LAB ASSIGNMENT 3

UCS522: Computer Vision

SUBMITTED TO:
Dr. Shailendra Tiwari

SUBMITTED BY:
Pranav Kakkar (401703019)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

PATIALA, PUNJAB

Develop an efficient Face recognition system using Principal Component Analysis (PCA)

Algorithm:-

1. Compute the mean feature vector

   $\mu = \frac{1}{p}\sum_{k=1}^{p} x_k$, where, x_k is a pattern (k = 1 to p), p = number of patterns, x is the feature matrix

2. Find the covariance matrix

   $C = \frac{1}{p}\sum_{k=1}^{p}\{x_k - \mu\}\{x_k - \mu\}^T$ where, T represents matrix transposition

3. Compute Eigen values λ_i and Eigen vectors v_i of covariance matrix

   $Cv_i = \lambda_i v_i$ $\quad$ (i = 1, 2, 3,....q), q = number of features

4. Estimating high-valued Eigen vectors
   (i) Arrange all the Eigen values (λ_i) in descending order
   (ii) Choose a threshold value, θ
   (iii) Number of high-valued λ_i can be chosen so as to satisfy the relationship

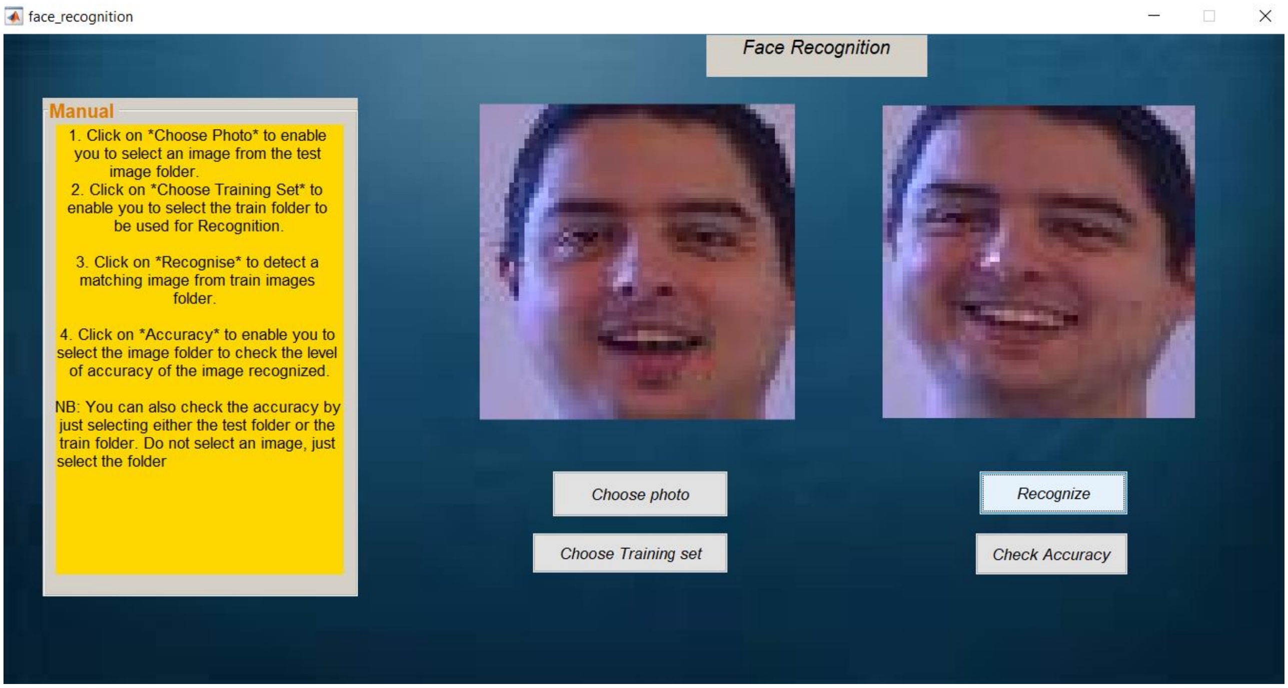
   $$\left(\sum_{i=1}^{s}\lambda_i\right)\left(\sum_{i=1}^{q}\lambda_i\right)^{-1} \geq \theta$$, where, s = number of high valued λ_i chosen

   (iv) Select Eigen vectors corresponding to selected high valued λ_i

5. Extract low dimensional feature vectors (principal components) from raw feature matrix.

   $P = V^T x$, where, V is the matrix of principal components and x is the feature matrix

Results:-



Explanation: We can see from the various tests made that we were able to detect images based on corresponding features. Some particular images had the similar features with other images and the accuracy found was 80%. This could be due to the face that they made similar gestures or they had the same dimensions. Other tests showed great results by identifying similar image and gave an accuracy of 100%. We realized that the recognition was based on similar gestures. This could be the smile made or the raising of the eye browse and could also be the wideness of the face. There could be many other reasons but in this case we know this was done due to the features corresponding to the sample images given. The level of accuracy given depends on the total number of database in which we have. Less database gives gives low accuracy level but the more the images in the test folder the higher that accuracy level.

Code:-

```
function varargout = face_recognition(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@face_recognition_OpeningFcn, ...
```

```
                   'gui_OutputFcn',
@face_recognition_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before face_recognition is made visible.
function face_recognition_OpeningFcn(hObject, eventdata,
handles, varargin)
handles.output = hObject;
% create an axes that spans the whole gui
ah = axes('unit', 'normalized', 'position', [0 0 1 1]);
% import the background image and show it on the axes
bg = imread('wall.jpg'); imagesc(bg);
% prevent plotting over the background and turn the axis off
set(ah,'handlevisibility','off','visible','off')
% making sure the background is behind all the other
uicontrols
uistack(ah, 'bottom');

% Update handles structure
guidata(hObject, handles);



% --- Outputs from this function are returned to the command
line.
function varargout = face_recognition_OutputFcn(hObject,
eventdata, handles)
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
global im;
[filename, pathname] = uigetfile({'*.jpg'},'choose photo');
str = [pathname, filename];
```

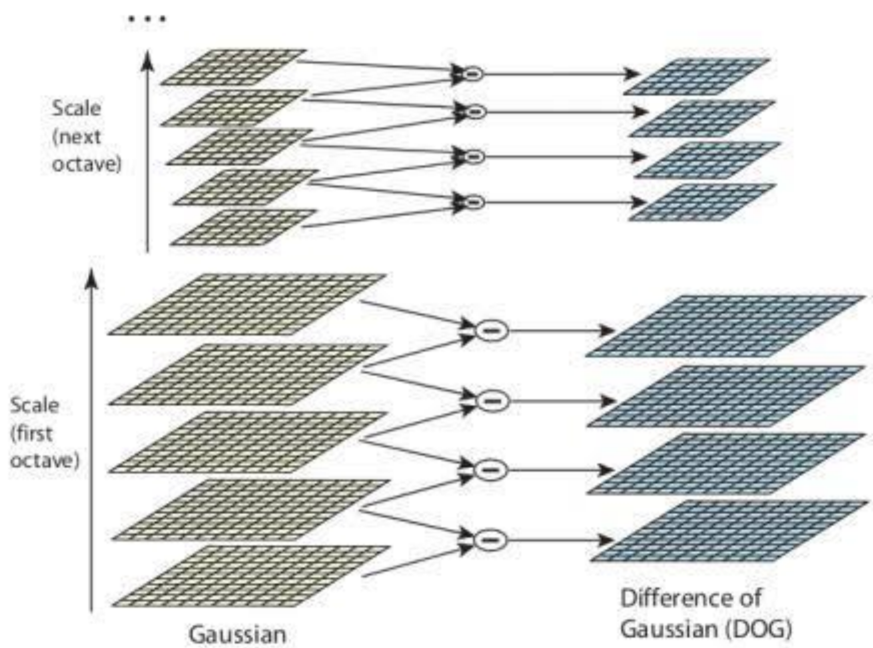
```
im = imread(str);
axes( handles.axes1);
imshow(im);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)

global im
global reference
global W
global imgmean
global col_of_data
global pathname
global img_path_list


im = double(im(:));
objectone = W'*(im - imgmean);
distance = 100000000;


for k = 1:col_of_data
    temp = norm(objectone - reference(:,k));
    if(distance>temp)
        aimone = k;
        distance = temp;
        aimpath = strcat(pathname, '/',
img_path_list(aimone).name);
        axes( handles.axes2 )
        imshow(aimpath)
    end
end
pathname = uigetdir;
img_path_list = dir(strcat(pathname,'\*.jpg'));
img_num = length(img_path_list);
imagedata = [];
if img_num >0
    for j = 1:img_num
        img_name = img_path_list(j).name;
        temp = imread(strcat(pathname, '/', img_name));
        temp = double(temp(:));
        imagedata = [imagedata, temp];
```

```
    end
end
col_of_data = size(imagedata,2);


imgmean = mean(imagedata,2);
for i = 1:col_of_data
    imagedata(:,i) = imagedata(:,i) - imgmean;
end
covMat = imagedata'*imagedata;
[COEFF, latent, explained] = pcacov(covMat);


i = 1;
proportion = 0;
while(proportion < 95)
    proportion = proportion + explained(i);
    i = i+1;
end
p = i - 1;
global W
global reference
col_of_data = 30;

pathname = uigetdir;
img_path_list = dir(strcat(pathname,'\*.jpg'));
img_num = length(img_path_list);
testdata = [];
if img_num >0
    for j = 1:img_num
        img_name = img_path_list(j).name;
        temp = imread(strcat(pathname, '/', img_name));
        temp = double(temp(:));
        testdata = [testdata, temp];
    end
end


col_of_test = size(testdata,2);
testdata = center( testdata );
object = W'* testdata;

error = 0;
for j = 1:col_of_test
    distance = 1000000000000;
    for k = 1:col_of_data;
        temp = norm(object(:,j) - reference(:,k));
        if(distance>temp)
```

```
            aimone = k;
            distance = temp;
        end
    end
    if ceil(j/3)==ceil(aimone/4)
       error = error + 1;
    end
end
% calculating the accuracy
accuracy = ((1-(error/col_of_test))*100);
msgbox(['Accuracy level is :    ',
num2str(accuracy),sprintf('%%')],'accuracy')




% --- Executes during object creation, after setting all
properties.
function listbox3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: listbox controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Scale Invariant Feature Transform (SIFT)

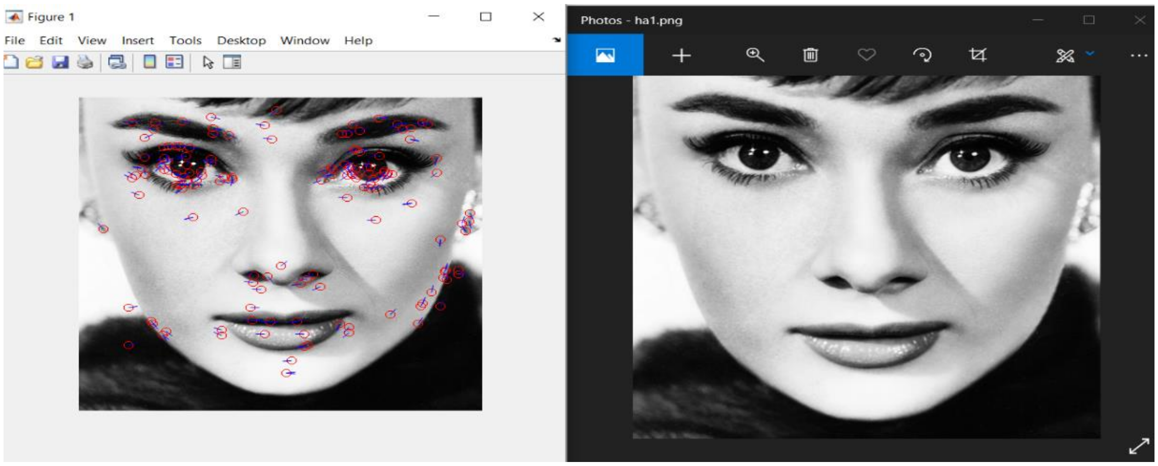
Algorithm:-



In general, SIFT algorithm can be decomposed into four steps:

1. Feature point (also called keypoint) detection
2. Feature point localization
3. Orientation assignment
4. Feature descriptor generation.

Result:-



Code:-

```
clear;
clc;
image = imread('images/ha1.png');
image = rgb2gray(image);
image = double(image);
keyPoints = SIFT(image,3,5,1.3);
image = SIFTKeypointVisualizer(image,keyPoints);
imshow(uint8(image))
function Descriptors = SIFT(inputImage, Octaves, Scales, Sigma)
% This function is to extract sift features from a given image

   %% Setting Variables.
   Sigmas = sigmas(Octaves,Scales,Sigma);
   ContrastThreshhold = 7.68;
   rCurvature = 10;
   G = cell(1,Octaves); % Gaussians
   D = cell(1,Octaves); % DoG
   GO = cell(1,Octaves); % Gradient Orientation
   GM = cell(1,Octaves); % Gradient Scale
   P = [];
   Descriptors = {}; % Key Points

   %% Calculating Gaussians
   for o = 1:Octaves
      [row,col] = size(inputImage);
      temp = zeros(row,col,Scales);
      for s=1:Scales
         temp(:,:,s) = imgaussfilt(inputImage,Sigmas(o,s));
      end
      G(o) = {temp};
      inputImage = inputImage(2:2:end,2:2:end);
```

```
    end

    %% Calculating DoG
    for o=1:Octaves
        images = cell2mat(G(o));
        [row,col,Scales] = size(images);
        temp = zeros([row,col,Scales-1]);
        for s=1:Scales-1
            temp(:,:,s) = images(:,:,s+1) - images(:,:,s);
        end
        D(o) = {temp};
    end

    %% Calculating orientation of gradient in each scale
    for o = 1:Octaves
        images = cell2mat(G(o));
        [row,col,Scales] = size(images);
        tempO = zeros([row,col,Scales]);
        tempM = zeros([row,col,Scales]);
        for s = 1:Scales
            [tempM(:,:,s),tempO(:,:,s)] = imgradient(images(:,:,s));
        end
        GO(o) = {tempO};
        GM(o) = {tempM};
    end

    %% Extracting Key Points
    for o=1:Octaves
        images = cell2mat(D(o));
        GradientOrientations = cell2mat(GO(o));
        GradientMagnitudes = cell2mat(GM(o));
        [row,col,Scales] = size(images);
        for s=2:Scales-1
            % Weight for gradient vectors
            weights = gaussianKernel(Sigmas(o,s));
            radius = (length(weights)-1)/2;
            for y=14:col-12
                for x=14:row-12
                    sub = images(x-1:x+1,y-1:y+1,s-1:s+1);
                    if sub(2,2,2) > max([sub(1:13),sub(15:end)]) || sub(2,2,2) <
min([sub(1:13),sub(15:end)])
                        % Getting rid of bad Key Points
                        if abs(sub(2,2,2)) < ContrastThreshhold
                            % Low contrast.
                            continue
                        else
                            % Calculating trace and determinant of hessian
```

```
% matrix.
Dxx = sub(1,2,2)+sub(3,2,2)-2*sub(2,2,2);
Dyy = sub(2,1,2)+sub(2,3,2)-2*sub(2,2,2);
Dxy = sub(1,1,2)+sub(3,3,2)-2*sub(1,3,2)-2*sub(3,1,2);
trace = Dxx+Dyy;
determinant = Dxx*Dyy-Dxy*Dxy;
curvature = trace*trace/determinant;
if curvature > (rCurvature+1)^2/rCurvature
    % Not a corner.
    continue
end
end
%% Calculating orientation and magnitude of pixels at key point vicinity
% Fixing overflow key points near corners and edges
% of image.
a=0;b=0;c=0;d=0;
if x-1-radius < 0;a = -(x-1-radius);end
if y-1-radius < 0;b = -(y-1-radius);end
if row-x-radius < 0;c = -(row-x-radius);end
if col-y-radius < 0;d = -(col-y-radius);end
tempMagnitude = GradientMagnitudes(x-radius+a:x+radius-c,y-
radius+b:y+radius-d,s).*weights(1+a:end-c,1+b:end-d);
tempOrientation = GradientOrientations(x-radius+a:x+radius-c,y-
radius+b:y+radius-d,s);
[wRows, wCols] = size(tempMagnitude);
% 36 bin histogram generation.
gHist = zeros(1,36);
for i = 1:wRows
    for j = 1:wCols
        % Converting orientation calculation window
        temp = tempOrientation(i,j);
        if temp < 0
            temp = 360 + temp;
        end
        bin = floor(temp/10)+1;
        gHist(bin) = gHist(bin) + tempMagnitude(i,j);
    end
end
%% Extracting keypoint coordinates
% TODO: Interpolation for X and Y value to get
% subpixel accuracy.
%% Extracting keypoint orientation
% Marking 80% Threshold
orientationThreshold = max(gHist(:))*4/5;
tempP = [];
for i=1:length(gHist)
    if gHist(i) > orientationThreshold
```

```
                    % Connrection both ends of the histogram
                    % for interpolation
                    if i-1 <= 0
                        X = 0:2;
                        Y = gHist([36,1,2]);
                    elseif i+1 > 36
                        X = 35:37;
                        Y = gHist([35,36,1]);
                    else
                        X = i-1:i+1;
                        Y = gHist(i-1:i+1);
                    end
                    % interpolation of Orientation.
                    dir = interpolateExterma([X(1),Y(1)],[X(2),Y(2)],[X(3),Y(3)])*10; %
Orientation
                    mag = gHist(i); % Size
                    % Filtering points with the same
                    % orientation.
                    if ismember(dir,tempP(5:6:end)) == false
                        tempP = [tempP,x,y,o,s,dir,mag];
                    end
                end
            end
            P = [P,tempP];
        end
    end
  end
 end
end

%% Creating feature Descriptors
% TODO: Extract Descriptors
weights = gaussianKernel(Sigmas(o,s),13);
weights = weights(1:end-1,1:end-1);
for i = 1:6:length(P)
    x = P(i);
    y = P(i+1);
    oct = P(i+2);
    scl = P(i+3);
    dir = P(i+4);
    mag = P(i+5);
    directions = cell2mat(GO(oct));
    directions = directions(x-13:x+12,y-13:y+12,scl);
    magnitudes = cell2mat(GM(oct));
    magnitudes = magnitudes(x-13:x+12,y-13:y+12,scl).*weights;
    descriptor = [];
    for m = 5:4:20
```

```
        for n = 5:4:20
            hist = zeros(1,8);
            for o = 0:3
                for p = 0:3
                    [newx,newy] = rotateCoordinates(m+o,n+p,13,13,-dir);
                    % Creating 8 bin histogram.
                    hist(categorizeDirection8(directions(newx,newy))) =
magnitudes(newx,newy);
                end
            end
            descriptor = [descriptor, hist];
        end
    end
    descriptor = descriptor ./ norm(descriptor,2);
    for j =1:128
        if descriptor(j) > 0.2
            descriptor(j) = 0.2;
        end
    end
    descriptor = descriptor ./ norm(descriptor,2);
    % Creating keypoint object
    kp = KeyPoint;
    kp.Coordinates = [x*2^(oct-1),y*2^(oct-1)];
    kp.Magnitude = mag;
    kp.Direction = dir;
    kp.Descriptor = descriptor;
    kp.Octave = oct;
    kp.Scale = scl;
    Descriptors(end+1) = {kp};
  end
end

%% Function to extract Sigma values
function matrix = sigmas(octave,scale,sigma)
% Function to calculate Sigma values for different Gaussians
  matrix = zeros(octave,scale);
  k = sqrt(2);
  for i=1:octave
    for j=1:scale
      matrix(i,j) = i*k^(j-1)*sigma;
    end
  end
end

%% Calculating Gaussian value given SD
function result = gaussianKernel(SD, Radius)
% Returns a gaussian kernet
```

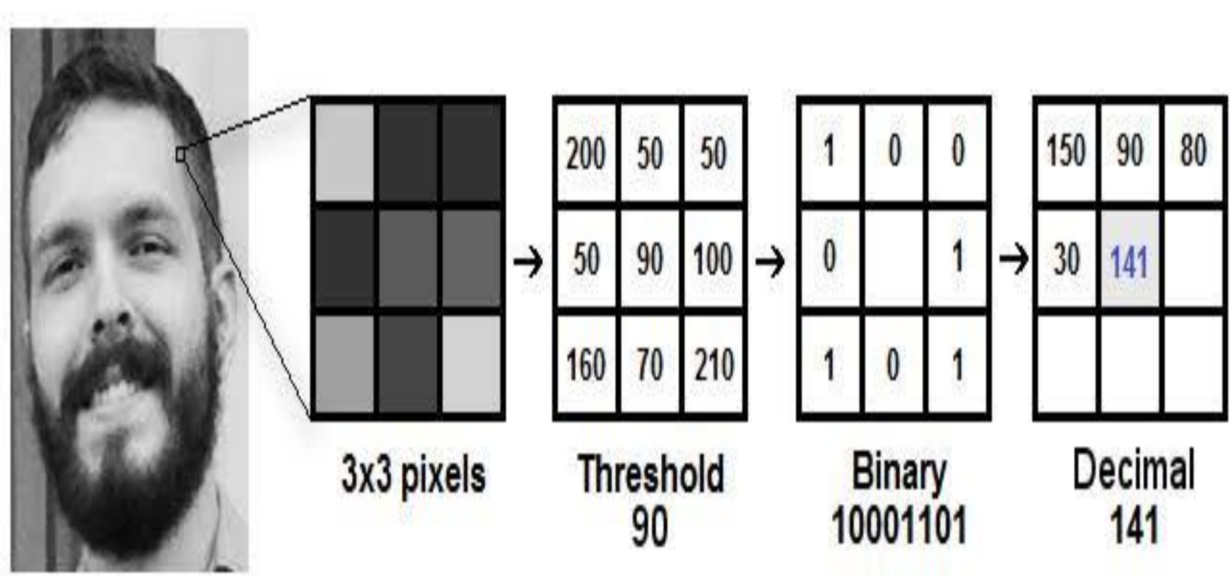
```
% By default a radius will be chosen to so kernel covers 99.7 % of data.
    if nargin < 2
        Radius = ceil(3*SD);
    end
    side = 2*Radius+1;
    result = zeros(side);
    for i = 1:side
        for j = 1:side
            x = i-(Radius+1);
            y = j-(Radius+1);
            result(i,j)=(x^2+y^2)^0.5;
        end
    end
    result = exp(-(result .^ 2) / (2 * SD * SD));
    result = result / sum(result(:));
end

%% Interpolation - Fiting a parabola into 3 points and extracting more exact Exterma
function exterma = interpolateExterma(X, Y, Z)
% Exterpolation and Exterma extraction
% Each input is an array with 2 values, t and f(t).
    exterma = Y(1)+...
        ((X(2)-Y(2))*(Z(1)-Y(1))^2 - (Z(2)-Y(2))*(Y(1)-X(1))^2)...
        /(2*(X(2)-Y(2))*(Z(1)-Y(1)) + (Z(2)-Y(2))*(Y(1)-X(1)));
end

%% Function to assign bins to orientations
% 8 bin assignment
function bin = categorizeDirection8(Direction)
    if Direction <= 22.5 && Direction > -22.5
        bin = 1;
    elseif Direction <= 67.5 && Direction > 22.5
        bin = 2;
    elseif Direction <= 112.5 && Direction > 67.5
        bin = 3;
    elseif Direction <= 157.5 && Direction > 112.5
        bin = 4;
    elseif Direction <= -157.5 || Direction > 157.5
        bin = 5;
    elseif Direction <= -112.5 && Direction > -157.5
        bin = 6;
    elseif Direction <= -67.5 && Direction > -112.5
        bin = 7;
    elseif Direction <= -22.5 && Direction > -67.5
        bin = 8;
    end
```

Linear Binary Pattern (LBP)

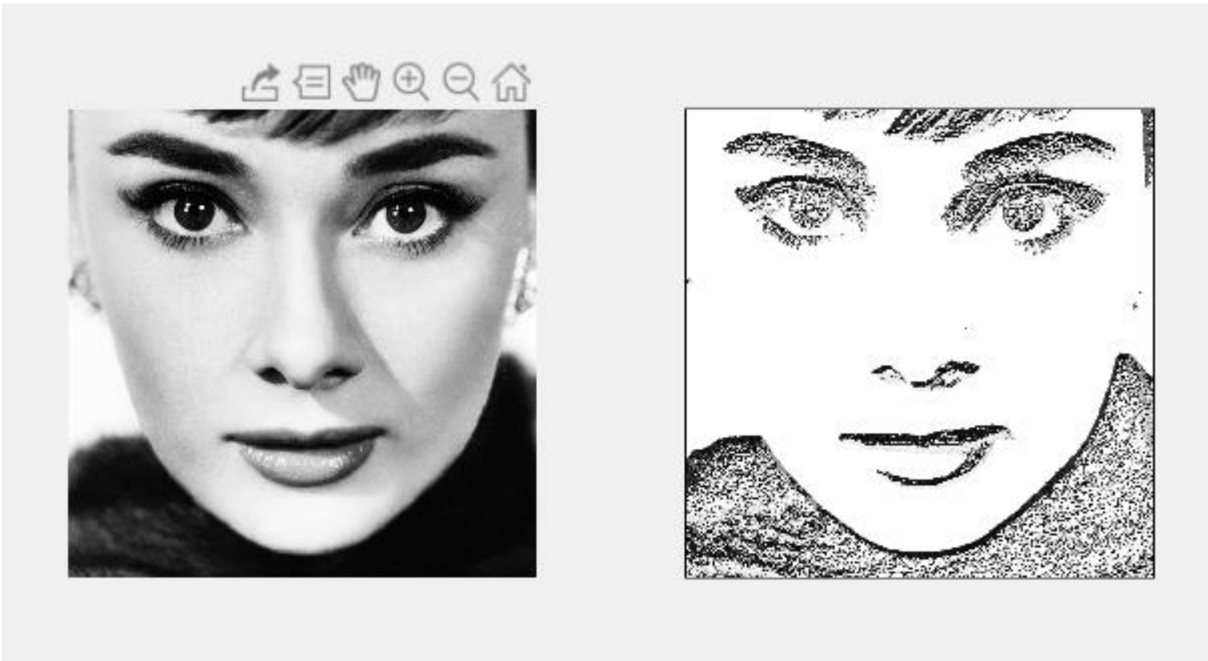
Algorithm:-

The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.



Result:-



Code:-

```
clear all;

% load image
img = rgb2gray(imread('ha.png'));

% run descriptor
filtered_img = lbp(img);

% plot results
subplot(1,2,1);
imshow(img);
subplot(1,2,2);
imshow(filtered_img);

function filtered_img = lbp(img)
%% LBP image descriptor
[nrows, ncols] = size(img);

filtered_img = zeros(nrows, ncols, 'uint8');
for j = 2:ncols-1
    for i = 2:nrows-1
        nhood = nhood8(i, j);
        for k = 1:size(nhood, 1)
```

```
            filtered_img(i, j) = filtered_img(i, j) + ...
                (int8(img(nhood(k, 1), nhood(k, 2)))-int8(img(i, j)) >= 0) * 2^(k-1);
        end
    end
end

end

function idx = nhood8(i, j)
%% Computes the 8-neighborhood of a pixel
idx = [
    i-1, j-1;
    i-1, j;
    i-1, j+1;

    i, j-1;
    i, j+1;

    i+1, j-1;
    i+1, j;
    i+1, j+1
];

End
```

K-mean/Fuzzy C-mean Clustering techniques in Image segmentation

Algorithm:-

Algorithm 1 K-means clustering algorithm

Compute the intensity distribution /*the histogram of intensities*/.
Initialize the centroids with k random intensities /*the number of clusters to be found*/.
Initialize $\{u_i\}i^k = 1$
FOR: Each cluster C_j
REPEAT:
Cluster the points based on distance of their intensities from the centroid intensities.

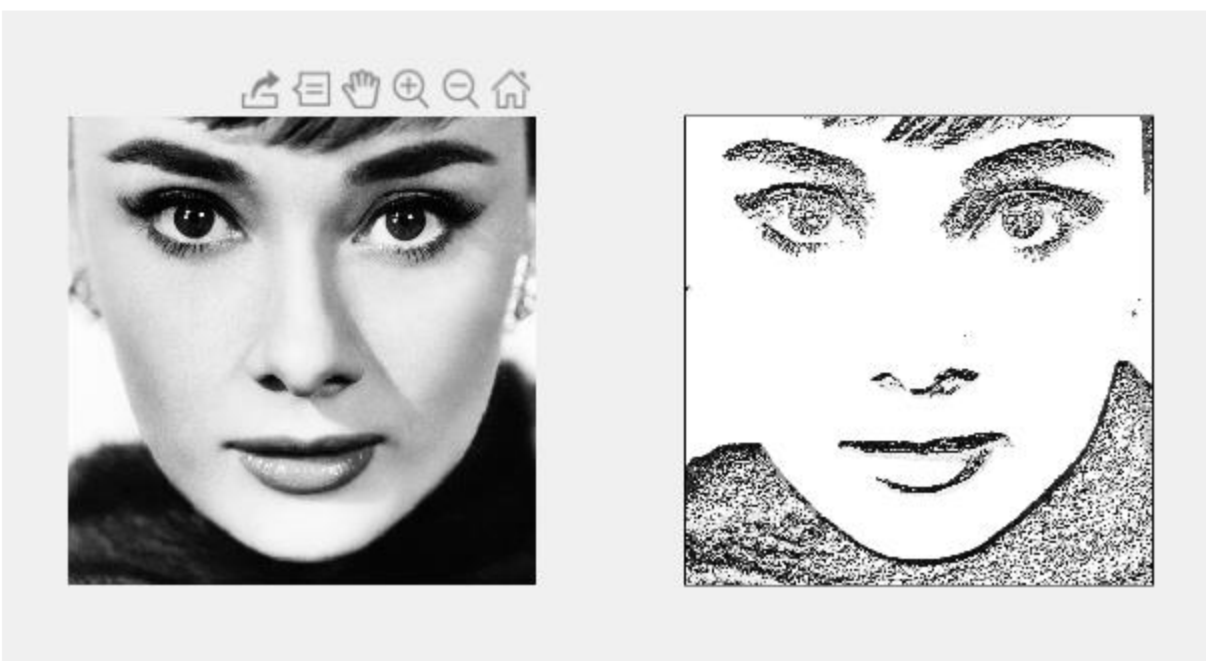
$$c^{(i)} := arg\min_j \|x^{(i)} - \mu_i\|^2 \tag{2}$$

Compute the new centroid for each of the clusters.

$$\mu_i := \frac{\sum_{i=1}^{m} 1\{c_{(i)=j}\}x^{(i)}}{\sum_{i=1}^{m} 1\{c_{(i)=j}\}} \tag{3}$$

where i iterates over the all intensities, j iterates over all the centroids, and μ_i is the centroid intensity.
UNTIL: cluster labels of the image does not change anymore.
ENDFOR

Result:-



Code:-

```
 % function main
clc;
clear all;
close all;

im = imread('ha.png');
subplot(2,1,1),imshow(im);
subplot(2,1,2),imhist(im(:,:,1));
title('INPUT IMAGE
HISTOGRAM');%figure,imhist(im(:,:,2)),title('blue');figure,imhist(im(:,:,3)),title('Green');

figure;
I = imnoise(rgb2gray(im),'salt & pepper',0.02);
subplot(1,2,1),imshow(I);
title('Noise adition and removal using median filter');
K = medfilt2(I);
subplot(1,2,2),imshow(K);


im = double(im);
s_img = size(im);
r = im(:,:,1);
g = im(:,:,2);
b = im(:,:,3);
% [c r] = meshgrid(1:size(i,1), 1:size(i,2));
data_vecs = [r(:) g(:) b(:)];

k= 4;

[ idx C ] = kmeansK( data_vecs, k );
% d = reshape(data_idxs, size(i,1), size(i,2));
% imagesc(d);

palette = round(C);

%Color Mapping
idx = uint8(idx);
outImg = zeros(s_img(1),s_img(2),3);
temp = reshape(idx, [s_img(1) s_img(2)]);
for i = 1 : 1 : s_img(1)
    for j = 1 : 1 : s_img(2)
        outImg(i,j,:) = palette(temp(i,j),:);
    end
end
```

```
cluster1 = zeros(size(r));
cluster2 = zeros(size(r));
cluster3 = zeros(size(r));
cluster4 = zeros(size(r));

figure;
cluster1(find(outImg(:,:,1)==palette(1,1))) = 1;
subplot(2,2,1), imshow(cluster1);
cluster2(find(outImg(:,:,1)==palette(2,1))) = 1;
subplot(2,2,2), imshow(cluster2);
cluster3(find(outImg(:,:,1)==palette(3,1))) = 1;
subplot(2,2,3), imshow(cluster3);
cluster4(find(outImg(:,:,1)==palette(4,1))) = 1;
subplot(2,2,4), imshow(cluster4);

cc = imerode(cluster4,[1 1]);
figure,imshow(imerode(cluster4,[1 1]));
title('result image');

[label_im, label_count] = bwlabel(cc,8);
stats = regionprops(label_im, 'centroid');

for i=1:label_count
    area(i) = stats(i).Area;
end

[maxval, maxid] = max(area);

label_im(label_im ~= maxid) = 0;
label_im(label_im == maxid) = 1;

figure,imshow(label_im);
title('lbp');

% outImg = uint8(outImg);
% imtool(outImg);


code_end = 1;
```