
MATLAB code for eigenface for face recognition

```
% Principal Component Analysis for face recognition
% M training images, sized N pixels wide by N pixels tall
% c recognition images, also sized N by N pixels
% Mp = desired number of principal components

% Feature Extraction:
% merge column vector for each training face
X = [x1 x2 ... xm]
% compute the average face
me = mean(X,2)
A = X - [me me ... me]
% avoids N^2 by N^2 matrix computation of [V,D]=eig(A*transpose(A))
% only computes M columns of U: A=U*E*transpose(V)
[U,E,V] = svd(A,0)
eigVals = diag(E)
lmda = eigVals(1:Mp)
% pick face-space principal components (eigenfaces)
P = U(:,1:Mp)
% store weights of training data projected into eigenspace
train_wt = transpose(P)*A

Nearest-Neighbor Classification:
% A2 created from the recog data (in similar manner to A)
recog_wt = transpose(P)*A2
% euclidean distance for ith recog face, jth train face
euDis(i,j) = sqrt((recog_wt(:,j)-train_wt(:,i)).^2)
```

MATLAB code for implementation of two-dimensional PCA

```
function [ WA,WB ] = pca2d( A,B,D )
%perform 2-dimensional PCA on training set A
of size mxnxN and test set B of size mxnxP
% i.e. there are N training and P test samples (images)
each of size mxn
      % D is the dimension to which A will be reduced
      A=double(A);B=double(B);
      [m n N]=size(A);[m n P]=size(B);
      total=A(:,:,1);
      for k=2:N
          total=total+A(:,:,k);
      end
      miu=total/N; % mean of A
      for k=1:N
         A(:,:,k)=A(:,:,k)-miu; % adjust A
      end
      G=zeros(n,n);
      for k=1:N
          G=G+transpose(A(:,:,k))*A(:,:,k);
      end
      G=G/N;
      [y,l]=eig(G);% find eigen value and eigen vector
      l=diag(l);
      % find first D highest Eigen values
       and store the associated Eigen
      % vectors in Y
      [val,ind]=sort(l,"descend");
% sort Eigen values in descending order
      Y=[];
      for j=1:D
          Y=[Y y(:,ind(j))];
      end
      Y=Y./D; % normalize Y
      for k=1:N
          X(:,:,k)=A(:,:,k)*Y;
      end
      WA=X
%  find space projection projB of test set B
   for k=1:P
         B(:,:,k)=B(:,:,k)-miu; % adjust A
   end
   for k=1:P
          WB(:,:,k)=B(:,:,k)*Y;
   end
   end
```

MATLAB code for implementation of Kernel PCA

```
function [ WA,WB ] = pcaKernel( A,B,D )}
   \texttt{%perform Kernel PCA on training set A and test set B
   % D is the dimension to which A will be reduced
      A=double(A);B=double(B);
[M N]=size(A);[M P]=size(B);
      miu=mean(transpose(transpose(A)));}
\texttt{% find row-wise mean of A
      for j=1:N
         A(:,j)=A(:,j)-miu; % adjust A
      end
      KA=((transpose(A)*A)+4).^2; % kernel of A
      Kmiu=mean(transpose(transpose(KA)));
      for j=1:N
        KA(:,j)=KA(:,j)-Kmiu; % adjust KA
      end
      oneA=ones(N,N)./N;
      KA=KA-oneA*KA-KA*oneA+oneA*KA*oneA;
      [y,l]=eig(KA/N);
% find eigen value and eigen vector
      l=diag(l);
      % find first D highest Eigen values
       and store the associated Eigen
      % vectors in Y
      [val,ind]=sort(l,"descend");
% sort Eigen values in descending order
      Y=[];
      D=D;
      for j=1:D
          Y=[Y y(:,ind(j))];
      end
      Y=Y./D; % normalize Y
      X=KA*Y;
      WA=X*transpose(KA);
% D-dimensional space projection of training images
   KB=((transpose(B)*A)+4).^2;
   oneB=ones(P,N)./N;
   KB=(KB-(oneB*KA)-(KB*oneA)+(oneB*KA*oneA));
   WB=X*transpose(KB);
end
```

```
% Fisherface

%% same training & recognition images, also sized N by N pixels
% P1 = eigenface result
% Feature Extraction:
% same as eigenface
A = X - [me me ... me]
% compute N^2 by N^2 between-class scatter matrix
for i=1:c
   Sb = Sb + clsMeani*transpose(clsMeani)

% compute N^2 by N^2 within-class scatter matrix
for i=1:c, j=1:ci
   Sw = Sw + (X(j)-clsMeani)*transpose(X(j)-clsMeani)
% project into (N-c) by (N-c) subspace using PCA
Sbb = transpose(P1)*Sb*P1
Sww = transpose(P1)*Sw*P1
% generalized eigenvalue decomposition
% solves Sbb*V = Sww*V*D
[V,D] = eig(Sbb,Sww)
eigVals = diag(D)
lmda = eigVals(1:Mp)
P = P1*V(:,1:Mp)

% store training weights
train_wt = transpose(P)*A
%% Nearest-Neighbor Classification:
% same as eigenface
```

PYTHON code for implementation of principal component analysis

```
# Classification for two class case using PCA

import numpy as np
from matplotlib import pyplot as plt
from operator import itemgetter

plt.rc("font", family="serif",size=18,weight="light")
#plt.rc("text",usetex=True)
```

```
#class1 = np.array([[2.5,2.4],[2.2,2.9],[3.1,3.0],[2.3,2.7],[1.9,2.2]])
#class2 = np.array([[0.5,0.7],[1,1.1],[1.5,1.6],[1.1,0.9],[2,1.6]])
plt.close("all")
class1 =np.array([[1.,2.],[2.,3.],[3.,3.],[4.,5.],[5.,5.]])
N1 = len(class1)
class2 = np.array([[1.,0.],[2.,1.],[3.,1.],[3.,2.],[5.,3.],[6.,5.]])
N2 = len(class2)
data = np.vstack((class1,class2))

plt.figure(1)
plt.scatter(data[0:N1,0],data[0:N1,1],s=240,c=[0.9,0.9,0.9],\
marker="o",label="original class-1",alpha=0.9)
plt.scatter(data[N1:N1+N2,0],data[N1:N1+N2,1],\
s=240,c=[0.0,0.0,0.0],marker="4",label="original class-2")
plt.grid(axis="both")
plt.legend(loc=0,prop={"size":14})
plt.title("Original data")
plt.xlim(-1,1.5*data.max())
plt.ylim(-1,1.5*data.max())
plt.xlabel("variable-1")
plt.ylabel("variable-2")
plt.show()
m = np.array([data.mean(axis=0)])
M = np.tile(m,(data.shape[0],1))
D = data - M
Cov =  np.cov(D.T)
CovMat = float(1./(D.shape[0]-1.)) * np.dot(D.T,D)
val,vec = np.linalg.eig(CovMat)
tmp = np.zeros((val.shape))
tmpvec = np.zeros((vec.shape))
for i in range(len(val)):
    a = max(enumerate(val), key=itemgetter(1))[0]
    tmp[i] = val[a]
    tmpvec[:,i] = vec[:,a]
    val[a]=0


plt.figure(2)
plt.scatter(D[0:N1,0],D[0:N1,1],s=240,c=[0.9,0.9,0.9],\
marker="o",label="class-1,MS",alpha=0.9)
plt.scatter(D[N1:N1+N2,0],D[N1:N1+N2,1],s=240,c=[0,0,0],\
marker="4",label="class-2,MS")
plt.grid(axis="both")
plt.plot([-5*tmpvec[0,0],5*tmpvec[0,0]] ,[-5*tmpvec[1,0],\
5*tmpvec[1,0]],"--k",lw=2,label="eigvec_1")
plt.plot([-5*tmpvec[0,1],5*tmpvec[0,1]] ,[-5*tmpvec[1,1],\
5*tmpvec[1,1]],"-k",lw=2,label="eigvec_2")
plt.xlim(-data.max(),data.max())
plt.ylim(-data.max(),data.max())
plt.legend(loc=0,prop={"size":14})
plt.title("Mean subtracted data")
plt.show()
## Data reconstruction with all eigen vectors
transData = np.dot(D,tmpvec) # taking all eigen vectors

plt.figure(3)
pc = tmpvec
reconstructed = np.dot(transData,pc.T) + M
plt.scatter(reconstructed[0:N1,0],reconstructed[0:N1,1],\
s=240,c=[0.9,0.9,0.9],marker="o",label="class-1 reconstructed")
plt.scatter(reconstructed[N1:N1+N2,0],\
reconstructed[N1:N1+N2,1],s=240,c=[0.0,0.0,0.0],\
marker="4",label="class-2 reconstructed")
plt.grid(axis="both")
plt.legend(loc=0,prop={"size":14})
plt.xlim(0,10)
plt.ylim(-1,10)
plt.title("Reconstructed with all eigenvectors")
plt.show()
## Data reconstruction with  eigen vector having maximum variance
plt.figure(4)
pc = np.array([tmpvec[:,0]]).T
rec = np.dot(D,pc) + M
plt.scatter(rec[0:N1,0],rec[0:N1,1],s=240,c=[0.9,0.9,0.9],\
marker="o",label="class-1 reconstructed",alpha=0.5)
plt.scatter(rec[N1:N1+N2,0],rec[N1:N1+N2,1],s=240,c=[0.0,0.0,0.0],\
marker="4",label="class-2 reconstructed")
plt.plot([-10*tmpvec[0,0],10*tmpvec[0,0]] ,[-10*tmpvec[1,0],\
10*tmpvec[1,0]],"--k",lw=2,label="eigvec_1")
plt.grid(axis="both")
plt.legend(loc=0,prop={"size":14})
plt.xlim(0,8)
plt.ylim(-1,8)
plt.title("Reconstructed with one eigenvector")

plt.show()
plt.figure(5)
rec = rec -M
rec[:,1] = 0
plt.scatter(rec[0:N1,0],rec[0:N1,1],s=200,c=[1,1,1],\
marker="o",label="Reduced Space-class-1")
plt.scatter(rec[N1:N1+N2,0],rec[N1:N1+N2,1],s=160,c=[1,1,1],\
marker="*",label="Reduced Space-class-2")
plt.xlim(-(rec.max()+0.5),rec.max()+0.5)
plt.ylim(-(rec.max()+0.5),rec.max()+0.5)
plt.grid(axis="both")
plt.legend(loc=0,prop={"size":12})
plt.show()
```

PYTHON code for implementation of Fisher linear discriminant analysis

---

```
# Classification for two class case using FLDA
import numpy as np
from matplotlib import pyplot as plt
from operator import itemgetter
plt.rc("font", family="serif",size=12,weight="light")
plt.close("all")
class1 =np.array([[1.,2.],[2.,3.],[3.,3.],[4.,5.],[5.,5.]])
#class1 =np.array([[4,2],[2,4],[2,3],[3,6],[4,4]])
N1 = len(class1)
class2 = np.array([[1.,0.],[2.,1.],[3.,1.],[3.,2.],[5.,3.],[6.,5.]])
#class2 = np.array([[9,10],[6,8],[9,5],[8,7],[10,8]])
N2 = len(class2)

m1 = np.array([class1.mean(axis=0)])
m2 = np.array([class2.mean(axis=0)])
d1 = class1 - np.tile(m1,(class1.shape[0],1))
d2 = class2 - np.tile(m2,(class2.shape[0],1))
S1 = float(1./(class1.shape[0]-1.)) * np.dot(d1.T,d1)
S2 = float(1./(class2.shape[0]-1)) * np.dot(d2.T,d2)
Sw = S1 + S2
Sb = np.dot((m1-m2).T,(m1-m2))

invSw = np.linalg.inv(Sw)
invSwSb = np.dot( invSw , Sb)
val,vec = np.linalg.eig(invSwSb)
```

```
tmp = np.zeros((val.shape))
tmpvec = np.zeros((vec.shape))
for i in range(len(val)):
    a = max(enumerate(val), key=itemgetter(1))[0]
    tmp[i] = val[a]
    tmpvec[:,i] = vec[:,a]
    val[a]=0
w = np.array([tmpvec[:,0]]).T
#"""" eigenvector with largest eigenvalue """"
#plt.plot([-5*vec[0,1],5*vec[0,1]] ,[-5*vec[1,1],5*vec[1,1]],
"-k",lw=2,label="eigvec_1")
plt.figure(1)
plt.scatter(class1[:,0],class1[:,1],s=240,c=[0.9,0.9,0.9],\
marker="o",label="class-1",alpha=0.9)
plt.scatter(class2[:,0],class2[:,1],s=240,c=[0.,0.,0.],\
marker="4",label="class-2")
plt.xlim(-5,15)
plt.ylim(-5,15)
plt.plot([-5*w[0],20*w[0]] ,[-5*w[1],20*w[1]],\
"--k",lw=2,label="Optimal eig_vec")
plt.legend(loc=0,prop={"size":14})
plt.grid(axis="both")
plt.title("Direction of optimal eigen vector")
plt.show()

p1 = np.dot(class1,tmpvec) # projection of class-1 on optimal eigenvector
p2 = np.dot(class2,tmpvec)
p1[:,1] = 0 # taking the projected values only on optimal vector
p2[:,1] = 0
rec = np.vstack((p1,p2))
plt.figure(2)
plt.scatter(rec[0:N1,0],rec[0:N1,1],s=250,c=[1,1,1],\
marker="o",label="Reduced Space-class-1")
plt.scatter(rec[N1:N1+N2,0],rec[N1:N1+N2,1],s=250,\
c=[1,1,1],marker="*",label="Reduced Space-class-2")
plt.xlim(-(rec.max()+0.5),rec.max()+0.5)
plt.ylim(-(rec.max()+0.5),rec.max()+0.5)
plt.grid(axis="both")
plt.legend(loc=0,prop={"size":12})
plt.title("Projected data in reduced space using FLDA")
plt.show()
```

PYTHON code for 1D Gaussian function

```
# 1D Gaussian function

import numpy as np
from matplotlib import pyplot as plt

plt.rc("font", family="serif",size=12,weight="light")
plt.close("all")

X = np.array([59,61,48,45,67,55])
Y = np.array([29,19,17,30,22,25])
mx = X.mean(axis=0)
sx = np.std(X)
Nx = np.zeros((100,1),dtype=float)
my = Y.mean(axis=0)
sy = np.std(Y)
Ny = np.zeros((100,1),dtype=float)

for i in range(0,100):
    Nx[i,:] = (1/np.sqrt(2*np.pi*sx**2)) * \
    np.exp(-(0.5/sx**2)*(i-mx)**2)\

    Ny[i,:] = (1/np.sqrt(2*np.pi*sy**2)) * np.exp(-(0.5/sy**2)*(i-my)**2)
plt.Figure  (1)
plt.xlabel("Data")
plt.ylabel("Normal Distribution")
plt.plot(Nx,"-b",label="X data",linewidth=3)
plt.plot(Ny,"-r",label="Y data",linewidth=3)
plt.legend(loc="upper right")
plt.grid(axis="both")
plt.show()
```

PYTHON code for bivariate Gaussian function

```
# Bivariate Gaussian function

from numpy import *
```

```
import numpy as np
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

plt.close("all")

K1 = array([[1,0.9],[0.3,1]])
m1 = array([2.5,2.5])
normfac = 1/(pow(np.linalg.det(K1),0.5)*(2*pi))
x = arange(0,5,0.05,dtype=float)
p = zeros((size(x,0),size(x,0)),dtype=float)
for i in range(0,size(x,0)):
    for j in range(0,size(x,0)):
        fst = dot(array([[x[i],x[j]] - m1]) , (np.linalg.inv(K1)))
        p[i,j] = normfac * exp(-0.5
        * dot (fst ,array([[x[i],x[j]] -
        m1]).T))

xx,yy = meshgrid(x,x)
fig = plt.Figure  ()
ax = Figure   gca(projection="3d")
ax.plot_surface(xx,yy,p,rstride=4,cstride=4,linewidth=1\
,antialiased=False,cmap=cm.coolwarm)
#cset = ax.contour(xx, yy, p, offset=-0.1, cmap=cm.coolwarm)
#ax.view_init(elev=65., azim=-110.)
ax.set_zlim(-0.1, p.max())
#fig, ax = plt.subplots()
#cset = ax.contour(x,x,p)

fig, ax = plt.subplots()
h=ax.contour(p, cmap=cm.RdBu, vmin=abs(p).min(),\
 vmax=abs(p).max(), extent=[0, 6, 0, 6])
numsamp = 1000
[lamda,eta] = np.linalg.eig(K1)
coeffs = np.random.rand(numsamp,2)
samples = dot(coeffs,eta.T) + np.ones((numsamp,1))*m1
plt.plot(samples[:,0],samples[:,1],".k")
plt.show()
```

MATLAB code for Bayes' boundary for linearly separable data

```
%% Bayes boundary for linearly separable data

x=linspace(-10,10,300);
y =linspace(-10,10,300);
[X,Y] = meshgrid(x,y);
mu = [3 6];
Sigma = [0.5 0; 0 2]; R = chol(Sigma);
z = repmat(mu,300,1) + randn(300,2)*R;
plot(z(:,1),z(:,2),"^r","MarkerSize",6);hold on
mu = [3 -2];
Sigma = [2 0; 0 2]; R = chol(Sigma);
z = repmat(mu,300,1) + randn(300,2)*R;
plot(z(:,1),z(:,2),"ob","MarkerSize",6);hold on
axis([-4 10 -10 10])
x=linspace(-10,10,100);
y = 3.335-1.125.*x+0.1875*x.^2;
plot(x,y,"-k","linewidth",3);
legend("class-1","class-2","Boundary");
```

MATLAB code for Bayes' boundary for non linearly separable data

```
%% Bayes boundary for non linearly separable data

syms x1 x2;
mu1 =[2,2];
sigma1 = [1,0;0,1];
mu2 =[3,-3];
sigma2=[1,0;0,1];
R1 = chol(sigma1);
z1 = repmat(mu1,300,1) + randn(300,2)*R1;
R2 = chol(sigma2);
z2 = repmat(mu2,300,1) + randn(300,2)*R2;
plot(z1(:,1),z1(:,2),"^g","MarkerSize",6);hold on
plot(z2(:,1),z2(:,2),"om","MarkerSize",6);hold on
g1 = -0.5*transpose([x1;x2]-transpose(mu1))*...
     inv(sigma1)*([x1;x2]-transpose(mu1))...
    -log(2*pi)-0.5*log(det(sigma1));
g2 = -0.5*transpose([x1;x2]-transpose(mu2))*...
```

```
     inv(sigma2)*([x1;x2]-transpose(mu2))...
    -log(2*pi)-0.5*log(det(sigma2));
g = g1-g2;
h = ezplot(g,[[-2,8],[-5,8]]);
set(h,"linewidth",3)
```

MATLAB code for face detection using BDF

```
%% Matlab Code for Face Detection using BDF
for i = 1:100
    img = im2double( imread(strcat("/home/pkb/scale/facepart/",...
        num2str(i),".jpg")));
    Y = bdffeature(img);
    Yf(:,i) = Y;
    clear Y
end
for i = 1:100
    img = im2double( imread(strcat("/home/pkb/scale/Nonface",...
        num2str(i),".jpg")));
    Y = bdffeature(img);
    Yn(:,i) = Y;
    clear Y
end
M = 60;N = size(Yf,2);
[vecf,valf,Mf] = bdfpca(Yf,M);
[vecn,valn,Mn] = bdfpca(Yn,M);
facecnt = 1;
for k = 1:350
    img = im2double( imread(strcat("/home/pkb/scale/Nonface/",...
        num2str(k),".jpg")));
    Y = bdffeature(img);
    Z = transpose(vecf)*(Y-Mf);
    zisq = Z.^2;
    lamda = valf(1:M);
    frac = zisq./lamda;
    fst =  sum(frac);
    ro = (1/(N-M))*sum(valf(M+1:N));
    snd = (norm(Y-Mf) - sum(zisq))/ro;
    trd = log(prod(valf(1:M)));
    frth = (N-M)*log(ro);
    deltf = (fst+snd+trd+frth)*10^-8+0.1;
    U = transpose(vecn)*(Y-Mn);
    uisq = U.^2;
    lamda = valn(1:M);
    frac = uisq./lamda;
    fst =  sum(frac);
    ep = (1/(N-M))*sum(valn(M+1:N));
    snd = (norm(Y-Mn) - sum(uisq))/ep;
    trd = log(prod(valn(1:M)));
    frth = (N-M)*log(ep);
    deltn = (fst+snd+trd+frth)*10^-7;
    if (deltf   < deltn) && (deltf<0)
        %display "face";
        facecnt
        facecnt=facecnt+1;
    else
        %display "nonface";
    end
end
```

MATLAB code for feature selection function using BDF

```
%% Feature selection function using BDF
function [Y,Xh] = bdffeature(img)
F=img(:);
F = (F-mean(F))/std(F);
for k = 1:size(img,1)-1
    h(:,k) =img(:,k+1)- img(:,k);
end
Xh = h(:);
Xh = (Xh-mean(Xh))/std(Xh);
for k = 1:size(img,2)-1
    v(k,:) =img(k+1,:)- img(k,:);
end
Xv = v(:);
Xv = (Xv-mean(Xv))/std(Xv);
Xr = sum(img,1);
Xr = (Xr-mean(Xr))/std(Xr);
Xc = sum(img,2);
Xc = (Xc-mean(Xc))/std(Xc);
```

```
Y = cat(1,F,Xh,Xv,transpose(Xr),Xc);
Y = (Y-mean(Y))/std(Y);
```

MATLAB code for face detection using BDF

```
%% Matlab Code for Face Detection using BDF
for i = 1:99
    img = im2double( imread(strcat...
        ("/home/pradipta/PRADIPTA/Database/facetrain/",...
        num2str(i),".jpg")));
    Y = bdffeature(img);
    Yf(:,i) = Y;
    clear Y img
end
for i = 1:355
    img = im2double( imread(strcat...
        ("/home/pradipta/PRADIPTA/Database/Nonface/",...
        num2str(i),".jpg")));
    Y = bdffeature(img);
    Yn(:,i) = Y;
    clear Y
end
M = 15
Nf = size(Yf,2);
Nn = size(Yn,2);
[vecf,valf,Mf] = bdfpca(Yf,M);
[vecn,valn,Mn] = bdfpca(Yn,M);
facecnt = 1;
for siz = 0.3:0.01:0.48
    test = imread...
        ("home/pradipta/PRADIPTA/Database/CroppedFaces/147.jpg");
    if size(test,3)>1
        test = rgb2gray(test);
    end
    test = imresize(test,siz);
    figure
    imshow(test,[]);title(num2str(siz))
    for i = 1:size(test,1)-32
        for k=1:size(test,2)-32
            t = im2double(test(i:i+31,k:k+31));
            %t = histeq(t);
            Y = bdffeature(t);
            Z = transpose(vecf)*(Y-Mf);
            zisq = Z.^2;
            lamda = valf(1:M);
            frac = zisq./lamda;
            fst =  sum(frac);
            ro = (1/(Nf-M))*sum(valf(M+1:Nf));
            snd = (norm(Y-Mf) - sum(zisq))/ro;
            trd = log(prod(valf(1:M)));
            frth = (Nf-M)*log(ro);
            deltf = abs((fst+snd+trd+frth)*10^-7);
            U = transpose(vecn)*(Y-Mn);
            uisq = U.^2;
            lamda = valn(1:M);
            frac = uisq./lamda;
            fst =  sum(frac);
            ep = (1/(Nn-M))*sum(valn(M+1:Nn));
            snd = (norm(Y-Mn) - sum(uisq))/ep;
            trd = log(prod(valn(1:M)));
            frth = (Nn-M)*log(ep);
            deltn = abs((fst+snd+trd+frth)*10^-7);

            if (deltf   > deltn+1.5)
                rectangle("Position",[k i 32 32],...
                    "LineWidth",3, "EdgeColor","b");
            end
        end
    end
    clear test
end
```

MATLAB Code for face detection in color image

```
%% Face detection in color image

clear all
close all
clc

im = imread("....jpg");
im = imresize(im,[340,480]);
```

```
img = rgb2gray(im);
ycbcr = rgb2ycbcr(im);
Cb = ycbcr(:,:,2);
Cr = ycbcr(:,:,3);
for ic = 1:size(im,1)
    for ik = 1:size(im,2)

        if (Cr(ic,ik)>135 && Cr(ic,ik)<165 && Cb(ic,ik)>...
                110 && Cb(ic,ik)<130)
            img(ic,ik) = 255;
        else
            img(ic,ik)=0;
        end
    end
end
clear Cb Cr ycbcr

st = strel("square",15);
img = imerode(img,st);
st = strel("square",3);
img = imdilate(img,st);
imbw = im2bw(img);
[L,n] = bwlabel(imbw,4);
tmp =0;
for ic = 1:n
    cc(ic) = size(find(L==ic),1);
    if tmp < cc(ic)
        indx =ic;
        tmp = cc(ic);
    end
end
figure
imshow(img);
[h1,h2] = find(L==indx);
imCrop = im(min(h1):max(h1),min(h2):max(h2),:);
figure
imshow(imCrop);
ycbcr  = rgb2ycbcr(imCrop);
Cb = im2double(ycbcr(:,:,2));
Cb2 = Cb.^2;
Cr = im2double(ycbcr(:,:,3));
figure
imshow(Cb./Cr,[]); title("cbByCr");
```

```
Cr2 = Cr.^2;
nCr2 = (1-Cr).^2;
figure, imshow(nCr2,[]); title("nCr2");
EyeMapC = (1/3)*(Cb2 + (1-Cr).^2 + (Cb./Cr));
figure
imshow(EyeMapC,[]);title("EyeMapC");
Y = ycbcr(:,:,1);
figure,imshow(Y,[]); title("Y");
Y = Y.*(255/max(max(Y)));
s = strel("ball",5,5);
Yd = imdilate(Y,s);
figure,imshow(Yd,[]);title ("dilate")
Ye = imerode(Y,s);
figure,imshow(Ye,[]);title ("erode")
EyeMapL = Yd./(Ye+1);
figure
imshow(EyeMapL,[]);title("EyeMapL");
EyeMapL = im2double(EyeMapL);
EyeMapL = EyeMapL.*(255./(max(max(EyeMapL))));
EyeMapC = EyeMapC.*(255./(max(max(EyeMapC))));
AndEye = EyeMapL.*EyeMapC;
s = strel("ball",5,5);
Eye = imerode(AndEye,s);
s = strel("ball",11,11);
Eye = imdilate(Eye,s);
figure
imshow(Eye,[]);title("Eye");
figure
imshow(Cb2);title("Cb2");
figure
Cr2 = Cr2.*(255./(max(max(Cr2))));
imshow(Cr2,[]);title("Cr2");
CrByCb = Cr./Cb;
CrByCb = CrByCb.*(255./(max(max(CrByCb))));
figure
imshow(CrByCb,[]);title("CrByCb");
eta = 0.95 * (sum(sum(Cr.^2))/(sum(sum(Cr/Cb))));
MouthMap =Cr2.*(abs(Cr2-eta*CrByCb));
figure, imshow(abs(Cr2-eta*CrByCb),[]),title("Diff");
s = strel("ball",5,5);
Mouth = imerode(MouthMap,s);
s = strel("ball",11,11);
Mouth = imdilate(Mouth,s);
```

```
figure
imshow(Mouth,[]);title("Mouth");
[h7,h8] = find(Mouth==max(max(Mouth)));
mr = h7+min(h1);
mc = h8+min(h2);
figure
imshow(im);hold on
rectangle("Position",[mc-30,mr-10,40,20],...
    "LineWidth",3,"EdgeColor","b")
[h3,h4]=find(Eye==max(max(Eye)));
rer = h3+min(h1);
rec = h4+min(h2);
rectangle("Position",[rec-10,rer-10,20,20],...
    "LineWidth",3,"EdgeColor","r")
Eye(h3-20:h3+20,h4-20:h4+20)=0;
[h5,h6]=find(Eye==max(max(Eye)));
ler = h5+min(h1);
lec = h6+min(h2);
rectangle("Position",[lec-10,ler-10,20,20],...
    "LineWidth",3,"EdgeColor","g")
if lec<rec
    rectangle("Position",[lec-20,ler-30,110,120],...
        "LineWidth",4,"EdgeColor","c")
else
    rectangle("Position",[rec-20,rer-30,110,120],...
        "LineWidth",4,"EdgeColor","c")
end
```

MATLAB code for shading correction

```
%% Shading correction

clear all
close all
clc

im = imread("....jpg");
im = double(im);
IN{1}  = imresize(im,[27,18]);
imshow(IN{1},[]);
MASK = buildmask;
```

```
% Retrieve the indices for the given mask
IND = find(MASK);
figure
% Set up matrices for planar projection calculation
% i.e. Ax = B  so  x = (transpose(A)*A)^-1 * transpose(A)*B
x = 1:1:size(IN{1},2);
y = 1:1:size(IN{1},1);
[mx,my] = meshgrid(x,y);
mxc = mx(IND);
myc = my(IND);
mcc = ones(size(myc));
A = [mxc, myc, mcc];

% Cycle through each image removing shading plane
% and adjusting histogram
for i=1:1

   % Calculate plane: z = ax + by + c
   B = IN{i}(IND);
   x = inv(transpose(A)*A)*transpose(A)*B;
   a = x(1); b = x(2); c = x(3);

   %This is the color plane itself
   SHADING{i} = mx.*a + my.*b + c;
   imshow(SHADING{1},[])
   %This is the image minus the color plane
   %(the constant will be normalized out in histogram recentering)
   OUT{i} = IN{i} - (mx.*a + my.*b + c);

   % Now, recenter the histogram
   maximum = max(max(OUT{i}.*MASK));
   minimum = min(min(OUT{i}.*MASK));   %minimum = min(min(OUT{i}))
   diff = maximum - minimum;
   OUT{i} = ((OUT{i}-minimum)./diff).*MASK;
end
figure
imshow(OUT{1},[]);
Histout = histeq(OUT{1});
figure
imshow(Histout,[])
```

PYTHON Code for AdaBoost classification

```
# AdaBoost classification example

from numpy import *

x= array([[0,1],[1,1],[2,1],[3,-1],[4,-1],\
[5,-1],[6,1],[7,1],[8,1],[9,-1]])
p = array([[0,0.1],[1,0.1],[2,0.1],[3,0.1],\
[4,0.1],[5,0.1],[6,0.1],[7,0.1],[8,0.1],[9,0.1]])
h_final =0
Thres = zeros((4,1))
alpha=zeros((4,1))

for t in range(0,3):
    err= zeros((9,1))
    thr = array([0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5])

    for k in range(0,9):
            if t ==2:
                h = sign(x[:,0]-thr[k])
            else:
                h = sign(thr[k]-x[:,0])

            for j in range(0,10):
                if h[j] != x[j,1]:
                    err[k] = err[k] + p[j,1]

    for l in range(0,9):
        if err[l] == err.min():
            indx = l
            break
    Thres[t]= thr[l]

    if t==2:
        h = sign(x[:,0]-thr[l])
    else:
        h = sign(thr[l]-x[:,0])

    alpha[t] = 0.5 * log((1-err.min())/err.min())
    q1 = exp(-alpha[t])
    q2 = exp(alpha[t])
    Zt = 2*sqrt(err.min()*(1-err.min()))


    for j in range(0,10):
        if h[j] == x[j,1]:
            p[j,1] = (q1*p[j,1])/Zt
        else:
            p[j,1] = (q2*p[j,1])/Zt


    f = alpha[t]*(h)
    h_final = h_final+f

decision = sign(h_final)
print decision
```

PYTHON Code for OpenCV based face detection[1]

```
# OpenCV based face detection
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier \
("haarcascade_frontalface_alt2.xml")
img = cv2.imread("/home/pradipta/PRADIPTA/" \
"Database/CalTechfaces/16.jpg",0)
faces = face_cascade.detectMultiScale(img, 1.3,5)
for (x,y,w,h) in faces:
   cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
   roi_gray = img[y:y+h, x:x+w]
cv2.imshow("Faces",img)
```

MATLAB code for open and close face recognition

```
\texttt{function v2()
% finding recognition rate between test and
```

---

[1] http://docs.opencv.org/trunk/doc/py-tutorials/py-objdetect/py-face-detection/py-face-detection.html

training face database

```
clc
clear all

disp("************ MENU ******************")
disp("***** Choose your Dataset *********")
disp("1. ARFACE 2.FERET 3. Yale 4. ORL 5.VITIMID")
choice=input(" Enter your choice ");

if choice==1
     load("arface.mat");
     noc=30;    %  max number of class considered
     nocA=10;   %  number of class in training set A
     ipc=13;        %  images per class
     imseqA=1:2:13; % image sequence for training class
     imseqB=2:2:10; % image sequence for test class
     m=72;n=96; % mxn is the image size agter being scaled
elseif choice==2
     load("feret.mat");
     noc=30;    %  max number of class considered
     nocA=10;   %  number of class in training set A
     ipc=11;        %  images per class
     imseqA=[5 6 7 8 9]; % image sequence for training class
     imseqB=[1 2 3 4 5]; % image sequence for test class
     m=64;n=43; % mxn is the image size agter being scaled
elseif choice==3
     load("yale.mat");
     noc=15;    %  max number of class considered
     nocA=5;   %  number of class in training set A
     ipc=11;        %  images per class
     imseqA=1:2:11; % image sequence for training class
     imseqB=2:2:11; % image sequence for test class
     m=64;n=64; % mxn is the image size agter being scaled
elseif choice==4
     load("orl.mat");
     noc=40;    %  max number of class considered
     nocA=10;   %  number of class in training set A
     ipc=10;        %  images per class
     imseqA=[1 3 5 7 9]; % image sequence for training class
     imseqB=[2 4 6 8 10]; % image sequence for test class
     m=56;n=46; % mxn is the image size agter being scaled
elseif choice==5
     load("vitimid.mat");
     noc=10;    %  max number of class considered
     nocA=5;   %  number of class in training set A
     ipc=40;        %  images per class
     imseqA=1:2:40; % image sequence for training class
     imseqB=2:2:40; % image sequence for test class
     m=192;n=256; % mxn is the image size agter being scaled

else
     disp(" invlid choice ")
     return;
end


for pca_choice=1:4 % pca_choice==1 indicates PCA
                   % pca_choice==2 indicates 2D PCA
                   % pca_choice==3 indicates Kernel PCA
                   % pca_choice==4 indicates PCA_LDA
    plotCounter=0;
    nop=3;     % number of points within one mst edge

    for nocB=(nocA):noc
        %  number of class in test set B

        % step 0. preliminary calculations
        ipcA=length(imseqA);    %  images per class in  A
        ipcB=length(imseqB);    %  images per class in  B

         % step 1. read the training set A and test set B
         A=[];
         for classA=1:nocA
             for imgA=imseqA
                 col=(classA-1)*ipc+imgA;
                 A=[A set(:,col)];
             end
         end
         B=[];
         for classB=1:nocB
             for imgB=imseqB
                 col=(classB-1)*ipc+imgB;
                 B=[B set(:,col)];
             end
         end
         A=double(A);
```

```
            B=double(B);

            if pca_choice==1
% perform pca on train set A to find face space projection projA
                [ projA projB]=pca(A,B,size(A,2));
            elseif pca_choice==2  % 2D PCA
                % preprocessing
                clear projA;clear projB;

                for j=1:size(A,2)
                    train(:,:,j)=reshape(A(:,j),m,n);
                end
                for j=1:size(B,2)
                    test(:,:,j)=reshape(B(:,j),m,n);
                end
                % 2d pca
                [pA pB]=pca2d(train,test,n);

                 % postprocessing
                 for j=1:size(pA,3)
                     temp=pA(:,:,j);
                     projA(:,j)=temp(:);
                 end
                 for j=1:size(pB,3)
                     temp=pB(:,:,j);
                     projB(:,j)=temp(:);
                 end
            elseif pca_choice==3   % Kernel PCA
                clear projA;clear projB;
                [projA projB]=pcaKernel(A,B,size(A,2));
            elseif pca_choice==4   % PCA LDA
                clear projA;clear projB;
                [projA projB]=pcalda(A,B,noc,ipcA);
            end

            classA=1;
            for imgA=1:ipcA:size(projA,2)
                exist=projA(:,imgA:imgA+ipcA-1);
                % form the weighted graph wg, where
                %weight=distance between nodes
                wg=distMat(exist,exist);
                [cost,next]=prim(wg,1);
% apply prim"s algo to get the MST
```

```
                if nocB<=nocA
                theta(classA)=max(cost);
% threshold for each class
                else

                    theta(classA)=max(cost)/2;
                   % threshold for each class
                end
                classA=classA+1;
               end

            euDist=distMat(projA,projB);
         [correct wrong rate1]=clfr_nn(euDist,nocA,nocB);
 [correct wrong rate2]=clfr_minDist(euDist,nocA,nocB,theta);
 [correct wrong rate3]=clfr_maxHit(euDist,nocA,nocB,theta);

            plotCounter=plotCounter+1;
            nImpost=(nocB-nocA)*ipcB;
            xx(plotCounter)=nImpost;
            if pca_choice==1
                pca1(plotCounter)=rate1;
                pca2(plotCounter)=rate2;
                pca3(plotCounter)=rate3;
            elseif pca_choice==2
                pca2d1(plotCounter)=rate1;
                pca2d2(plotCounter)=rate2;
                pca2d3(plotCounter)=rate3;
            elseif pca_choice==3
                pcak1(plotCounter)=rate1;
                pcak2(plotCounter)=rate2;
                pcak3(plotCounter)=rate3;
            elseif pca_choice==4
                pcalda1(plotCounter)=rate1;
                pcalda2(plotCounter)=rate2;
                pcalda3(plotCounter)=rate3;

            end


       end % next nocB

   end % next pca_choice
```

```
  axis([20 100 0 100]);
  hold all;

  % plot for PCA
  plot(xx,pca1,"-b")  % classifier 1
  plot(xx,pca2,"-r")  % classifier 2


  % plot for 2D PCA
  plot(xx,pca2d1,"-b+")
  plot(xx,pca2d2,"-r+")

  % plot for Kernel PCA
   plot(xx,pcak1,"-b*")
   plot(xx,pcak2,"-r*")

  % plot for PCA LDA
  plot(xx,pcalda1,"-bs")
  plot(xx,pcalda2,"-rs")

 end



function euDist=distMat(A,B)
        for a=1:size(A,2)
           for b=1:size(B,2)
            % eucledian distance
               euDist(a,b)=sum((A(:,a)-B(:,b)).^2).^0.5;
           end
        end
end

function [correct wrong rate]=clfr_nn(euDist,nocA,nocB)
        ipcA=floor(size(euDist,1)/nocA);
        ipcB=floor(size(euDist,2)/nocB);
        correct=0;
        wrong=0;
        for imgB=1:size(euDist,2)
            [val imgA]=min( euDist(:,imgB) );
            classB=floor((imgB-1)/ipcB)+1;
            classA=floor((imgA-1)/ipcA)+1;

            if(classA==classB)
                correct=correct+1;
            else
                wrong=wrong+1;
            end
        end
        rate=(correct*100)/(correct+wrong);
    end

function [correct wrong rate]=clfr_minDist(euDist,nocA,nocB,thresh)
        ipcA=floor(size(euDist,1)/nocA);
        ipcB=floor(size(euDist,2)/nocB);
        correct=0;
        wrong=0;
        for imgB=1:size(euDist,2)
            classB=floor((imgB-1)/ipcB)+1;
  % find all candidate near images of imgB
            count=0;
            for imgA=1:size(euDist,1)
                classA=floor((imgA-1)/ipcA)+1;
                if euDist(imgA,imgB)<thresh(classA)
                    count=count+1;
                    candidateImg(count)=imgA;
                    candidateDist(count)=euDist(imgA,imgB);
                end
            end
            if count==0 && classB>nocA % imposter
                correct=correct+1;
            elseif count==0 && classB<=nocA
                wrong=wrong+1;
            else
                [val ind]=min(candidateDist);
                 minImg=candidateImg(ind);
                 classA=floor((minImg-1)/ipcA)+1;

                 if(classA==classB)
                     correct=correct+1;
                 else
                          wrong=wrong+1;
                 end
            end
        end
        rate=(correct*100)/(correct+wrong);
    end



 function [correct wrong rate]=clfr_maxHit(euDist,nocA,nocB,thresh)
        ipcA=floor(size(euDist,1)/nocA);
        ipcB=floor(size(euDist,2)/nocB);
        correct=0;
        wrong=0;
        for imgB=1:size(euDist,2)
            classB=floor((imgB-1)/ipcB)+1;
            counter=zeros(1,nocA);
            for imgA=1:size(euDist,1)
                classA=floor((imgA-1)/ipcA)+1;
                if euDist(imgA,imgB)<thresh(classA)
                    counter(classA)=counter(classA)+1;
                end

            end
            [val,classA]=max(counter);
 % in which class imgB is classified max no of times
            if (classA==classB) || (val==0 && classB>nocA)
                correct=correct+1;
            else
                   wrong=wrong+1;
            end
        end
        rate=(correct*100)/(correct+wrong);
    end
}
```

MATLAB code for filter functions

```
%% All filter function

function H = Filter(A,alpha,beta,gamma,d1,d2)
d = d1*d2;
```

```
M = zeros(d,1);
S = zeros(d,1);
C = ones(d,1);
noI = size(A,3)

for ic = 1:noI
    f = A(:,:,ic);
    %f = double(f);
    f = imresize(f,[d1 d2]);
    F = fft2(f);
    M = M+F(:);
    S = S+F(:).*conj(F(:));
end
M = M./noI;
S = S./noI;
D = S;
S = S - M.*conj(M);

h = M./(alpha*C+beta*D+gamma*S);
H = reshape(h,d1,d2);
```

MATLAB code for PSR calculation

```
%% Function for PSR calculation
function [out] = PsrCalculation(Corr)
peak = max(max(Corr));
[h1,h2]= find(Corr==peak);
a = size(Corr,1);
b = size(Corr,2);
if h1>10 && h2>10 && h1<a-10 && h2<b-10
    Corr(ceil(h1-2):ceil(h1+2),ceil(h2-2):ceil(h2+2))=0;
    Mask = Corr(h1-10:h1+10,h2-10:h2+10);
    cnt = 1;
    for ic = 1:size(Mask,1)
        for ik= 1:size(Mask,2)
            if Mask(ic,ik) == 0

            else
                Annular(cnt,:)=Mask(ic,ik);
                cnt=cnt+1;
            end
        end
    end
    mn = mean(Annular);
    st = std(Annular);
    psr = (peak-mn)/st;
else
    psr=0;
end
out = psr;
```

MATLAB code for correlation filter for face recognition

```
%% Correlation filter for face recognition
%% UMACE, MACH, OTMACH etc..
close all
clear all
clc

ss1 =[1 7 8 9 37 38 36];
ss2 = [5 11 12 13 15 39 40 41 42 44 10 2];
ss3 =[3 6 14 16 17 19 20 45 48 49 43 46];
ss4 = [18 21 22 23 24 25 26 50 51 52 53 54];
ss5 =[4 35 29 30 31 32 33 34 28 27 64 63 62 61 56 57 58 59 60];

for ic = 1:64
    alltest(ic)=ic;
end

trainSet = ss5;
testSet = alltest;

d1 = 100; d2 = 100;
PsrUmace = zeros(size(testSet,2),10);
PsrMach = zeros(size(testSet,2),10);
PsrOtmach = zeros(size(testSet,2),10);
for class = 1:10

    for ic = 1:size(trainSet,2)
        A(:,:,ic) = imread(strcat("yaleB",num2str(class),...
        "_",num2str(trainSet(:,ic)),".pgm"));
    end
    Umace = Filter(A,0,1,0,d1,d2);
    Mach = Filter(A,0,0,1,d1,d2);
    Otmach = Filter(A,0.9,0.9,0.8,d1,d2);


    for ic = 1:size(testSet,2)
        t = imread(strcat("yaleB",num2str(class),...
        "_",num2str(testSet(:,ic)),".pgm"));
        %t = double(t);
        t = imresize(t,[d1 d2]);
        T = fft2(t);
        corr = real(fftshift(ifft2(T.*conj(Umace))));
        psr = PsrCalculation(corr);
        PsrUmace(ic,class) = psr;
        clear corr psr
        corr = real(fftshift(ifft2(T.*conj(Mach))));
        psr = PsrCalculation(corr);
        PsrMach(ic,class) = psr;
        clear psr corr
        corr = real(fftshift(ifft2(T.*conj(Otmach))));
        psr = PsrCalculation(corr);
        PsrOtmach(ic,class) = psr;

    end
end
AvgPsrUmace = mean(PsrUmace,2);
AvgPsrMach = mean(PsrMach,2);
AvgPsrOtmach = mean(PsrOtmach,2);

% Class specific PCA
for trainPerson=1:10
    T=[];
    No_of_Training_Images=size(trainSet,2); % for each class
    C=1;
    for h=1:No_of_Training_Images
        hh=int2str(trainPerson);
        kk=int2str(trainSet(:,h));
        b=strcat("yaleB",hh,"_",kk);
        img=imread(strcat(b,".pgm"));
        f = double(img);
        f = imresize(f,[100 100]);
        F = fft2(f);
        Df = F(:).*conj(F(:));
        hf = F(:)./Df;
        Pf = exp(1j.*angle(hf));
        [m1,n1]=size(f);

        T=[T Pf];
        C=C+1;
    end
    % Number of classes (or persons)
    Class_number = ( size(T,2) )/(C-1);
    % Number of images in each class
    Class_population = C-1;
    % Total number of training images
    P = Class_population * Class_number;

    % figure(1)
    m_total=mean(T,2);
    Mimg=reshape(m_total,m1,n1);

    %imshow(Mimg,[]);title("MEAN IMAGE");

    Difference=[];
    for ic=1:size(T,2)
        diff=T(:,ic)-m_total;
        Difference=[Difference diff];
    end

    Covar=Difference"*Difference;
    [U,E,V]=svd(Covar);
    val=diag(E);

    figure(3)
    stem(val);title("EIGEN VALUE");
    drawnow;
    Eigen_Vector=Difference*U;
    Eigen_Vector=U;
    figure(4)
    for ic=1:size(U,2)
         Eigen_Face=Eigen_Vector(:,ic);
         Eigen_Face_Image=reshape(Eigen_Face,m1,n1);
         subplot(ceil(sqrt(size(U,2))),ceil(sqrt(size(U,2))),ic);
         imshow(Eigen_Face_Image,[]);
         drawnow;
```

```
 end

PC=Eigen_Vector;
for ic=1:size(PC,2)
    PC(:,ic)=PC(:,ic)./norm(PC(:,ic));
end

%%% Weight calculation of Training Images

ProjectedImages_PCA = [];
for ic = 1 : P
    temp = transpose(PC)*Difference(:,ic);
    ProjectedImages_PCA = [ProjectedImages_PCA temp];
end

% Reconstruction by PCA
clear f

testPerson = trainPerson;
for ic = 1:size(testSet,2) %person index
    hh=int2str(testPerson);
    k = testSet(:,ic);
    kk=int2str(k);
    b=strcat("yaleB",hh,"_",kk);
    f=imread(strcat(b,".pgm"));

    f=double(f);
    f= imresize(f,[100 100]);
    F = fft2(f);
    Df = F(:).*conj(F(:));
    hf = F(:)./Df;
    Pf = exp(1j.*angle(hf));
             figure(1)
             imagesc(L),colormap(gray);
             figure(1)
             imagesc(L),colormap(gray); title("Test Image");
    diff=Pf-m_total;
    projected=transpose(PC)*diff;
    reconstructed=m_total+PC*projected;
    Recn=reshape(reconstructed,m1,n1);
    Pf = reshape(Pf,[100 100]);

    corr = real(fftshift(ifft2(Pf.*conj(Recn))));
```

```
        %figure, surf(corr);view([-34 10])
        psr = PsrCalculation(corr);
        PSR(ic,testPerson) = psr;

    end
end
AvgPsrCsPca = mean(PSR,2);
AvgPSR = [AvgPsrUmace AvgPsrMach AvgPsrOtmach AvgPsrCsPca];
plot(AvgPSR),axis([0 64 0 100])
```

MATLAB code for Coreface

```
%% Coreface matlab program

clear all
close all
clc
%% NO NORMALIZATION HAVE BEEN DONE

% dir = "F:\croppedyale";
% cd(dir);
PSR = zeros(64,10);
o =imread("NLP1_1.jpg");
%imagesc(o);colormap(gray)

ss1 =[1 7 8 9 37 38 36];
ss2 = [5 11 12 13 15 39 40 41 42 44 10 2];
ss3 =[3 6 14 16 17 19 20 45 48 49 43 46];
ss4 = [18 21 22 23 24 25 26 50 51 52 53 54];
ss5 =[4 35 29 30 31 32 33 34 28 27 64 63 62 61 56 57 58 59 60];

ss6 = [ 1 5 3 18 4];
ss7 =[1 4 40 54 25];
for Knownclass=1:1
    Knownclass
    T=[];
    No_of_Training_Images=size(ss1,2); % for each class
    C=1;
        for h=1:No_of_Training_Images
            hh=int2str(Knownclass);
            kk=int2str(ss1(:,h));
            b=strcat("yaleB",hh,"_",kk);
            img=imread(strcat(b,".pgm"));
            f = double(img);
            f = imresize(f,[100 100]);
            F = exp(1j.*angle(fft2(f)));
            [m1,n1]=size(f);
            figure(1);
            subplot(ceil(sqrt(12)),ceil(sqrt(12)),C);
            imshow(img);
            if h==3
                title("TRAINING IMAGES");
            end
            T=[T F(:)];
            C=C+1;
        end

    Class_number = ( size(T,2) )/(C-1);
    Class_population = C-1;
    P = Class_population * Class_number;
    m_total=mean(T,2);
    Mimg=reshape(m_total,m1,n1);
    %imshow(Mimg,[]);title("MEAN IMAGE");

    Difference=[];
    for ic=1:size(T,2)
        diff=T(:,ic)-m_total;
        Difference=[Difference diff];
    end

    Covar=transpose(Difference)*Difference;
    [U,E,V]=svd(Covar);
    val=diag(E);

    figure(3)
    stem(val);title("EIGEN VALUE");
    drawnow;
    Eigen_Vector=Difference*U;
     Eigen_Vector=U;
     figure(4)
     for ic=1:size(U,2)
         Eigen_Face=Eigen_Vector(:,ic);
         Eigen_Face_Image=reshape(Eigen_Face,m1,n1);
         subplot(ceil(sqrt(size(U,2))),ceil(sqrt(size(U,2))),ic);
         imshow(Eigen_Face_Image,[]);

         drawnow;
     end

    PC=Eigen_Vector;
    for ic=1:size(PC,2)
        PC(:,ic)=PC(:,ic)./norm(PC(:,ic));
    end
    %% Weight calculation of Training Images

    ProjectedImages_PCA = [];
    for ic = 1 : P
        temp = transpose(PC)*Difference(:,ic);
        ProjectedImages_PCA = [ProjectedImages_PCA temp];
    end

    %Reconstruction by PCA

    person = 1;
    PSR = [];
      testSet = ss2;
        for  ic = 1:size(testSet,2) %person index
            hh=int2str(person);
            k = testSet(:,ic);
            kk=int2str(k);
            b=strcat("yaleB",hh,"_",kk);
            f=imread(strcat(b,".pgm"));
            f=double(f);
            f= imresize(f,[100 100]);
            F = exp(1j.*angle(fft2(f)));
            %         figure(1)
            %         imagesc(L),colormap(gray);
            %         figure(1)
            %         imagesc(L),colormap(gray); title("Test Image");
            diff=F(:)-m_total;
            projected=transpose(PC)*diff;
            reconstructed=m_total+PC*projected;
            Recn=reshape(reconstructed,m1,n1);
            corr = real(fftshift(ifft2(Recn.*conj(F))));
            %figure,surf(corr);view([-34 10])
            psr = PsrCalculation(corr);
```

```
            PSR = [PSR;psr];

        end
    end
PSR
```

MATLAB code for uncostrained video filter

```
%% Unconstrained Video Filter

clear all
close all
clc
warning off
numVolumes = 20;
volumes = cell(1, numVolumes);

figure(1); cn=1;
for v = 1: 20%numVolumes
    inFile = sprintf("FTrain%d.avi", v);
    ifp = aviinfo(inFile);
    volume = zeros(ifp.Height, ifp.Width, ifp.NumFrames, "uint8");
    for f = 1 : ifp.NumFrames
        frame = aviread(inFile, f);
        rgbImg = frame.cdata;
        grayImg = rgb2gray(rgbImg);
        edgeImg = sobel(grayImg);
        volume(:,:,f) = edgeImg;
        imshow(volume(:,:,f));
        pause(0.0000002)
    end
    volumes{cn} = volume;
    cn=cn+1;
end

%% Make 3D Filter

[imgRows imgCols timeSamples] = size(volumes{1});
d = imgRows * imgCols * timeSamples;
N = length(volumes);
```

```
x = zeros(d, N);
for i = 1 : N
    fft_volume = fft3(double(volumes{i}),[imgRows  imgCols  timeSamples]);
    x(:,i) = fft_volume(:);
end
clear volumes;
mx = mean(x, 2);
c = ones(d,1);  % 2 * ones(d,1);
dx = mean(conj(x) .* x, 2);
temp = x - repmat(mx, 1, N);
sx = mean(conj(temp) .* temp, 2);

alpha = 0.9%0.1%0.01;  % 0.05; 1e-3; %0.05; % 0.01;
beta =  0.0000000000000009 % 1e-15; % 1e-12;    % 0.3
gamma = 0.00000000000000006; % 1e-12;  0.1;
h_den = (alpha * c) + (beta * dx) + (gamma * sx);
h = mx ./ h_den;
h = reshape(h, [imgRows, imgCols, timeSamples]);
h = real(ifft3(h));
h = uint8(scale(h, min3(h), max3(h), 0, 255));
UVF = h;
save UVF.mat UVF
%% Save 3D MACH as a short movie clip

outFile = "UVF.avi";
mov = avifile(outFile, "COMPRESSION", "None", "FPS", ifp.FramesPerSecond,
"QUALITY", 100);
% "Indeo5" is better and offer more compression than "Cinepak"
figure(2);
for f = 1 : ifp.NumFrames
    rgbMACH = cat(3, UVF(:,:,f), UVF(:,:,f), UVF(:,:,f));
    m = im2frame(rgbMACH);
    mov = addframe(mov, m);
    imshow(rgbMACH);
    pause(0.08);
end
mov = close(mov);
clear c
```

MATLAB code for DCCF for multiclass pattern recognition

```
%% DCCF for multiclass pattern recognition

close all
clear all
clc

NoC = 3;
NoI = 14;
M = zeros(4096,NoC);
D = M;
S = M;
ic=1;
for ih = 1:NoC
    for ik = ih*14-14+1:ih*14-14+NoI
        f = imread(strcat(num2str(ik),".jpg"));
        %f = im2double(f);
        imshow(f);pause(0.2)
        F = fft2(f);
        M(:,ic) = M(:,ic) + F(:);
        D(:,ic) = D(:,ic) + F(:).*conj(F(:));
    end
    M(:,ic) = M(:,ic)/NoI;
    S(:,ic) = D(:,ic)./NoI;
    S(:,ic) = S(:,ic)-M(:,ic).*conj(M(:,ic));
    ic = ic+1;
end

Stotal = zeros(4096,1);
Mtotal = Stotal;
for ic = 1:NoC

    Stotal = Stotal+S(:,ic);
    Mtotal = Mtotal + M(:,ic);
end

Stotal = Stotal/NoC;
Mtotal = Mtotal/NoC;

% Formulating E
E =[];
for ic = 1:NoC
    e = Mtotal - M(:,ic);
    E = [E e];
```

```
end

% Formulating V
V = (conj(E))"*E;

% Eigens of V

[P,val,Q]=svd(V);

% Calculating Phi
Phi = E*P*(val)^(-0.5);

% calculating a
term =[];
for ic = 1: size(Phi,2)
    t = Phi(:,ic)./Stotal;
    term = [term t];
end

temp = val*transpose(conj(Phi))*term;
[a,lambda,a1] = svd(temp);
amax = a(:,1);

% Calculating h
h = (Phi*amax)./Stotal;

for ic = 1:NoC

    bf = (M(:,ic).*conj(h));
    b(:,ic) = transpose(bf)*bf/4096;
end
for ic = 1:NoC
    hf(:,ic) = h.*conj(h).*M(:,ic);
    H(:,:,ic) = reshape(hf(:,ic),64,64);
end
%save DCCF.mat H h b

Dist =[];
r=1;
c=1;
for ih = 1:NoC
    for ik = ih*14-14+1:ih*14-14+14
        test = imread(strcat(num2str(ik),".jpg"));
        imshow(test,[]);pause(0.02);
        ftest = fft2(test);
        z = ftest(:);

        p = (z.*conj(h));
        p = transpose(p)*p/4096;

        for ic = 1:NoC
            g = real(fftshift(ifft2(ftest.*conj(H(:,:,ic)))));
            d  = p + b(:,ic) - 2* max(g(:));

            Dist =[Dist d];
        end
        indx = find(Dist==min(Dist));
        Index(r,c) = indx;
        r = r+1;
        Dist=[];
    end
    c=c+1;
    r=1;
end
Index
```

MATLAB code for synthetic face generation

```
\texttt{% finding recognition rate between test and
   %training (existing plus synthetic) images
 of FERET face database
   clc
   clear all

   % STEP 1: Read training and test set

   % INPUT
   nPointRange=1; % number of points taken
within one mst edge should be varied in this range
   nClass=50; % total number of classes
   nImg=11; % total number of images in each class
   m=64;n=43; % size of the image is mXn
```

```
after being scaled by 1/3
   totImgId=[1 2 3 4 5 6 7 8 9 10 11];
   nImgTrain=6; % number of images in each train class
   nImgTest=5; % number of images in each test class
   nCombination=20;

   load("feret.mat");
% load the dataset in the variable "set"

   % find all combinations of training and test set
   trainImgId=nchoosek(totImgId,nImgTrain);
   for row=1:size(trainImgId,1)
       testImgId(row,:)=setdiff(totImgId,trainImgId(row,:));
   end
   resultSynth=[];resultClub=[];

   %nCombination=size(trainImgId,1);
 % maximum number of combinations
   for row=1:nCombination
       tic
       col=1;
       fprintf("\n *******
 %d th COMBINATION of train and test set *******\n",row);
   trainImgId(row,:)
   testImgId(row,:)
   % Extract train and test set from "set"
   train=extractSet(set,1:1:nClass,trainImgId(row,:),nImg);
   test=extractSet(set,1:1:nClass,testImgId(row,:),nImg);

       % STEP 2: Perform PCA on train
and test set and find out the recognition rate
using nearest-neighbor.
       [projTrain projTest]=pca(train,test,size(train,2));
       fprintf
       ("\n\n Recognition rate without using synthesized images :\n")
       [correct,wrong,rate]=nearNeighbor(projTrain,projTest,nClass)
       resultSynth(row,col)=rate;
resultClub(row,col)=correct;resultClub(row,col+1)
=wrong;resultClub(row,col+2)=rate;

       % STEP 3: For each class of the training set
 form an MST using Prim’s algorithm, considering each
       % image of the class as a node.
       % Now, generate "nPoint" number of
EQUIDISTANT synthetic image within each edge.
       % Combine the synthetic images with
 existing ones to form the clubbed set and
       % find out the recognition rate between
 clubbed set and test set using nearest-neighbour.
       for nPoint=nPointRange
           col=col+1;
           projSynth=[];
 % projection matrix of synthetic set
         projClub=[];
  % projection matrix of clubbed (existing + synthetic) set
           for class=1:nClass
               imaginary=[];
% synthetic space projection for single class "class"
               img=0;
% not necessary : used as image index
only for  writing purpose
               lowImg=(class-1)*nImgTrain+1;
% lower bound of image in class "class"
               upImg=lowImg+nImgTrain-1;
 % upper bound of image in class "class"
% already existing space projection for single class "class"
               exist=projTrain(:,lowImg:upImg);
               existDist=edistMat(exist);
               [cost,next]=prim(existDist,1);
% apply prims algo to get the MST
               for node=2:nImgTrain
% next(1) is always zero
                   nextNode=next(node);
 % node and nextNode are two vertices of the mst edge
                   lambda=0.0;
                   for point=1:nPoint
                       lambda=lambda+1/(nPoint+1);
  % this is a synthetic node (image point)
 between node and nextNode
restore1=lambda*exist(:,node)+(1-lambda)*exist(:,nextNode);
             imaginary=[imaginary restore1];
imwrite(restore4,path);
               end % next point
               end % next node
               threshold(class)=max(cost)/2.0;
               projSynth=[projSynth imaginary];
               projClub=[projClub exist imaginary];
           end % next class
           fprintf("\n WHEN NUMBER OF POINTS (IMAGES)
 TAKEN WITHIN ONE EDGE OF MST IS = %d",nPoint)
fprintf("\n\n the recognition rate using original plus
synthesized images :\n")
   [correct,wrong,rate]=nearNeighbor(projClub,projTest,nClass)
resultClub(row,col+2)=correct;resultClub(row,col+3)=wrong;
resultClub(row,col+4)=rate;
       end % now vary number of synthetic points
       fprintf(" time taken in
%d th combination out of total %d combinations is",row,nCombination);
       toc
   end % next combination

 resultClub

y=resultClub(:,2)
z=resultClub(:,5)

ybar=mean(y);zbar=mean(z);
stdDev_y=sqrt(var(y))
stdDev_z=sqrt(var(z))
n1=length(y)
n2=length(z)
est=sqrt(var(y)/n1+var(z)/n2)
g1=var(y)/n1;
g2=var(z)/n2;
df_nume=(g1+g2)^2*(n1-1)
df_deno=g1^2+g2^2
df=df_nume/df_deno
t=(ybar-zbar)/est}
```