Assignment 2

ZHANG YIFEI

Task1 Filtering

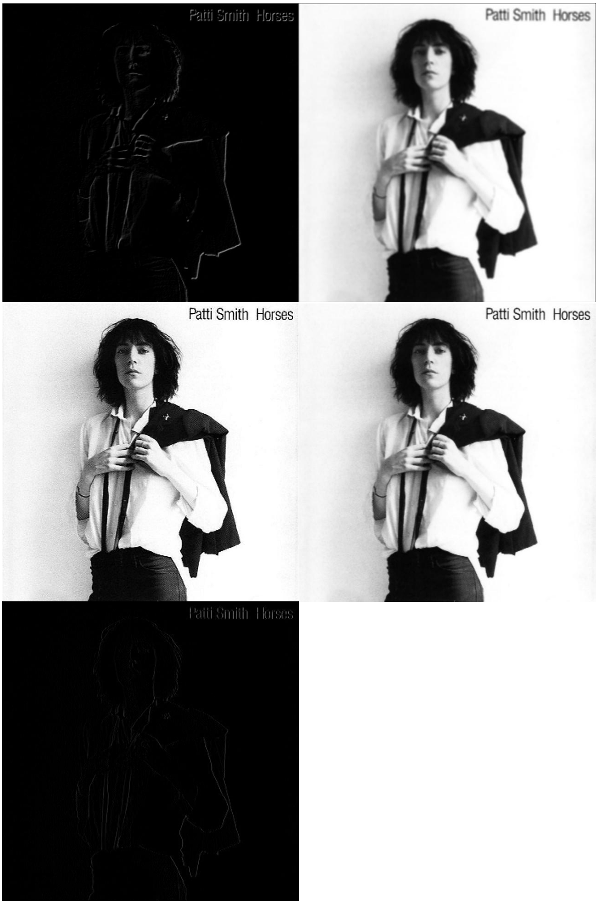
```
clc
clear
close all

% Read Original Image
image = imread('img.jpg');
image = rgb2gray(image);

%% Define the con_kernel
% custom_kernel = 1/3*[
%     1, 1, 0;
%     1, 0, -1;
%     0, -1, -1
% ];

% custom_kernel = 1/25*[1 1 1 1 1;
% 1 1 1 1 1;
% 1 1 1 1 1;
% 1 1 1 1 1;
% 1 1 1 1 1
% ];


% custom_kernel = [
%     0, -1, 0;
%     -1, 5, -1;
%     0, -1, 0
% ];

% custom_kernel = [
%     0, 0, 0;
%     0, 1, 0;
```

```
%     0, 0, 0
% ];
%
custom_kernel = [
    1, -2, 1
];

% Converlution Operation
convolved_image = conv2(double(image), custom_kernel, 'same'); % 'same' to  guarantee the shape

% plot the orignal image and the covlutioned image
subplot(1, 2, 1);
imshow(uint8(image));
title('Original Image');

subplot(1, 2, 2);
imshow(uint8(convolved_image));
title('Convolved Image');

% Save the image
imwrite(uint8(convolved_image), 'convolved_image5.jpg');
```

Result:

- The result of f1 is image B , because this convolution kernel with positive and negative distributions on each side can detect black and white boundaries.
- The result of f2 is image A , because using this convolution kernel is equivalent to averaging the value of that pixel point with the values of the surrounding pixel points, and the image will become blurrier.
- The result of f3 is image C , It is equivalent to magnifying the difference between the center pixel point and the surrounding pixels, and the noise will be more obvious.
- The result of f4 is image E , this convolution operation actually preserves the original pixel values, all indistinguishable from the original.

- The result of f5 is image D, this convolution operation is sensitive to changes in pixel values in the x-direction, which manifests itself in the image as distinct vertical stripes.
- **image**



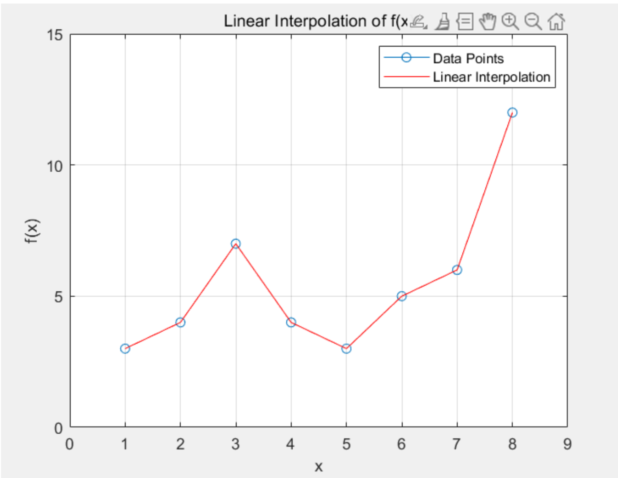
Task2 Interpolation

a)

- Linear interpolation is a method used to estimate a value (or points) that lies between two known values within a continuous range or dataset. It assumes a linear relationship between the known data points and uses this assumption to calculate an intermediate value.
- matlab code:

```
clc
% Given data
x = 1:8; % Data points for x
y = [3, 4, 7, 4, 3, 5, 6, 12]; % Corresponding y values

% Values of x for interpolation
xi = 1:0.1:8; % Values of x for interpolation

l_xi = size(xi,2);
yi = zeros(1,l_xi);

% Linear interpolation
l_x = size(x,2);
    for i = 1:l_xi
        for j = 1:l_x-1
            % Suppose it is necessary to compute the interpolation formula
            if x(j+1) > xi(i)
                yi(i) = y(j)+(y(j+1)-y(j))/(x(j+1)-x(j))*(xi(i)-x(j));
                break;
            end
            % If the data at the interpolation point is already measured
            % The value is given directly to it, saving computational resources.
            if x(j) == xi(i)
                yi(i) = y(j);
                break;
            end
        end
        % The above does not take the last data point into account and needs to be added.
        yi(l_xi) = y(l_x);
```

```
    end

% Plot the original data points and linear interpolation

plot(x, y, 'o-', 'DisplayName', 'Data Points');
xlim([0, 9]);
ylim([0, 15]);
hold on;
plot(xi, yi, 'r-', 'DisplayName', 'Linear Interpolation');
title('Linear Interpolation of f(x)');
xlabel('x');
ylabel('f(x)');
legend;
grid on;
```

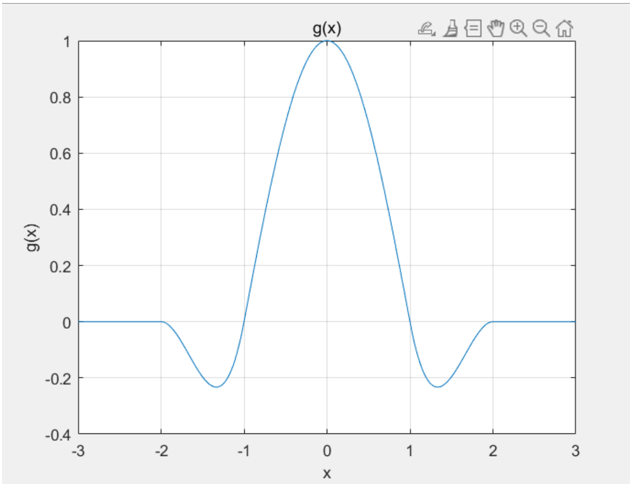


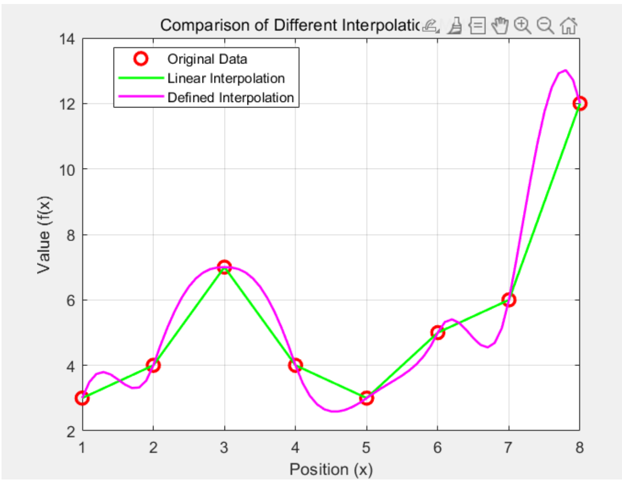
- The line plot appears continuous because it connects data points with straight lines, but in cases where the function has abrupt changes or sharp corners, it is not differentiable at those specific points, even though the plot itself appears continuous.

b)

$$g(x) = \begin{cases} 1 - |x| & if|x| < 1 \\ 0 & otherwise \end{cases}$$

c)






```
clc
clear
close all

```

```
% Define original data points
x = 1:8;
f = [3 4 7 4 3 5 6 12];

% Define interpolation points
xi = 1:0.1:8; % Interpolate between original data points

%% Linear Interpolation Function 1: Linear Interpolation

g1 = @(x) (1 - abs(x)) .* (abs(x) <= 1); % Linear interpolation weights
Fi1 = zeros(size(xi));
fi1 = zeros(size(xi));
for j = 1:length(xi)
    for i = 1 : 8
        fi1(j)= g1(xi(j)-i).*f(i);
        Fi1(j)=Fi1(j) + fi1(j);
    end
end


%% Linear Interpolation Function 2: Defined Interpolation
g2 = @(x) cos(pi/2 * abs(x)) .* (abs(x) <= 1)-(pi/2) *(abs(x)^3 - 5*abs(x)^2 + 8*abs(x)-4).*(abs(x) <= 2 && abs(x) > 1); % Cubic interpolation weights
Fi2 = zeros(size(xi));
fi2 = zeros(size(xi));
for j = 1:length(xi)
    for i = 1 : 8
        fi2(j)= g2(xi(j)-i).*f(i);
        Fi2(j)=Fi2(j) + fi2(j);
    end
end

%% Determine whether F2 is differentiable or not
F2_derivative = diff(Fi2);
F2_derivative = isAlways(iscontinuous(F2_derivative, x, 1, 8));
```

```
if ~any(isnan(F2_derivative)) && (F2_derivative)
    disp('F2(x) is differentiable');
else
    disp('F2(x) is not differentiable');
end

%% Plot original data and results of different interpolation methods
figure;
plot(x, f, 'ro', 'MarkerSize', 8, 'LineWidth', 2, 'DisplayName', 'Original Data');
hold on;
plot(xi, Fi1, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Linear Interpolation');
plot(xi, Fi2, 'm-', 'LineWidth', 1.5, 'DisplayName', 'Defined Interpolation');
xlabel('Position (x)');
ylabel('Value (f(x)');
title('Comparison of Different Interpolation Methods');
legend('Location', 'Best');
grid on;
hold off;

```

- The image is continuous because there exists a mapping of all values in the interval 1-8.
- The function is differentiable because I have computed it using MATLAB, and all points are differentiable, with continuous derivatives. This holds true, especially at the inflection points where the derivatives remain continuous.

Task3 Classification using Nearest Neighbour and Bayes theorem

3.1 Nearest Neighbours

```
clc
clear
close all

% Define class measurements and labels
class1_measurements = [0.4003, 0.3988, 0.3998, 0.3997];
class2_measurements = [0.2554, 0.3139, 0.2627, 0.3802];
class3_measurements = [0.5632, 0.7687, 0.0524, 0.7586];
class_labels = [1, 2, 3]; % Corresponding class labels

% Define test measurements
test_measurements = [
    [0.4010, 0.3995, 0.3991]; % Test data for Class 1
    [0.3287, 0.3160, 0.2924]; % Test data for Class 2
    [0.4243, 0.5005, 0.6769]  % Test data for Class 3
];

% Initialize counter for correct classifications
correct_classifications = 0;

% Loop through each test measurement
for i = 1:size(test_measurements, 1)
    test_measurement = test_measurements(i, :);
    for p = 1:length(test_measurement)
        % Initialize variables for nearest neighbor search
        nearest_class = 0;
        min_distance = Inf;
        % Loop through training measurements in each class
        for j = 1:numel(class_labels)
            class_label = class_labels(j);

            % Get the training measurements for the current class
```

```
            train_measurements = [];
            if class_label == 1
                train_measurements = class1_measurements;
            elseif class_label == 2
                train_measurements = class2_measurements;
            elseif class_label == 3
                train_measurements = class3_measurements;
            end

            for k = 1:length(train_measurements)
                % Calculate distance between the test measurement and each training measurement
                distance = abs(train_measurements(k) - test_measurement(p));

                % Find the minimum distance and corresponding class label
                if distance < min_distance
                    min_distance = distance;
                    nearest_class = class_label;
                end
            end
        end

        % Check if the nearest neighbor classification is correct
        if nearest_class == i
            correct_classifications = correct_classifications + 1;
        end
    end
end

% Display the number of correctly classified test measurements
disp(['Correctly classified measurements: ' num2str(correct_classifications)]);

```

```
Correctly classified measurements: 8
>>
```

3.2 Gaussian distributions

```
clc
clear
close all

%% Define class parameters & test measurements
class_params = [
    struct('mean', 0.4, 'variance', 0.01),   % Class 1
    struct('mean', 0.32, 'variance', 0.05),  % Class 2
    struct('mean', 0.55, 'variance', 0.2)   % Class 3
];

test_measurements = [
     0.4003; 0.3988; 0.3998; 0.3997; 0.4010; 0.3995; 0.3991;
    0.2554; 0.3139; 0.2627; 0.3802;0.3287; 0.3160; 0.2924;
    0.5632; 0.7687; 0.0524; 0.7586;0.4243; 0.5005; 0.6769
];

% Initialize counter for correct classifications
correct_classifications = 0;
class_probabilities = zeros(size(test_measurements, 1), numel(class_params));

%% Loop through each test measurement
for i = 1:size(test_measurements, 1)
    test_measurement = test_measurements(i, :);
    for p = 1:length(test_measurement)
        for j = 1:numel(class_params)
            params = class_params(j);
```

```
            mean = params.mean;
            variance = params.variance;
            % Calculate likelihood using normal distribution
            likelihood = normpdf(test_measurement(p), mean, variance);
            class_probabilities(i,j) = prod(likelihood);
        end
    end
end

%% Predict label
[~, predictions] = max(class_probabilities , [], 2);
% Display the number of correctly classified test measurements
correct_count = sum(predictions == [1; 1; 1; 1; 1; 1;1;2; 2; 2; 2; 2; 2;2;3;3;3;3;3;3;3]);
predictions= reshape(predictions, 7, 3)';
disp('Probabilities : ')
disp(class_probabilities);
disp('Prediction : ')
disp(predictions);
disp(['Correctly classified measurements: ' num2str(correct_count)]);

```

```
Probabilities :
   39.8763    2.1972    1.5074
   39.6080    2.3046    1.4989
   39.8862    2.2326    1.5046
   39.8763    2.2398    1.5040
   39.6953    2.1481    1.5113
   39.8444    2.2541    1.5029
   39.7330    2.2829    1.5006
    0.0000    3.4631    0.6741
    0.0000    7.9197    0.9937
    0.0000    4.1377    0.7109
    5.6183    3.8651    1.3911
    0.0000    7.8590    1.0815
    0.0000    7.9534    1.0061
    0.0000    6.8513    0.8703
    0.0000    0.0001    1.9904
    0.0000    0.0000    1.0971
    0.0000    0.0000    0.0903
    0.0000    0.0000    1.1579
    2.0829    0.9058    1.6372
    0.0000    0.0118    1.9345
    0.0000    0.0000    1.6310

Prediction :
     1     1     1     1     1     1     1
     2     2     2     1     2     2     2
     3     3     3     3     1     3     3

Correctly classified measurements: 19
>>
```

Task4 Image Classification

a) Case 1

Bayes' theorem:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Priori:

$$P(X_1) = \frac{1}{4}; P(X_2) = \frac{1}{2}; P(X_3) = \frac{1}{4}$$

Calculate the conditional probability:

$$P(Y|X_1) = 0.1 \times 0.9^3 = 0.0729$$

$$P(Y|X_2) = 0.1^3 \times 0.9 = 0.0009$$

$$P(Y|X_3) = 0.1^2 \times 0.9^2 = 0.0081$$

Then, calculate the normalizing constant P(Y):

$$P(Y) = P(Y|X_1) \times P(X_1) + P(Y|X_2) \times P(X_2) + P(Y|X_3) \times P(X_3)$$

$$= (0.0729 \times 1/4) + (0.0009 \times 1/2) + (0.0081 \times 1/4) = 0.0207$$

The posterior probabilities are:

$$P(X_1|Y) = \frac{P(Y|X_1)P(X_1)}{P(Y)} = \frac{0.0729 \times 0.25}{0.0207} = 0.8804 \approx 88.04\%$$

$$P(X_2|Y) = \frac{P(Y|X_2)P(X_2)}{P(Y)} = \frac{0.0009 \times 0.5}{0.0207} = 0.0217 \approx 2.17\%$$

$$P(X_3|Y) = \frac{P(Y|X_3)P(X_2)}{P(Y)} = \frac{0.0081 \times 0.25}{0.0207} = 0.0978 \approx 9.78\%$$

So the result of MAP (maximum a posteriori) estimation should be A (X1).

b) case 2

Calculate the conditional probability:

$$P(Y|X_1) = 0.4 \times 0.6^3 = 0.0864$$

$$P(Y|X_2) = 0.4^3 \times 0.6 = 0.0384$$

$$P(Y|X_3) = 0.4^2 \times 0.6^2 = 0.0576$$

Then, calculate the normalizing constant P(Y):

$$P(Y) = P(Y|X_1) \times P(X_1) + P(Y|X_2) \times P(X_2) + P(Y|X_3) \times P(X_3)$$

$$= (0.0864 \times 1/4) + (0.0384 \times 1/2) + (0.0576 \times 1/4) = 0.0552$$

The posterior probabilities are:

$$P(X_1|Y) = \frac{P(Y|X_1)P(X_1)}{P(Y)} = \frac{0.0864 \times 0.25}{0.0552} = 0.3913 \approx 39.13\%$$

$$P(X_2|Y) = \frac{P(Y|X_2)P(X_2)}{P(Y)} = \frac{0.0384 \times 0.5}{0.0552} = 0.3478 \approx 34.78\%$$

$$P(X_3|Y) = \frac{P(Y|X_3)P(X_2)}{P(Y)} = \frac{0.0576 \times 0.25}{0.0552} = 0.2608 \approx 26.08\%$$

The result of MAP (maximum a posteriori) estimation should be A (X1).

Task5 Line Classification

```
clc
clear
close all

% Define matrix O
O = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 1 0 0];

% Define Assume matrices
Assume_1 = [1 0 0 0; 1 0 0 0; 1 0 0 0; 1 0 0 0];
Assume_2 = [0 1 0 0; 0 1 0 0; 0 1 0 0; 0 1 0 0];
Assume_3 = [0 0 1 0; 0 0 1 0; 0 0 1 0; 0 0 1 0];
Assume_4 = [0 0 0 1; 0 0 0 1; 0 0 0 1; 0 0 0 1];

% Concatenate Assume matrices along the third dimension
Assume = cat(3, Assume_1, Assume_2, Assume_3, Assume_4);
```

```

% Initialize the Result matrix with zeros
Result = zeros(4, 4, 4);

% Define prior probabilities for each case
priori = [0.3 0.2 0.2 0.3];

% Initialize variables
py = 0;
pyx = zeros(1, 4);
prob = zeros(1, 4);

% Iterate through each 4x4x4 submatrix
for i = 1:4
    for j = 1:4
        for k = 1:4
            % Compare the current 4x4 submatrix with matrix O
            % If the elements are the same, set the corresponding result to 0.8; otherwise, set it to 0.2
            Result(i, j, k) = (Assume(i, j, k) == O(i, j)) * 0.8 + (Assume(i, j, k) ~= O(i, j)) * 0.2;
        end
    end
end

% Calculate pyx and py
for i = 1:4
    pyx(i) = prod(prod(Result(:, :, i)));
    py = py + pyx(i) * priori(i);
end

% Calculate the probability for each case
for i = 1:4
    prob(i) = (pyx(i) * priori(i)) / py;
end

% Display the Result matrix and probabilities
disp(Result);
disp(prob);
```

```
Compare Result:

(:,:,1) =

    0.8000    0.8000    0.8000    0.8000
    0.2000    0.2000    0.8000    0.8000
    0.2000    0.8000    0.2000    0.8000
    0.2000    0.2000    0.8000    0.8000


(:,:,2) =

    0.2000    0.2000    0.8000    0.8000
    0.8000    0.8000    0.8000    0.8000
    0.8000    0.2000    0.2000    0.8000
    0.8000    0.8000    0.8000    0.8000


(:,:,3) =

    0.2000    0.8000    0.2000    0.8000
    0.8000    0.2000    0.2000    0.8000
    0.8000    0.8000    0.8000    0.8000
    0.8000    0.2000    0.2000    0.8000


(:,:,4) =

    0.2000    0.8000    0.8000    0.2000
    0.8000    0.2000    0.8000    0.2000
    0.8000    0.8000    0.2000    0.2000
    0.8000    0.2000    0.8000    0.2000

prob:
    0.0807    0.8605    0.0538    0.0050
```

Assume that the image is Y. The posterior probabilities are:

$$P(col1|Y) = 0.0805 \approx 8.05\%$$

$$P(col2|Y) = 0.8595 \approx 85.95\%$$

$$P(col3|Y) = 0.0536 \approx 5.36\%$$

$$P(col4|Y) = 0.0050 \approx 0.50\%$$

The result of MAP (maximum a posteriori) estimation should be Column 2.

Task6 Character Classification

```
clc
clear
close all

% Define the input matrix x
x = [0 0 0; 1 0 0; 0 1 0; 0 0 1; 1 1 0];

% Define Assume matrices
Assume_1 = [1 1 0; 1 0 1; 1 1 0; 1 0 1; 1 1 0];
Assume_2 = [0 1 0; 1 0 1; 1 0 1; 1 0 1; 0 1 0];
Assume_3 = [0 1 0; 1 0 1; 0 1 0; 1 0 1; 0 1 0];

% Concatenate Assume matrices along the third dimension
Assume = cat(3, Assume_1, Assume_2, Assume_3);

% Initialize the Result matrix with zeros
Result = zeros(5, 3, 3);

% Define prior probabilities for each case
priori = [0.35 0.4 0.25];

% Initialize variables
py = 0;
pyx = zeros(1, 3);
prob = zeros(1, 3);

% Iterate through each 5x3x3 submatrix
for i = 1:5
    for j = 1:3
        for k = 1:3
            % Compare the current 5x3 submatrix with matrix Assume
            if Assume(i, j, k) == x(i, j) && Assume(i, j, k) == 1
                Result(i, j, k) = 0.8;
            elseif Assume(i, j, k) == x(i, j) && Assume(i, j, k) == 0
                Result(i, j, k) = 0.7;
            elseif Assume(i, j, k) ~= x(i, j) && Assume(i, j, k) == 1
```

```
                Result(i, j, k) = 0.2;
            elseif Assume(i, j, k) ~= x(i, j) && Assume(i, j, k) == 0
                Result(i, j, k) = 0.3;
            end
        end
    end
end

% Calculate pyx and py
for i = 1:3
    pyx(i) = prod(prod(Result(:,:,i)));
    py = py + pyx(i) * priori(i);
end

% Calculate the probability for each case
for i = 1:3
    prob(i) = (pyx(i) * priori(i)) / py;
end

% Display the Compare Result matrix and probabilities
disp("Compare Result:");
disp(Result);
disp("prob:");
disp(prob);
```

```
Compare Result:

(:,:,1) =

    0.2000    0.2000    0.7000
    0.8000    0.7000    0.2000
    0.2000    0.8000    0.7000
    0.2000    0.7000    0.8000
    0.8000    0.8000    0.7000


(:,:,2) =

    0.7000    0.2000    0.7000
    0.8000    0.7000    0.2000
    0.2000    0.3000    0.2000
    0.2000    0.7000    0.8000
    0.3000    0.8000    0.7000


(:,:,3) =

    0.7000    0.2000    0.7000
    0.8000    0.7000    0.2000
    0.7000    0.8000    0.7000
    0.2000    0.7000    0.8000
    0.3000    0.8000    0.7000

prob:
    0.2251    0.0362    0.7387
```

$$P('B'|x) == 0.2251 \approx 22.51\%$$

$$P('0'|x) = 0.0365 \approx 3.62\%$$

$$P('8'|x) = 0.7379 \approx 73.87\%$$

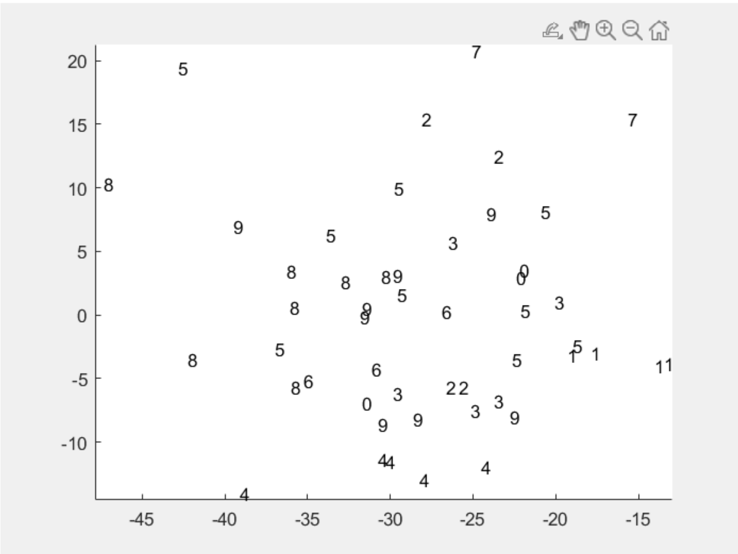
The result of MAP (maximum a posteriori) estimation should be "8".

Task7 The OCR system - part 2 - Feature extraction

```
function features = segment2features(I)
% Compute perimeter
stats = regionprops(I, 'Perimeter', 'Area');
perimeter = stats.Perimeter;
area = stats.Area;
compactness = perimeter^2 / (4*pi*area);

% Calculate area
```

```
stats = regionprops(I, 'Area');
area = stats.Area;

% Calculate convex hull area ratio
stats = regionprops(I, 'Area', 'ConvexHull');
area = stats.Area;
convex_hull_area = polyarea(stats.ConvexHull(:, 1), stats.ConvexHull(:, 2));
convex_hull_ratio = area / convex_hull_area;

% Compute histogram features
histogram_features = sum(I, 2);

% Define parameters for circle detection
radius_range = [6, 15]; % Range of circle radii
sensitivity = 0.9; % Sensitivity, adjust as needed
edge_threshold = 0.1; % Edge threshold, adjust as needed

% Detect circles in the image
[centers, radii] = imfindcircles(I, radius_range, 'Sensitivity', sensitivity, 'EdgeThreshold', edge_threshold);
num_circles = length(centers);

% Compute skeleton length
skeleton = bwmorph(I, 'skel', Inf);
skeleton_length = sum(skeleton(:));

% Extract LBP features
lbp_features = extractLBPFeatures(I);
lbp_features = lbp_features';

% Define HOG parameters
cell_size = [8, 8];
block_size = [2, 2];
num_bins = 9;

% Calculate bounding box area ratio
stats = regionprops(I, 'BoundingBox');
bounding_box = stats.BoundingBox;
```

```
bounding_box_area = bounding_box(3) * bounding_box(4);
bounding_box_ratio = bounding_box(3) / bounding_box(4);

% Extract HOG features
hog_features = extractHOGFeatures(I, 'CellSize', cell_size, 'BlockSize', block_size, 'NumBins', num_bins);
hog_features = hog_features';

% Combine all features into a feature vector
% features = [perimeter; compactness; area; skeleton_length; num_circles; convex_hull_ratio; histogram_features];
features = [num_circles;hog_features;histogram_features;convex_hull_ratio];
```



Several techniques were investigated to extract meaningful features from images. Following rigorous assessment, the the methods that works best are Histogram of Oriented Gradients (HOG), Hough Circle Detection, pixel-level histogram analysis, and the computation of Convex Hull Area Ratios.

HOG is employed to capture intricate texture and shape details by scrutinizing gradient orientations within localized image regions. Hough Circles are utilized to detect circular patterns within the image, with the flexibility to fine-tune parameters for enhanced detection accuracy. The computation of Convex Hull Area Ratios offers valuable insights into object convexity, facilitating comprehensive shape characterization.

Due to the extensive number of parameters associated with HOG, the data pertaining to the remaining features are illustrated in Figure.

0	2.0000	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
5.0000	0	5.0000
8.0000	0	7.0000
7.0000	5.0000	8.0000
6.0000	7.0000	6.0000
5.0000	10.0000	4.0000
4.0000	9.0000	5.0000
5.0000	7.0000	5.0000
5.0000	7.0000	5.0000
4.0000	8.0000	5.0000
4.0000	8.0000	5.0000
4.0000	8.0000	5.0000
5.0000	9.0000	5.0000
5.0000	8.0000	4.0000
4.0000	8.0000	4.0000
5.0000	7.0000	6.0000
5.0000	8.0000	5.0000
7.0000	11.0000	6.0000
8.0000	10.0000	7.0000
4.0000	6.0000	3.0000
0	1.0000	0
0	0	0
0	0	0
0	0	0
0.4878	0.6478	0.5525

0	0	2.0000	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	6.0000	0	0
0	0	8.0000	3.0000	6.0000
0	6.0000	7.0000	9.0000	8.0000
7.0000	9.0000	5.0000	10.0000	5.0000
10.0000	10.0000	4.0000	8.0000	4.0000
6.0000	7.0000	4.0000	8.0000	5.0000
5.0000	5.0000	8.0000	8.0000	8.0000
8.0000	8.0000	7.0000	9.0000	10.0000
5.0000	7.0000	8.0000	4.0000	7.0000
6.0000	7.0000	6.0000	2.0000	3.0000
3.0000	8.0000	4.0000	4.0000	3.0000
6.0000	8.0000	4.0000	6.0000	3.0000
9.0000	4.0000	3.0000	9.0000	3.0000
6.0000	5.0000	4.0000	11.0000	3.0000
5.0000	10.0000	5.0000	9.0000	4.0000
7.0000	12.0000	8.0000	7.0000	3.0000
8.0000	11.0000	12.0000	0	3.0000
9.0000	8.0000	9.0000	0	5.0000
2.0000	3.0000	8.0000	0	3.0000
0	0	0	0	2.0000
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0.5948	0.6827	0.6045	0.6837	0.5146