.min_Err = min_min_Err;
         BestClassifier.classification_results = Class_results;
         BestClassifier.Features = i;
         BestClassifier.polarity = Polarity;

     end
end
end
```