DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

REPORT ON TB IMAGE CLASSIFICATION
(CS846E03 – DIGITAL IMAGE PROCESSING)

BY

Lokik Gupta (2160366)

B. Tech in CSE

School of Engineering and Technology,

CHRIST (Deemed to be University), Kumbalagodu,

Bengaluru-560 074

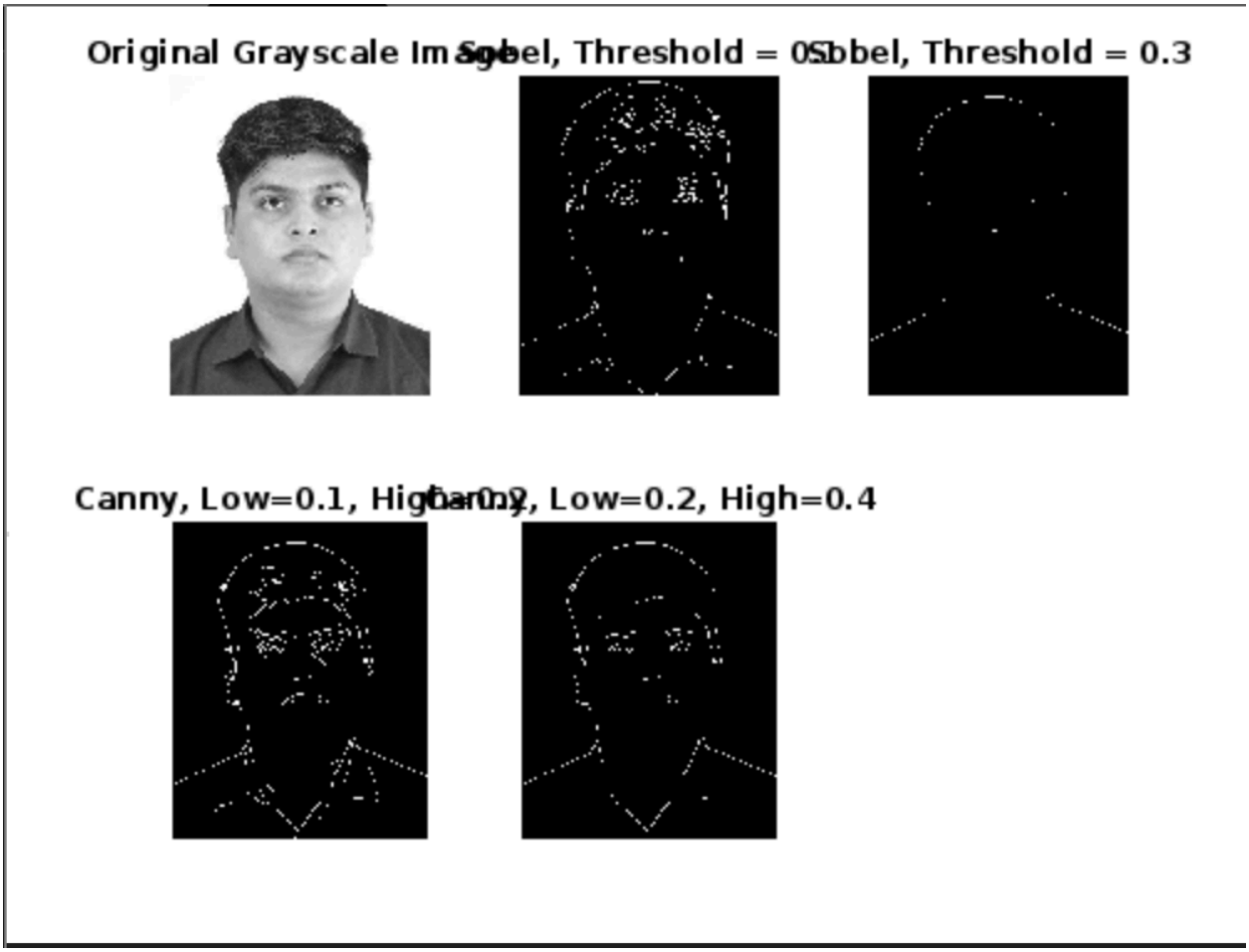
DECEMBER 2024

1. **Use gradient-based methods (e.g., Sobel or Canny) to detect edges in an image. Analyze the effect of changing the threshold parameters.**

CODE

```
% Read and display the input image
img = imread('1165805.jpg'); % Replace with your image file
grayImg = rgb2gray(img); % Convert to grayscale if input is RGB

figure;
subplot(2, 3, 1);
imshow(grayImg);
title('Original Grayscale Image');

% Sobel Edge Detection
sobelEdges1 = edge(grayImg, 'Sobel', 0.1); % Threshold = 0.1
sobelEdges2 = edge(grayImg, 'Sobel', 0.3); % Threshold = 0.3

subplot(2, 3, 2);
imshow(sobelEdges1);
title('Sobel, Threshold = 0.1');

subplot(2, 3, 3);
imshow(sobelEdges2);
title('Sobel, Threshold = 0.3');

% Canny Edge Detection
cannyEdges1 = edge(grayImg, 'Canny', [0.1 0.2]); % Low = 0.1, High = 0.2
cannyEdges2 = edge(grayImg, 'Canny', [0.2 0.4]); % Low = 0.2, High = 0.4

subplot(2, 3, 4);
imshow(cannyEdges1);
title('Canny, Low=0.1, High=0.2');

subplot(2, 3, 5);
imshow(cannyEdges2);
title('Canny, Low=0.2, High=0.4');

% Analyze threshold effect
% Sobel thresholds
thresholds = 0:0.05:0.5;
numEdges = zeros(size(thresholds));

for i = 1:length(thresholds)
    edges = edge(grayImg, 'Sobel', thresholds(i));
    numEdges(i) = sum(edges(:)); % Count the number of edge pixels
end

figure;
plot(thresholds, numEdges, '-o');
xlabel('Threshold');
ylabel('Number of Edge Pixels');
title('Effect of Threshold on Sobel Edge Detection');
grid on;
```

OUTPUT


Original Grayscale Image
Sobel, Threshold = 0.2
Sobel, Threshold = 0.3
Canny, Low=0.1, High=0.2
Canny, Low=0.2, High=0.4

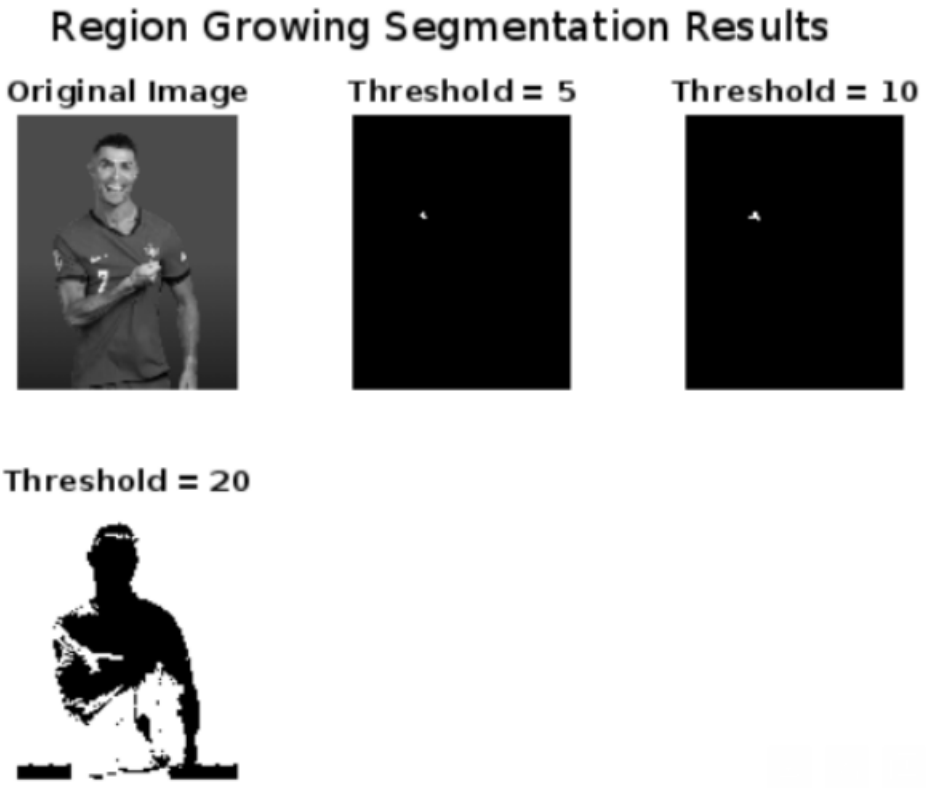
2. Write a program to perform region growing segmentation. Select a seed point manually and analyze the effect of different similarity criteria.

CODE

```
% Load the image
img = imread('blur.jpeg');

% Convert the image to grayscale if it's RGB
if size(img, 3) == 3
    img = rgb2gray(img); % Convert to grayscale
end

% Display the original image and prompt for seed point
figure;
subplot(2, 3, 1);
imshow(img);
title('Original Image');
disp('Click on the image to select a seed point.');

% Select a seed point (x, y)
[x, y] = ginput(1); % User clicks on the image to select the seed point
x = round(x);
y = round(y);
disp(['Seed point selected at (', num2str(x), ', ', num2str(y), ')']);

% Define thresholds for comparison
thresholds = [5, 10, 20]; % Different threshold values
segmented_images = cell(1, length(thresholds)); % To store segmented images

% Loop over thresholds
for idx = 1:length(thresholds)
    threshold = thresholds(idx);

    % Initialize the segmented image
    segmented_img = zeros(size(img));

    % Create a stack for region growing
    stack = [y, x]; % Start with the seed point (row, col)

    % Region growing process
    while ~isempty(stack)
        % Get the current pixel from the stack
        current_pixel = stack(end, :);
        stack(end, :) = []; % Remove the last element
        cy = current_pixel(1);
        cx = current_pixel(2);

        % Check if the current pixel is within bounds
        if cy < 1 || cy > size(img, 1) || cx < 1 || cx > size(img, 2)
            continue;
        end

        % If this pixel has not been visited and satisfies the similarity
criterion
```

```
            if segmented_img(cy, cx) == 0
                if abs(double(img(cy, cx)) - double(img(y, x))) < threshold
                    segmented_img(cy, cx) = 1; % Mark as part of the region

                    % Add neighbors (4-connectivity) to the stack
                    if cy > 1
                        stack = [stack; cy-1, cx]; % Top neighbor
                    end
                    if cy < size(img, 1)
                        stack = [stack; cy+1, cx]; % Bottom neighbor
                    end
                    if cx > 1
                        stack = [stack; cy, cx-1]; % Left neighbor
                    end
                    if cx < size(img, 2)
                        stack = [stack; cy, cx+1]; % Right neighbor
                    end
                end
            end
        end

        % Store the segmented image
        segmented_images{idx} = segmented_img;

        % Display the segmented image in a subplot
        subplot(2, 3, idx + 1); % Place in appropriate subplot
        imshow(segmented_img);
        title(['Threshold = ', num2str(threshold)]);
    end

    % Add an overall title to the figure
    sgtitle('Region Growing Segmentation Results for Different Thresholds');
```

OUTPUT


Region Growing Segmentation Results
Original Image
Threshold = 5
Threshold = 10
Threshold = 20

3. Implement a graph-cut-based segmentation algorithm. Use it to separate the foreground and background of an image.

CODE

```
% Graph-Cut Segmentation Algorithm in MATLAB

% Load the image
image = imread('virat.jpeg');
figure;
imshow(image);
title('Original Image');

% Convert the image to grayscale
if size(image, 3) == 3
    grayImage = rgb2gray(image);
else
    grayImage = image;
end

% Create a binary mask for the foreground and background
threshold = 100; % Adjust threshold according to the image
foregroundMask = grayImage < threshold;
backgroundMask = ~foregroundMask;

% Visualize the foreground and background masks
figure;
subplot(1, 2, 1);
imshow(foregroundMask);
title('Foreground Mask');

subplot(1, 2, 2);
imshow(backgroundMask);
title('Background Mask');

% Prepare the Graph Cut algorithm
% Initialize node and edge weights
[height, width] = size(grayImage);
numPixels = height * width;

% Create cost matrix for terminal nodes
DataCost = zeros(height, width, 2); % [foregroundCost, backgroundCost]
DataCost(:, :, 1) = ~foregroundMask * 10; % Higher cost for assigning
background pixels to foreground
DataCost(:, :, 2) = foregroundMask * 10; % Higher cost for assigning
foreground pixels to background

% Define edge weights for neighboring pixels
SmoothnessCost = 2; % Penalty for assigning different labels to neighbors
lambda = 10; % Regularization parameter
Neighbors = 8; % 4 or 8 connectivity

% Perform Graph Cut segmentation
[Labels, Energy] = graphcutmex(DataCost, SmoothnessCost, lambda, Neighbors);
```

```
% Reshape the result into the original image size
segmentedImage = reshape(Labels, height, width);

% Visualize the segmented result
figure;
imshow(segmentedImage);
title('Segmented Image (Foreground and Background)');
```

OUTPUT

4. **Apply the morphological watershed algorithm to an image. Use the gradient of the image to compute the watershed lines and segment the regions.**

CODE

```
% Morphological Watershed Segmentation in MATLAB

% Load the image
image = imread('img.jpg');
figure;
imshow(image);
title('Original Image');

% Convert the image to grayscale if it's a color image
if size(image, 3) == 3
    grayImage = rgb2gray(image);
else
    grayImage = image;
end

% Compute the gradient of the grayscale image
gradientImage = imgradient(grayImage);

% Invert the gradient image for watershed segmentation
invertedGradient = -gradientImage;

% Create a binary marker image to highlight the foreground
% Use morphological operations to reduce noise and define markers
markerImage = imextendedmin(invertedGradient, 2); % Adjust sensitivity with
the second parameter
markerImage = imimposemin(invertedGradient, markerImage); % Combine gradient
and markers

% Apply the watershed algorithm
L = watershed(markerImage);

% Mark the watershed lines on the original image
watershedLines = L == 0;
segmentedImage = grayImage;
segmentedImage(watershedLines) = 255; % Set watershed lines to white

% Display results
figure;

% Display grayscale image
subplot(2, 2, 1);
imshow(grayImage);
title('Grayscale Image');

% Display gradient image
subplot(2, 2, 2);
imshow(gradientImage, []);
title('Gradient Image');
```

```
% Display marker-based gradient
subplot(2, 2, 3);
imshow(markerImage, []);
title('Markers on Gradient');

% Display final watershed result
subplot(2, 2, 4);
imshow(segmentedImage);
title('Watershed Segmentation');

% Display segmented regions as colored labels
coloredSegments = label2rgb(L, 'jet', 'k', 'shuffle');
figure;
imshow(coloredSegments);
title('Segmented Regions with Watershed Lines');
```

OUTPUT