**DIGITAL IMAGE and VIDEO PROCESSING**

# Project Report

**Title: Region-Based Image Compression Using Image Segmentation**

**Submitted by:**

Viniya N. Bhise
(A-67, 7th Sem, ECE)

# Contents

1. **Introduction** Page No. 3 to 4
2. **Objectives** Page No. 4
3. **Theoretical Background** Page No. 4 to 6
4. **Flow chart** Page No. 7
5. **Code** Page No. 7 to 13
6. **Results** Page No. 14 to 20
7. **Applications** Page No. 21
8. **Limitations and Future scope** Page No. 22
9. **References** Page No. 23

## 1. Introduction:

In the field of digital imaging, the efficient storage and transmission of visual information is very essential. Image compression techniques have emerged as indispensable tools to reduce the size of images, thereby optimizing storage space and network bandwidth. Traditional compression methods, such as JPEG and PNG, treat the entire image uniformly, applying a single compression rate to all pixels. While effective for many scenarios, these methods may not be optimal for images with regions of varying importance. Region-based image compression offers a more nuanced approach by dividing an image into distinct regions and applying different compression levels to each region based on its significance. This technique recognizes that certain areas of an image, such as edges or important objects, may require higher quality than others. By selectively compressing regions, region-based compression can achieve significant reductions in file size without compromising the overall quality of the image.

Image segmentation plays a pivotal role in region-based compression. It involves partitioning an image into meaningful segments or regions that share similar characteristics. By identifying regions of interest (ROI), such as edges, textures, or objects, segmentation enables the compression algorithm to prioritize these areas for higher quality preservation. This targeted approach ensures that critical details are not lost during the compression process, while less important regions can be compressed more aggressively to achieve greater compression ratios.

Image compression techniques aim to reduce the file size of images while maintaining as much visual quality as possible. There are two primary types of compression: lossy and lossless. Lossy compression, like JPEG, reduces file size by discarding some data, which may lead to slight quality degradation. Lossless methods, such as PNG, preserve all image data, ensuring no quality loss but achieving lower compression ratios. These techniques are widely used in applications such as web graphics, medical imaging, and multimedia, where efficient storage and faster transmission are essential.

Image segmentation is the process of partitioning an image into meaningful regions. It is a fundamental task in computer vision and image analysis. Various techniques can be employed for image segmentation, including thresholding, edge detection, region-based methods and clustering. The choice of segmentation algorithm depends on the specific characteristics of the image and the desired level of detail. Region-based compression techniques use image segmentation to identify regions of interest within an image. These regions can then be compressed using different compression parameters based on their importance. For example, regions containing important objects or edges may be compressed with higher quality, while regions with less critical information can be compressed more aggressively. It offers a promising approach for efficiently storing and transmitting images, especially those with regions of varying importance. By combining image segmentation with selective compression techniques, it is possible to achieve significant reductions in file size without compromising image quality. The aim of this project is to explore the application of region-based image compression techniques, with a particular focus on the role of image segmentation in identifying regions of interest. The scope of this project focuses on several key areas. Firstly, it involves the implementation and evaluation of various image segmentation algorithms to effectively partition images into meaningful regions.

This step is crucial for identifying areas of interest that require higher quality during compression. Then, the project aims to develop and implement region-based compression schemes, allowing for selective compression where different levels of compression are applied to different regions of the image based on their importance.

Additionally, the project will include a thorough performance evaluation of the proposed system by assessing metrics such as image quality, and storage space required to save image. To demonstrate the benefits of region-based compression, the project will also compare its performance with traditional compression techniques, which treat all image regions uniformly. By addressing these objectives, the project seeks to contribute to advancements in image compression techniques and offer insights into the potential benefits of region-based approaches for various applications, including medical imaging, remote sensing and multimedia.

## 2. Objectives:

1. Implement Image Segmentation to Identify Regions of Interest (ROI):

   The first objective is to implement image segmentation techniques that can effectively divide an image into different regions based on their importance or visual significance.

2. Apply Region-Based Compression Techniques:

   The second objective focuses on applying compression techniques, such as Wavelet Transform or JPEG, to the segmented regions of the image. The goal is to compress different regions with varying levels of quality, prioritizing the regions of interest (ROI) by applying lower compression rates while allowing higher compression for less important regions.

3. Evaluate and Compare the Performance of Region-Based Compression:

   The final objective is to evaluate the performance of the region-based compression system by assessing key metrics such as compression ratio, image quality, and storage space required to store an image. The project will also compare the results of region-based compression with traditional compression techniques to highlight the advantages in terms of data preservation and efficiency.

## 4. Theoretical Background:

### Image Compression

Image compression refers to the process of reducing the amount of data required to represent an image while preserving as much of its original quality as possible. The goal is to decrease the file size, which leads to reduced storage space and faster transmission over networks. This process is critical for efficient use of bandwidth and storage in applications such as multimedia, medical imaging, remote sensing, and surveillance. There are two main types of image compression: lossy and lossless. These two methods differ in their approach to preserving image quality and reducing file size.

Lossy Compression: Lossy compression is a technique where some image data is discarded during the compression process, resulting in a reduction in file size. This loss of information is typically imperceptible to the human eye, making it suitable for many image applications.

Popular Lossy Compression Algorithms

- JPEG (Joint Photographic Experts Group): Common for photographs, using DCT.
- MPEG (Moving Picture Experts Group): Used in video formats (temporal and spatial)
- HEVC (High Efficiency Video Coding): A newer video compression standard offering high efficiency.

Lossless Compression: Lossless compression is a technique where no image data is lost during the compression process, ensuring that, reconstructed image is identical to the original. This is achieved by using algorithms that can be reversed to recover the original data without any loss.

Common Lossless Compression Algorithms

- PNG (Portable Network Graphics): Used for web graphics, supporting transparency.
- GIF (Graphics Interchange Format): Suitable for simple images and animations.
- ZIP: General-purpose compression for various data types, including images.

### Image segmentation

Image segmentation involves dividing an image into distinct regions or segments that represent meaningful parts. This technique is crucial in many computer vision applications, such as medical imaging, object recognition, and scene understanding. By isolating specific areas, segmentation enables more focused analysis or processing of important regions. It plays a key role in tasks where certain areas hold more significance, like identifying pedestrians or road signs in autonomous vehicles. Various segmentation methods have been developed to address the unique challenges presented by different types of images -

1. Thresholding: This technique classifies pixels based on intensity. Pixels above a certain threshold form one segment, while those below form another. It works well for images with distinct foreground and background.

- Global Thresholding: A single threshold applied across the whole image.
- Adaptive Thresholding: Different thresholds used on smaller regions varying contrast.

Thresholding is computationally efficient but may struggle with complex images or those with overlapping intensity distributions between objects and background.

2. Edge Detection: Edge detection is another widely used segmentation technique that identifies boundaries between different regions based on abrupt changes in pixel intensity. Edges often

correspond to significant transitions in an image, such as object outlines, and edge detection algorithms can highlight these boundaries, allowing segmentation based on these detected edges. Some common edge detection algorithms include:

- Canny Edge Detection: A multi-stage process to detect edges. The steps include Gaussian filtering for noise reduction, calculation of intensity gradients, non-maximum suppression to thin out the edges, and hysteresis thresholding to determine edges.
- Sobel Operator: Detects intensity changes in horizontal and vertical directions.
- Laplacian of Gaussian (LoG): This method detects edges by identifying areas where the second derivative of the image intensity is zero (zero crossings).

Edge detection is useful for images where objects have distinct boundaries, but it may struggle when edges are weak or noisy.

5. Region-Based Methods

Region-based methods segment an image by grouping pixels that are similar according to some criteria, such as color, texture, or intensity. It aims to identify connected regions of similar pixel properties. Few common region-based segmentation techniques include region growing, watershed algorithm, etc. Region-based methods are particularly effective for images with connected or homogeneous regions, making them useful in applications like texture analysis and medical imaging.

**6.** Clustering-Based Methods

Clustering-based segmentation groups pixels into clusters based on their similarity in a multi-dimensional feature space, which might include color, texture, or spatial proximity. Unlike region-based methods, which typically operate on a pixel-by-pixel basis, clustering-based techniques attempt to group similar pixels globally across the image. etc. Clustering-based methods are useful when the image contains groups of pixels that are naturally clustered, such as in satellite or hyperspectral images.

## Region of Interest in Compression

Region-based image compression focuses on areas of interest (ROI), compressing them at higher quality while applying stronger compression to less important regions. This is crucial in applications like medical imaging, where regions such as tumors need to be preserved at high quality, or in surveillance where specific objects require emphasis.

ROI Identification

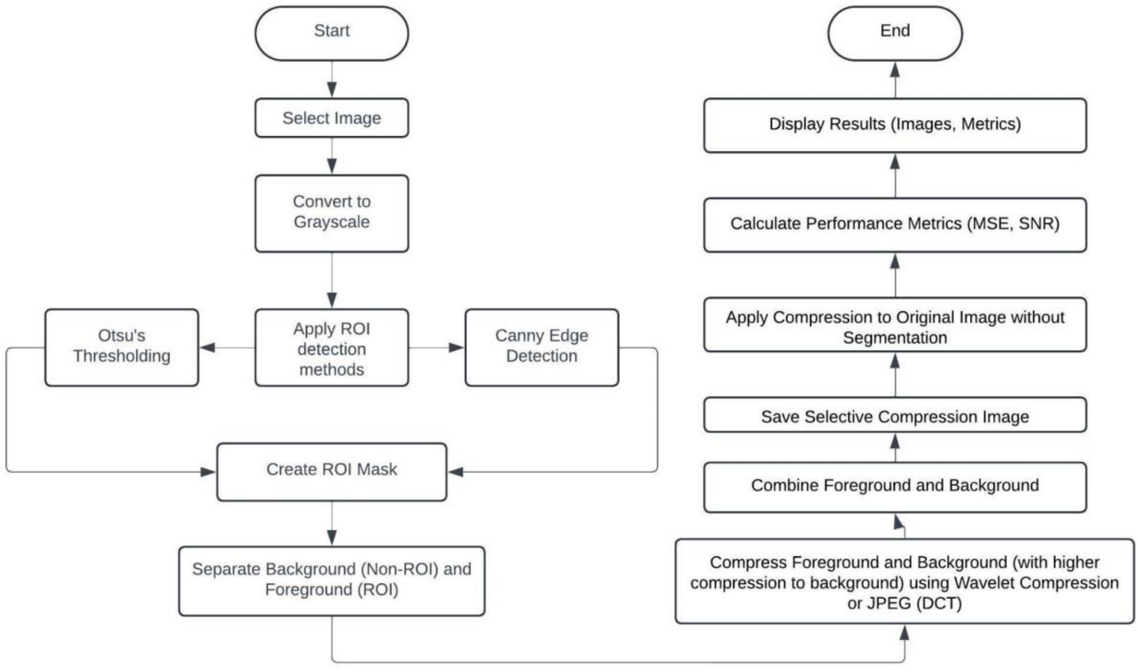
ROIs can be identified manually or automatically:

Manual Selection: Users manually select regions based on their knowledge, such as a radiologist highlighting a tumor. This method is accurate but time-consuming.

Automatic Detection: Algorithms analyze images to detect regions of interest based on characteristics like edges, texture, or motion.

Region-based compression algorithms adjust compression levels based on region importance, preserving key areas while compressing less significant ones. IT improves efficiency by focusing on important areas, balancing quality and file size. It preserves the visual quality of key regions while compressing less significant parts. The method is scalable and adaptable to various applications. Overall, it ensures efficient compression without losing critical data.

## 4. Flow chart:



Start
Select Image
Convert to Grayscale
Apply ROI detection methods
Otsu's Thresholding
Canny Edge Detection
Create ROI Mask
Separate Background (Non-ROI) and Foreground (ROI)
Compress Foreground and Background (with higher compression to background) using Wavelet Compression or JPEG (DCT)
Combine Foreground and Background
Save Selective Compression Image
Apply Compression to Original Image without Segmentation
Calculate Performance Metrics (MSE, SNR)
Display Results (Images, Metrics)
End

## 5. Code:

### I. <u>Segmentation Using Otsu's Thresholding and JPEG Compression (DCT)</u>

```
clc
clear all
close all

foregroundQuality = 90;
backgroundQuality = 35;
overallQuality = 100;
completeCompressionQuality = 35;

disp('Available image choices:');
disp('1. Desert Image (bmp)');
disp('2. Complex Image (bmp)');

image_choice = input("Enter the image choice (digits 1-2): ");
switch image_choice
    case 1
        img = 'desert.bmp';
    case 2
        img = 'complex_img.bmp;
    otherwise
        error('Error. Select from the available choices.');
end

image = imread(img);
grayImage = rgb2gray(image);

figure;
subplot(1,2,1), imshow(img), title("Original Image");
subplot(1,2,2), imshow(grayImage), title("Grayscale Image");
```

```
      % thresholding using Otsu's method
thresholdLevel = graythresh(grayImage);
binaryImage = imbinarize(grayImage, thresholdLevel); % Convert to binary

otsuThreshold = thresholdLevel * 255;
fprintf('\nOtsu''s Threshold Value: %.2f\n', otsuThreshold);

figure; % Plotting the histogram
subplot(1,2,1), imhist(grayImage); hold on;
xline(otsuThreshold, 'r', 'LineWidth', 2); hold off;
title('Histogram of Grayscale Image');

      % Clean the image
cleanedImage = imopen(binaryImage, strel('disk', 5));
cleanedImage = imfill(cleanedImage, 'holes'); subplot(1,2,2)
imshow(~cleanedImage), title("Cleaned mask after thresholding");

      % Label connected components and extract the largest one (foreground)
labeledImage = bwlabel(cleanedImage);
measurements = regionprops(labeledImage, 'Area');
allAreas = [measurements.Area];
[~, largestBlobIndex] = max(allAreas);
objectSegment = ismember(labeledImage, largestBlobIndex);
objectMask = uint8(objectSegment); % Convert logical mask to uint8

      % Separate foreground object (ROI) from the background
background = bsxfun(@times, image, cat(3, objectMask, objectMask, objectMask));
foreground = image - background;

figure;
subplot(1, 2, 1), imshow(foreground), title('Foreground (High Quality)');
subplot(1, 2, 2), imshow(background), title('Background (Low Quality)');

outputDir = 'Thresholding JPG Images';
if ~exist(outputDir, 'dir')
    mkdir(outputDir); % Create the directory
end

      % Compress the foreground with higher quality (lower compression)
imwrite(foreground, fullfile(outputDir, 'foreground_high_quality.jpg'), 'Quality',
foregroundQuality); % Higher quality

      % Compress the background with lower quality (higher compression)
imwrite(background, fullfile(outputDir, 'background_low_quality.jpg'), 'Quality',
backgroundQuality); % Lower quality

highQualityForeground = imread(fullfile(outputDir,
'foreground_high_quality.jpg'));
lowQualityBackground = imread(fullfile(outputDir, 'background_low_quality.jpg'));

      % Combine the high-quality foreground and low-quality background
combinedImage = highQualityForeground + lowQualityBackground;

      % Save the image
imwrite(combinedImage, fullfile(outputDir, 'combined_selective_compression.jpg'),
'quality', overallQuality);

      % Complete compression
imwrite(image, 'complete_compressed_img.jpg', 'jpg', 'Quality',
completeCompressionQuality);

orig = imread(img);
complete_compressed = imread('complete_compressed_img.jpg');
```

```
selective_compressed = imread(fullfile(outputDir,
'combined_selective_compression.jpg'));
figure;
subplot(1, 3, 1), imshow(image), title('Original Image');
subplot(1, 3, 2), imshow(combinedImage), title('Selectively Compressed');
subplot(1, 3, 3), imshow(complete_compressed), title('Complete Compressed');

      % Calculating performance metrics
mse1 = immse(orig, complete_compressed);
mse2 = immse(orig, selective_compressed);
fprintf('\nMSE Selective Compression: %f\n', mse2);
fprintf('MSE Complete Compression: %f\n', mse1);

      % SNR calculation for the foreground and foreground
snrForeground = calculate_snr(foreground, highQualityForeground);
fprintf('\nSNR Foreground: %f\n', snrForeground);
snrBackground = calculate_snr(background, lowQualityBackground);
fprintf('SNR Background: %f\n', snrBackground);

snr1 = calculate_snr(orig, complete_compressed);
snr2 = calculate_snr(orig, selective_compressed);
fprintf('SNR Selective Compression: %f\n', snr2);
fprintf('SNR Complete Compression: %f\n', snr1);

      % Compare file sizes
fileInfoOriginal = dir(img);
fileInfoForeground = dir(fullfile(outputDir, 'foreground_high_quality.jpg'));
fileInfoBackground = dir(fullfile(outputDir, 'background_low_quality.jpg'));
fileInfoCombined = dir(fullfile(outputDir, 'combined_selective_compression.jpg'));
fileInfoComplete = dir('complete_compressed_img.jpg');

fprintf('\nOriginal Image Size: %.2f MB\n', fileInfoOriginal.bytes / (1024 *
1024));
fprintf('Foreground (High Quality) Image Size: %.2f MB\n',
fileInfoForeground.bytes / (1024 * 1024));
fprintf('Background (Low Quality) Image Size: %.2f MB\n', fileInfoBackground.bytes
/ (1024 * 1024));
fprintf('Selectively Compressed Image Size (JPG): %.2f MB\n',
fileInfoCombined.bytes / (1024 * 1024));
fprintf('Complete Compressed Image Size (JPG): %.2f MB\n', fileInfoComplete.bytes
/ (1024 * 1024));

uncompressedSize = fileInfoOriginal.bytes;
selectiveCompressedSize = fileInfoCombined.bytes;
completeCompressedSize = fileInfoComplete.bytes;
selectiveCompressionRatio = uncompressedSize / selectiveCompressedSize;
completeCompressionRatio = uncompressedSize / completeCompressedSize;
fprintf('\nCompression Ratio (selective): %.2f\n', selectiveCompressionRatio);
fprintf('Compression Ratio (complete): %.2f\n', completeCompressionRatio);

function snrValue = calculate_snr(originalImage, noisyImage)
    % Convert images to double precision to avoid overflow
    originalImage = double(originalImage);
    noisyImage = double(noisyImage);

    % Compute the signal power (sum of squares of the original image)
    signalPower = sum(originalImage(:).^2);

    % Compute the noise (difference between the original and noisy/compressed
image)
    noise = originalImage - noisyImage;

    % Compute the noise power (sum of squares of the noise)
    noisePower = sum(noise(:).^2);

    % Calculate the SNR (in dB)
    snrValue = 10 * log10(signalPower / noisePower);
end
```

## II. RGB Separated Segmentation using Otsu's thresholding and JPEG Compression

```
clc
clear all
close all

foregroundQuality = 90;
backgroundQuality = 35;
overallQuality = 100;
completeCompressionQuality = 35;

disp('Available image choices:');
disp('1. Desert Image (bmp)');
disp('2. Complex Image (bmp)');
image_choice = input("Enter the image choice (digits 1-2): ");
switch image_choice
    case 1
        image = 'desert.bmp';
    case 2
        image = 'complex_img.bmp;
    otherwise
        error('Error. Select from the available choices.');
end
img = imread(image);

redChannel = img(:, :, 1); % Separate the RGB channels
greenChannel = img(:, :, 2);
blueChannel = img(:, :, 3);

      % Otsu's thresholding on each RGB channel independently
redThreshold = graythresh(redChannel);
greenThreshold = graythresh(greenChannel);
blueThreshold = graythresh(blueChannel);

figure; % Plot the histogram
subplot(2,2,1), imhist(redChannel), hold on;
xline(redThreshold*255, 'r', 'LineWidth', 2), hold off;
title('Histogram of Red');
subplot(2,2,2), imhist(greenChannel), hold on;
xline(greenThreshold*255, 'g', 'LineWidth', 2), hold off;
title('Histogram of Green');
subplot(2,2,3), imhist(blueChannel), hold on;
xline(blueThreshold*255, 'b', 'LineWidth', 2), hold off;
title('Histogram of Blue');
subplot(2,2,4), imhist(rgb2gray(img)), hold on;
xline(blueThreshold*255, 'b', 'LineWidth', 2);
xline(greenThreshold*255, 'g', 'LineWidth', 2);
xline(redThreshold*255, 'r', 'LineWidth', 2); hold off;
title('Histogram Grayscale with rgb thresholds');

      % Create binary masks for each channel based on the thresholds
redMask = imbinarize(redChannel, redThreshold);   % Binary mask for red
greenMask = imbinarize(greenChannel, greenThreshold);  % For green
blueMask = imbinarize(blueChannel, blueThreshold);   % Binary mask for blue
```

```
      % Clean the masks using morphological operations
redMask = imopen(redMask, strel('disk', 5));  % Open to remove noise
redMask = imfill(redMask, 'holes');          % Fill holes
greenMask = imopen(greenMask, strel('disk', 5));  % Open to remove noise
greenMask = imfill(greenMask, 'holes');          % Fill holes
blueMask = imopen(blueMask, strel('disk', 5));  % Open to remove noise
blueMask = imfill(blueMask, 'holes');          % Fill holes

      % Complement masks for the foreground
redCompMask = imcomplement(redMask);
greenCompMask = imcomplement(greenMask);
blueCompMask = imcomplement(blueMask);

      % Combine the masks (for representation only)
combinedMask = redCompMask | greenCompMask | blueCompMask;

      % Display the individual RGB masks and the concatenated mask
figure;
subplot(2, 2, 1), imshow(redCompMask), title('Red Mask');
subplot(2, 2, 2), imshow(greenCompMask), title('Green Mask');
subplot(2, 2, 3), imshow(blueCompMask), title('Blue Mask');
subplot(2, 2, 4), imshow(combinedMask), title('Overlapping Mask (for
Representation)');

      % Apply the respective masks to each color channel
foreground_red = uint8(redCompMask) .* redChannel;
foreground_green = uint8(greenCompMask) .* greenChannel;
foreground_blue = uint8(blueCompMask) .* blueChannel;

      % Combine the masked RGB channels to get the foreground
foreground = cat(3, foreground_red, foreground_green, foreground_blue);

      % Apply complement mask for the background
background_red = uint8(redMask) .* redChannel; % Background red channel
background_green = uint8(greenMask) .* greenChannel; % Background green channel
background_blue = uint8(blueMask) .* blueChannel; % Background blue channel

      % Combine the masked RGB channels to get the background
background = cat(3, background_red, background_green, background_blue);

      % Save the foreground with high-quality compression
imwrite(foreground, 'foreground_high_quality.jpg', 'jpg', 'Quality',
foregroundQuality);

      % Save the background with low-quality compression
imwrite(background, 'background_low_quality.jpg', 'jpg', 'Quality',
backgroundQuality);

      % Read the compressed images back
highQualityForeground = imread('foreground_high_quality.jpg');
lowQualityBackground = imread('background_low_quality.jpg');

      % Combine the high-quality foreground and low-quality background
final_combined_image = highQualityForeground + lowQualityBackground;

      % Save the final combined image
imwrite(final_combined_image, 'Final_combined_Image.jpg', 'jpg', 'Quality',
overallQuality);

      % Complete compression
imwrite(img, 'complete_compressed_img.jpg', 'jpg', 'Quality',
completeCompressionQuality);
```

## III. Segmentation using Canny Edge Detection and Selective JPEG Compression

```
clc
clear all
close all

foregroundQuality = 90;
backgroundQuality = 35;
overallQuality = 100;
completeCompressionQuality = 35;

disp('Available image choices:');
disp('1. Desert Image (bmp)');
disp('2. Complex Image (bmp)');
image_choice = input("Enter the image choice (digits 1-2): ");
switch image_choice
   case 1
       image = 'desert.bmp';
   case 2
       image = 'complex_img.bmp';
   otherwise
       error('Error. Select from the available choices.');
end
outputDir = 'Canny JPG Images';

img = imread(image);
img_gray = rgb2gray(img);

figure;
subplot(1,2,1), imshow(img), title("Original Image");
subplot(1,2,2), imshow(img_gray), title("Grayscale Image");
figure;
subplot(1,2,1), imhist(img_gray), title('Histogram of Gray Image');

% Edge detection using Canny to find the ROI
edges = edge(img_gray, "canny");
dilatedEdges = imdilate(edges, strel('disk', 5));   % Dilate the edges
roi_mask = imfill(dilatedEdges, 'holes');   % Fill the holes
subplot(1,2,2), imshow(roi_mask), title('ROI mask using Canny Edge Detection');

figure;
subplot(2,2,1), imshow(img), title("Original Image");
subplot(2,2,2), imshow(edges), title("Canny Edges");
subplot(2,2,3), imshow(dilatedEdges), title("Dilated Edges");
subplot(2,2,4), imshow(roi_mask), title("ROI Mask (Filled)");

% Separate the foreground (ROI) and background
mask = roi_mask; background_mask = ~mask;
img = im2uint8(img);

% Extract foreground using logical indexing
foreground = img;
foreground(~cat(3, mask, mask, mask)) = 0; % Set non-ROI pixels to 0 (black)

% Extract background
background = img;
background(~cat(3, background_mask, background_mask, background_mask)) = 0; % Set
non-ROI pixels to 0 (black)

% Compressing the foreground with higher quality (lower compression)
imwrite(foreground, fullfile(outputDir, 'foreground_high_quality.jpg'), 'Quality',
foregroundQuality); % Higher quality
```

```
% Compressing the background with lower quality (higher compression)
imwrite(background, fullfile(outputDir, 'background_low_quality.jpg'), 'Quality',
backgroundQuality); % Lower quality
highQualityForeground = imread(fullfile(outputDir,
'foreground_high_quality.jpg'));
lowQualityBackground = imread(fullfile(outputDir, 'background_low_quality.jpg'));

% Combine the foreground and background for the final result
combinedImage = foreground + background;
imwrite(combinedImage, fullfile(outputDir, 'combined_selective_compression.jpg'),
'Quality', overallQuality);

% Complete compression
imwrite(img, fullfile(outputDir, 'complete_compressed_img.jpg'), 'jpg', 'Quality',
completeCompressionQuality);
```

## IV. Canny Edge Detection for Segmentation and Wavelet Transform for Lossless Compression (JP2)

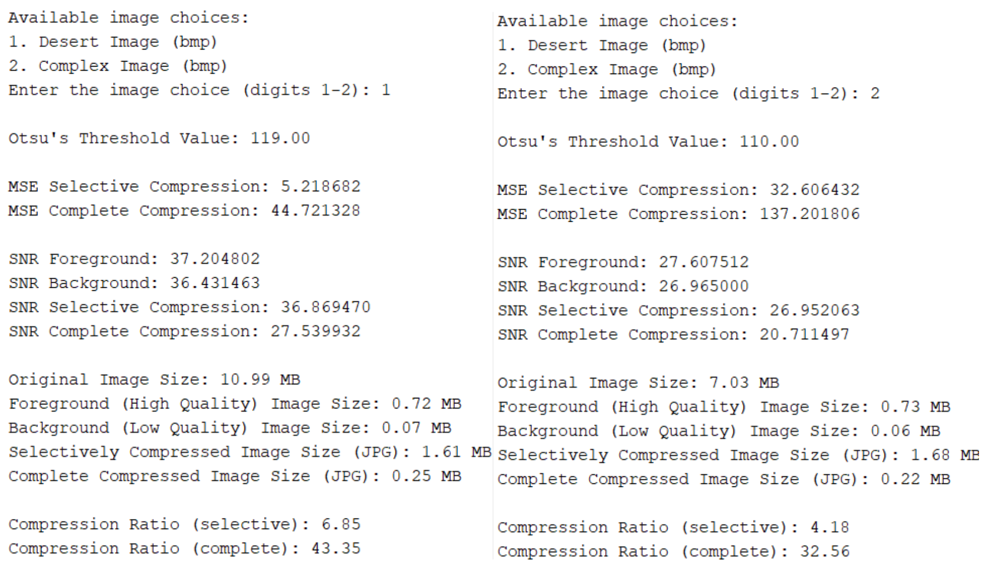
```
clc
clear all
close all

% foregroundRatio is lossless, when unspecified
backgroundRatio = 90;
completeCompressionRatio = 90;

disp('Available image choices:');
disp('1. Desert Image (bmp)');
disp('2. Horse Image (jpg)');

image_choice = input("Enter the image choice (digits 1-2): ");
switch image_choice
    case 1
        image = 'desert.bmp';
    case 2
        image = 'complex_img.bmp';
    otherwise
        error('Error. Select from the available choices.');
end

outputDir = 'Canny Wavelet Images';
if ~exist(outputDir, 'dir')
    mkdir(outputDir);
end

img = imread(image);
img_gray = rgb2gray(img);

figure;
subplot(1,2,1), imshow(img), title("Original Image");
subplot(1,2,2), imshow(img_gray), title("Grayscale Image");

% histogram
figure;
subplot(1,2,1), imhist(img_gray), title('Histogram of Grayscale Image');

% Edge detection using Canny to find the ROI (Region of Interest)
edges = edge(img_gray, "canny");
dilatedEdges = imdilate(edges, strel('disk', 5)); % Dilate the edges
roi_mask = imfill(dilatedEdges, 'holes'); % Fill the holes
```

```
subplot(1,2,2), imshow(roi_mask), title('ROI mask using Canny Edge Detection');

figure;
subplot(2,2,1), imshow(img), title("Original Image");
subplot(2,2,2), imshow(edges), title("Canny Edges");
subplot(2,2,3), imshow(dilatedEdges), title("Dilated Edges");
subplot(2,2,4), imshow(roi_mask), title("ROI Mask (Filled)");


% Separate the foreground (ROI) and background
mask = roi_mask;
background_mask = ~mask;

% Ensure img is in uint8 format (common for images)
img = im2uint8(img);

% Extract foreground using logical indexing
foreground = img;
foreground(~cat(3, mask, mask, mask)) = 0; % Set non-ROI pixels to 0 (black)

% Extract background
background = img;
background(~cat(3, background_mask, background_mask, background_mask)) = 0; % Set
non-ROI pixels to 0 (black)

figure;
subplot(1, 2, 1), imshow(foreground), title('Foreground (High Quality)');
subplot(1, 2, 2), imshow(background), title('Background (Low Quality)');

% Combine the foreground and background for the final result
final_combined = foreground + background;

% Save images in JP2 formats with lossy compression
imwrite(foreground, fullfile(outputDir, 'foreground_lossless.jp2'));
imwrite(background, fullfile(outputDir, 'background_lossy.jp2'),
'CompressionRatio', backgroundRatio); % 90 times smaller

% Save the final result in JP2 formats
imwrite(final_combined, fullfile(outputDir, 'Final_Combined_Image.jp2'));

% Complete compression
imwrite(img, 'complete_compressed_img.jp2', 'CompressionRatio',
completeCompressionRatio);
```

## 6. Results:

### I. Segmentation Using Otsu's Thresholding and JPEG Compression

The goal of the experiment was to perform selective JPEG compression on image segments based on foreground-background separation using Otsu's thresholding. This method aims to maximize the compression efficiency while maintaining a high-quality appearance of critical regions.



```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 1

Otsu's Threshold Value: 119.00

MSE Selective Compression: 5.218682
MSE Complete Compression: 44.721328

SNR Foreground: 37.204802
SNR Background: 36.431463
SNR Selective Compression: 36.869470
SNR Complete Compression: 27.539932

Original Image Size: 10.99 MB
Foreground (High Quality) Image Size: 0.72 MB
Background (Low Quality) Image Size: 0.07 MB
Selectively Compressed Image Size (JPG): 1.61 MB
Complete Compressed Image Size (JPG): 0.25 MB

Compression Ratio (selective): 6.85
Compression Ratio (complete): 43.35
```

```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 2

Otsu's Threshold Value: 110.00

MSE Selective Compression: 32.606432
MSE Complete Compression: 137.201806

SNR Foreground: 27.607512
SNR Background: 26.965000
SNR Selective Compression: 26.952063
SNR Complete Compression: 20.711497

Original Image Size: 7.03 MB
Foreground (High Quality) Image Size: 0.73 MB
Background (Low Quality) Image Size: 0.06 MB
Selectively Compressed Image Size (JPG): 1.68 MB
Complete Compressed Image Size (JPG): 0.22 MB

Compression Ratio (selective): 4.18
Compression Ratio (complete): 32.56
```

Fig.1- MATLAB output for Thresholding and JPG image compression for image 1 & 2

From the metrics, we can observe that the selective compression approach maintains a much higher **SNR** and lower **MSE** for the desert image compared to complete compression. The desert image has a more well-defined bimodal histogram, leading to an effective separation between the background and foreground.

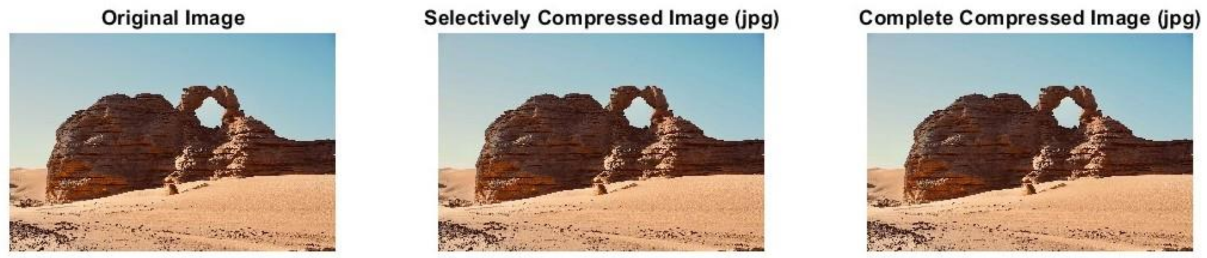

Original Image
Selectively Compressed Image (jpg)
Complete Compressed Image (jpg)

Fig.2- Original image 1 and Compressed image output

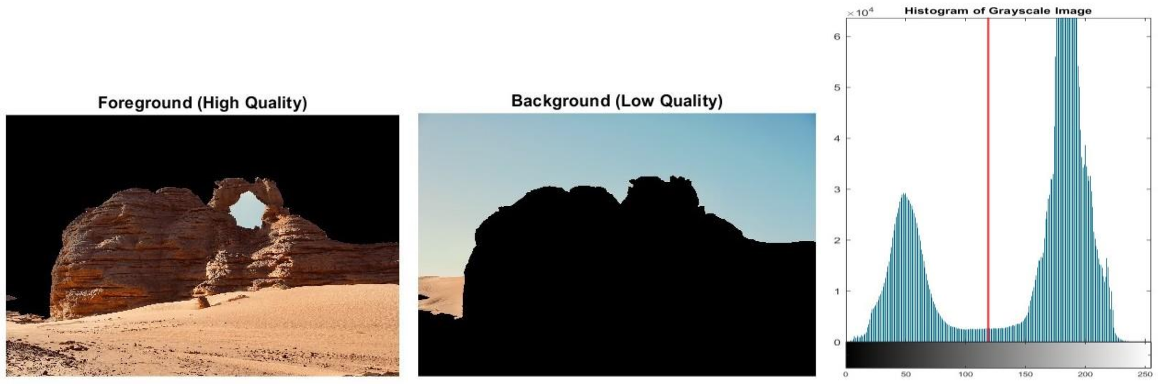


Fig.3 – Foreground, Background, and Histogram of the Grayscale image 1

For the complex image, Otsu’s thresholding does not perform well because the image does not have a bimodal histogram. This leads to less effective segmentation and subsequently poorer selective compression results compared to the desert image. Both **SNR** and **MSE** values are less favorable, with little improvement using selective compression over complete compression. One of the main advantages of selective compression is the trade-off between quality and file size. For the **Desert Image**, selective compression results in a file size of **1.61 MB** compared to the original size of **10.99 MB**. Complete compression results in a smaller file size of **0.25 MB**, but with a significant loss in quality.

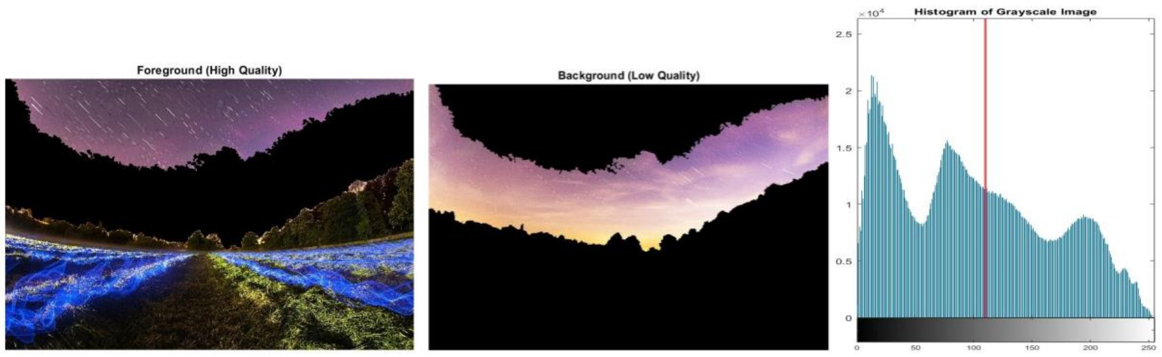


Fig.4- Foreground, Background, and Histogram of the Grayscale image 2

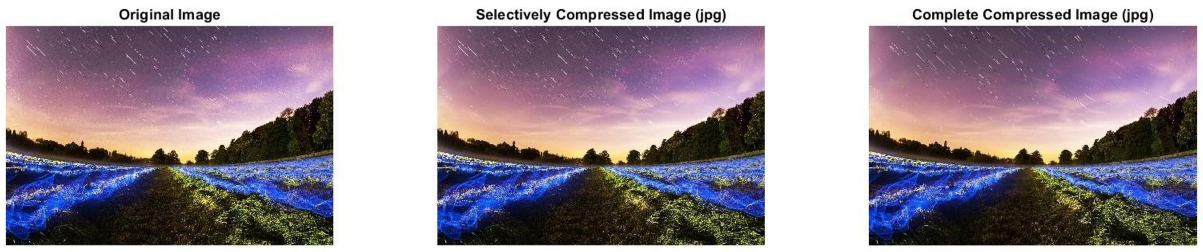
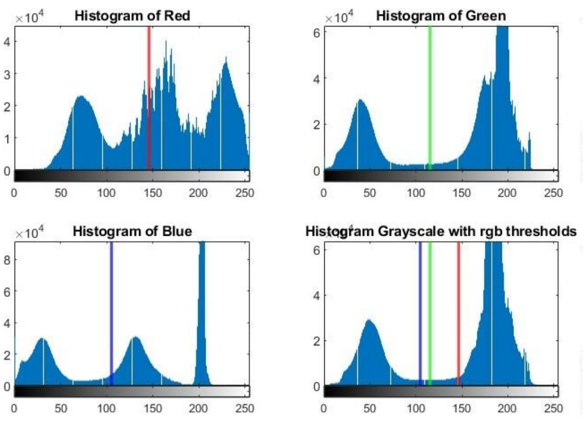

Original Image
Selectively Compressed Image (jpg)
Complete Compressed Image (jpg)

Fig.5- Original image 2 and Compressed image output

## II. RGB Separated Segmentation with Otsu’s thresholding and JPEG Compression

This experiment aimed to improve selective compression by applying Otsu's thresholding independently to the RGB channels, refining segmentation and enhancing compression. The results from the "Desert" and "Complex" images show varied performance based on the image's histogram characteristics.



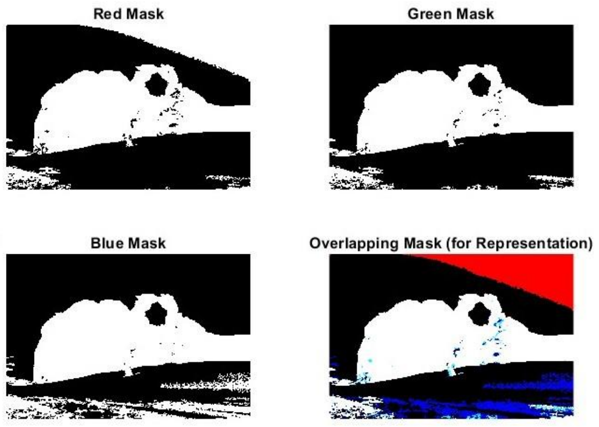


Fig.6- Histograms of RGB separate channels Fig.7- Individual masks for each channel

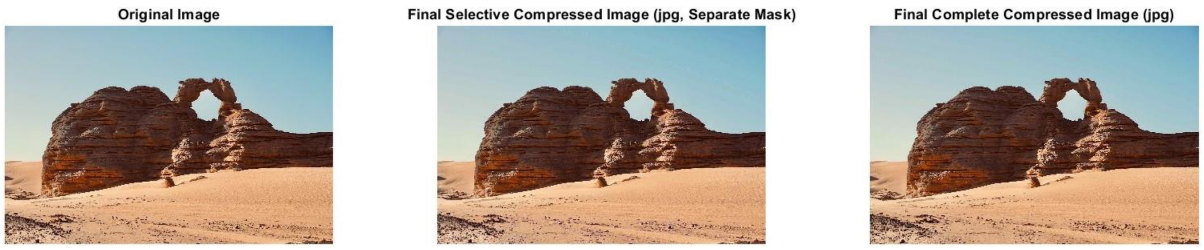
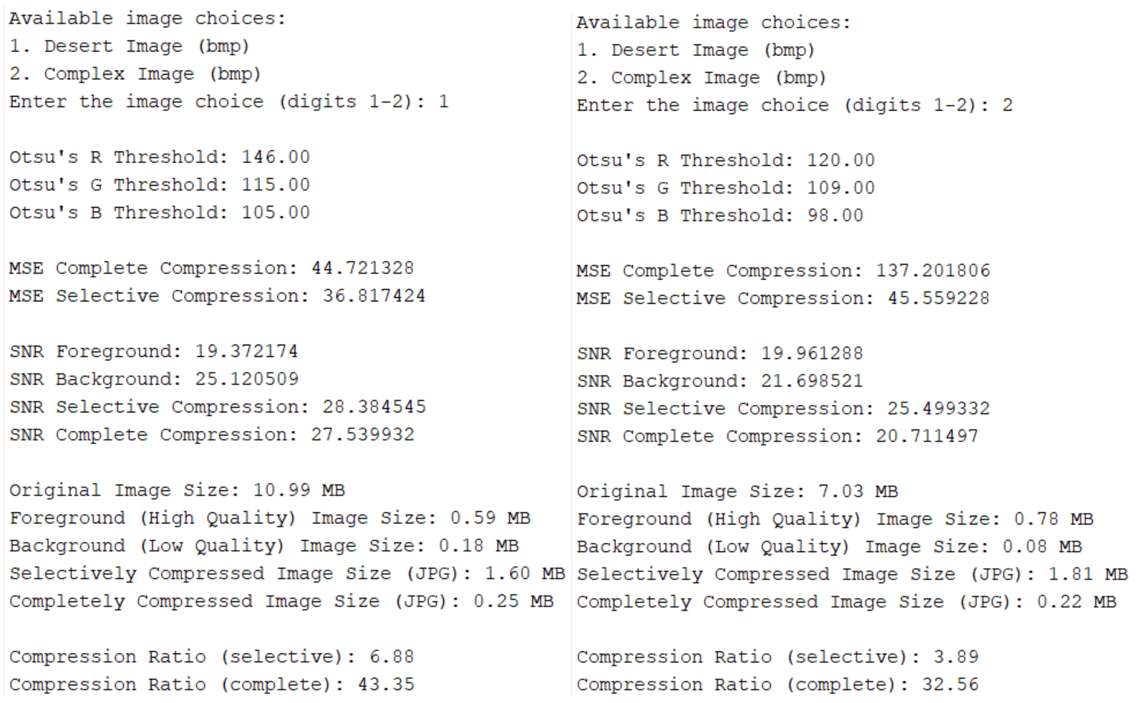

Original Image
Final Selective Compressed Image (jpg, Separate Mask)
Final Complete Compressed Image (jpg)

Fig.8- Original image 1, Selectively compressed and completely compressed images in jpg



```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 1

Otsu's R Threshold: 146.00
Otsu's G Threshold: 115.00
Otsu's B Threshold: 105.00

MSE Complete Compression: 44.721328
MSE Selective Compression: 36.817424

SNR Foreground: 19.372174
SNR Background: 25.120509
SNR Selective Compression: 28.384545
SNR Complete Compression: 27.539932

Original Image Size: 10.99 MB
Foreground (High Quality) Image Size: 0.59 MB
Background (Low Quality) Image Size: 0.18 MB
Selectively Compressed Image Size (JPG): 1.60 MB
Completely Compressed Image Size (JPG): 0.25 MB

Compression Ratio (selective): 6.88
Compression Ratio (complete): 43.35
```

```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 2

Otsu's R Threshold: 120.00
Otsu's G Threshold: 109.00
Otsu's B Threshold: 98.00

MSE Complete Compression: 137.201806
MSE Selective Compression: 45.559228

SNR Foreground: 19.961288
SNR Background: 21.698521
SNR Selective Compression: 25.499332
SNR Complete Compression: 20.711497

Original Image Size: 7.03 MB
Foreground (High Quality) Image Size: 0.78 MB
Background (Low Quality) Image Size: 0.08 MB
Selectively Compressed Image Size (JPG): 1.81 MB
Completely Compressed Image Size (JPG): 0.22 MB

Compression Ratio (selective): 3.89
Compression Ratio (complete): 32.56
```

Fig.9- MATLAB output for thresholding applied on RGB channels individually and JPG used for image compression for image 1 & 2

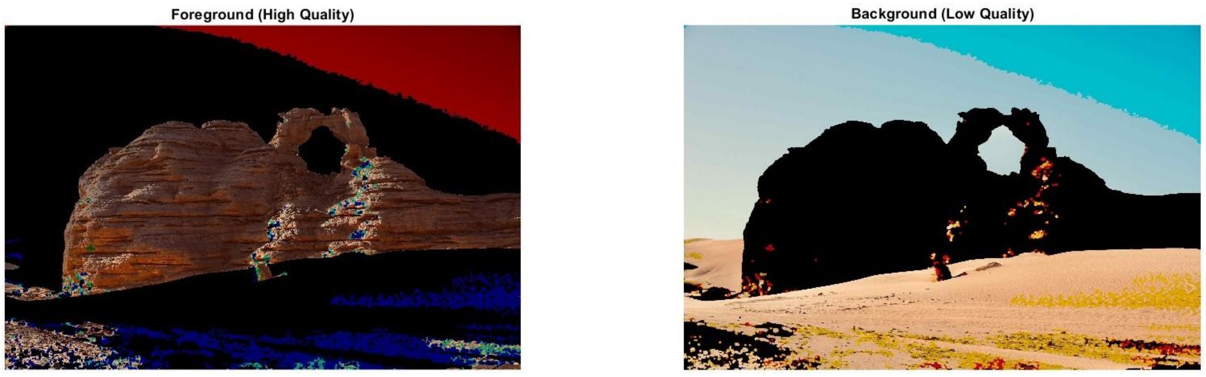
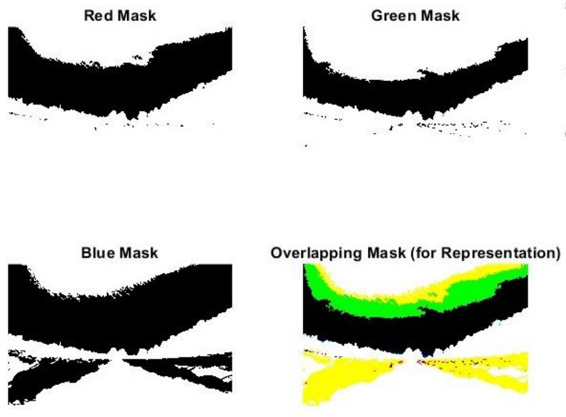


Fig.10- Foreground and background for image 1

For the Desert image, channel-separated segmentation was effective, especially in the green and blue channels, which had bimodal histograms. However, the Red channel's non-bimodal nature caused slight artifacts, such as visible red boundaries. Effective segmentation for the Desert image led to a successful balance of quality and size in selective compression, despite minor artifacts.



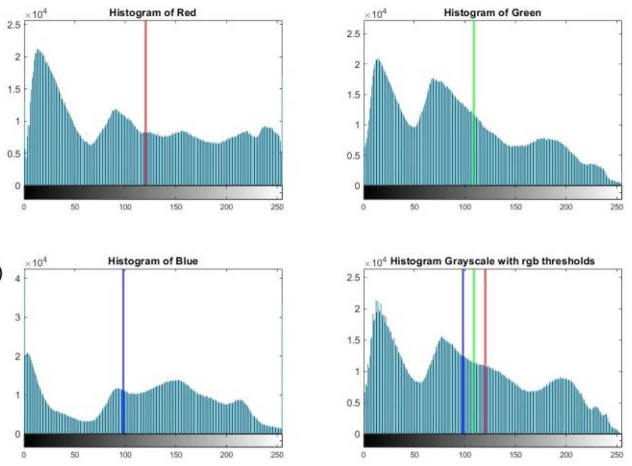


Fig.11- Separate masks for each channel

Fig.12- Histograms of each RGB channel and for grayscale image 1

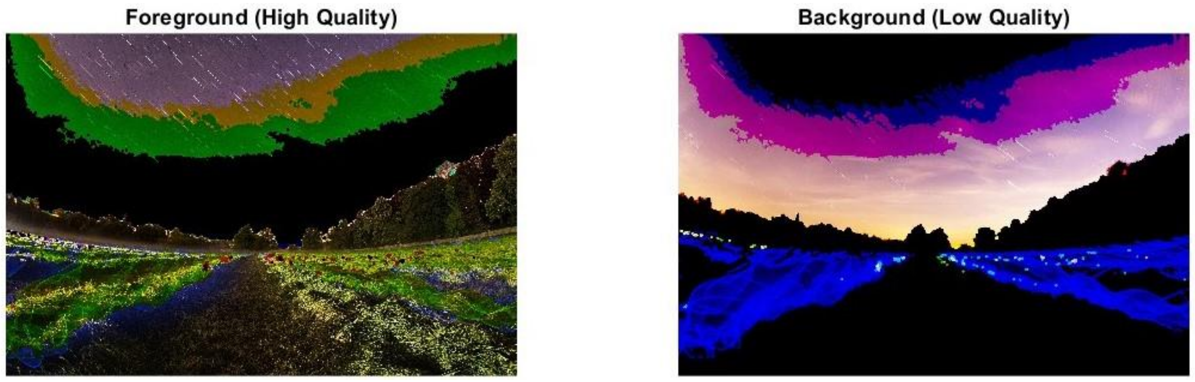


Fig.13- Foreground and Background for image 2

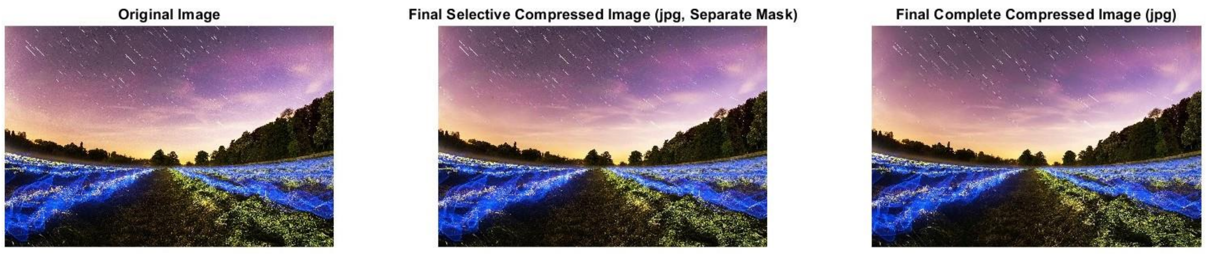
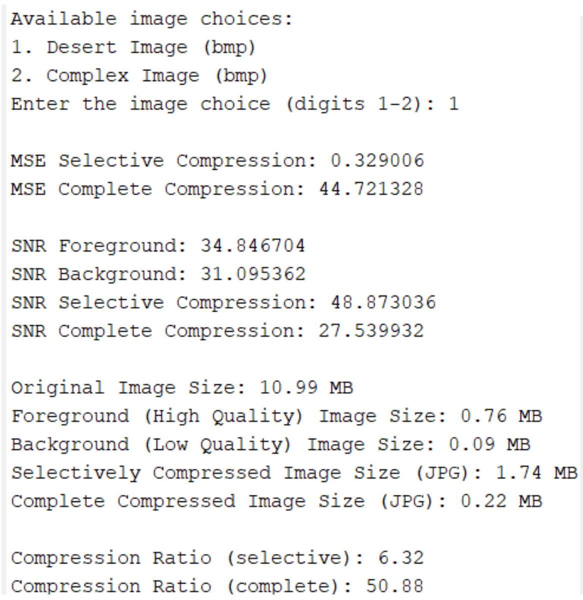

Original Image
Final Selective Compressed Image (jpg, Separate Mask)
Final Complete Compressed Image (jpg)

Fig.14- Original image 2, Selectively and Compressed images

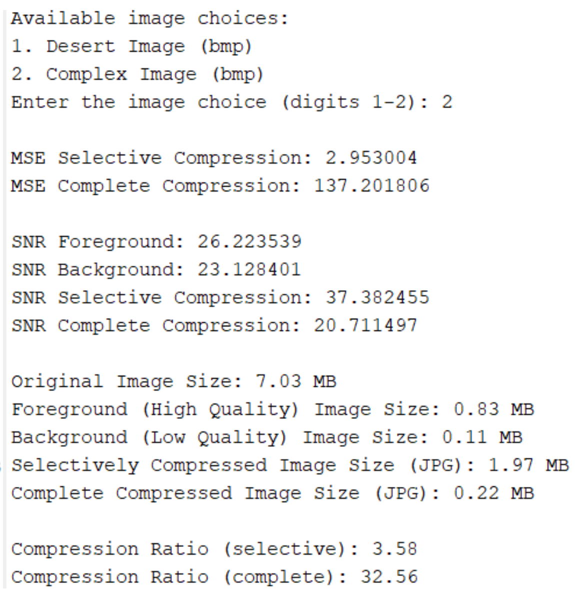
For the Complex image, none of the RGB channels had bimodal histograms, resulting in poor segmentation and inefficient compression. Selective compression fails for the Horse image due to ineffective segmentation.

## III. <u>Segmentation using Canny Edge Detection and Selective JPEG Compression</u>

This part of the experiment applied Canny edge detection to segment images and perform selective JPEG compression.



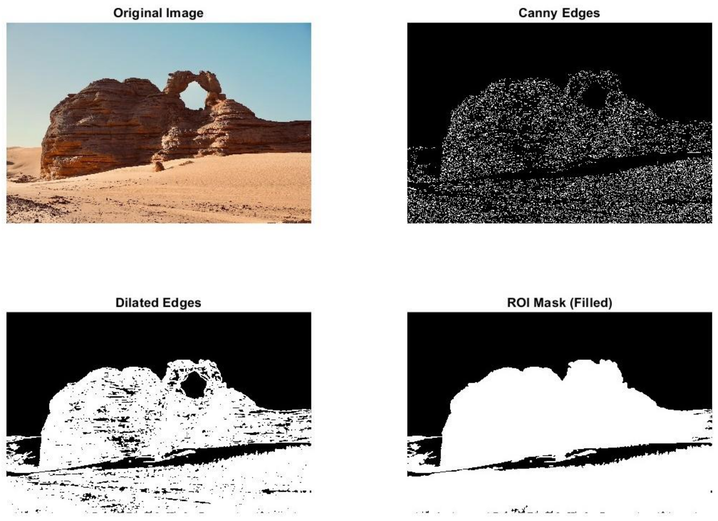
```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 1

MSE Selective Compression: 0.329006
MSE Complete Compression: 44.721328

SNR Foreground: 34.846704
SNR Background: 31.095362
SNR Selective Compression: 48.873036
SNR Complete Compression: 27.539932

Original Image Size: 10.99 MB
Foreground (High Quality) Image Size: 0.76 MB
Background (Low Quality) Image Size: 0.09 MB
Selectively Compressed Image Size (JPG): 1.74 MB
Complete Compressed Image Size (JPG): 0.22 MB

Compression Ratio (selective): 6.32
Compression Ratio (complete): 50.88
```



```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 2

MSE Selective Compression: 2.953004
MSE Complete Compression: 137.201806

SNR Foreground: 26.223539
SNR Background: 23.128401
SNR Selective Compression: 37.382455
SNR Complete Compression: 20.711497

Original Image Size: 7.03 MB
Foreground (High Quality) Image Size: 0.83 MB
Background (Low Quality) Image Size: 0.11 MB
Selectively Compressed Image Size (JPG): 1.97 MB
Complete Compressed Image Size (JPG): 0.22 MB

Compression Ratio (selective): 3.58
Compression Ratio (complete): 32.56
```

Fig.15- MATLAB output for canny edge detection used for segmentation and jpg for image compression

The canny edge detection along with selective JPEG compression on the Desert Image performed extremely well as for previous cases.



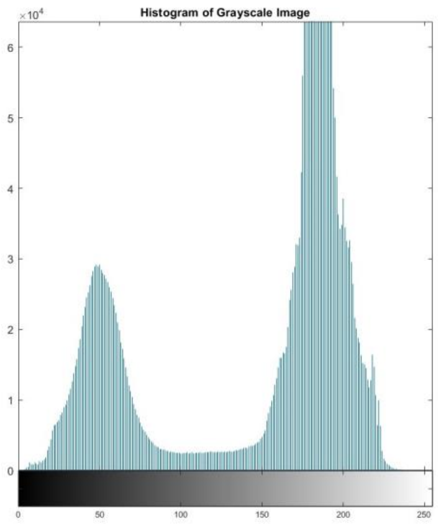


Fig.16- Original image, Canny edges, Edges after dilation and ROI mask

Fig.17- Histogram of the grayscale image 1

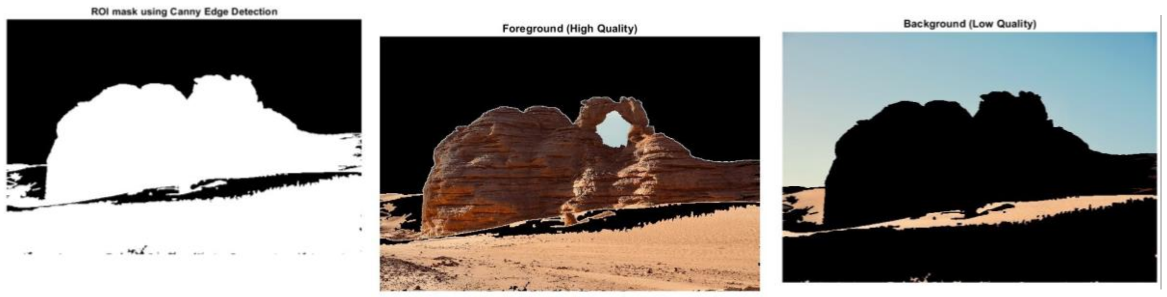

ROI mask using Canny Edge Detection
Foreground (High Quality)
Background (Low Quality)

Fig.18- ROI mask created using Canny Edge detection, Foreground and background for image 1

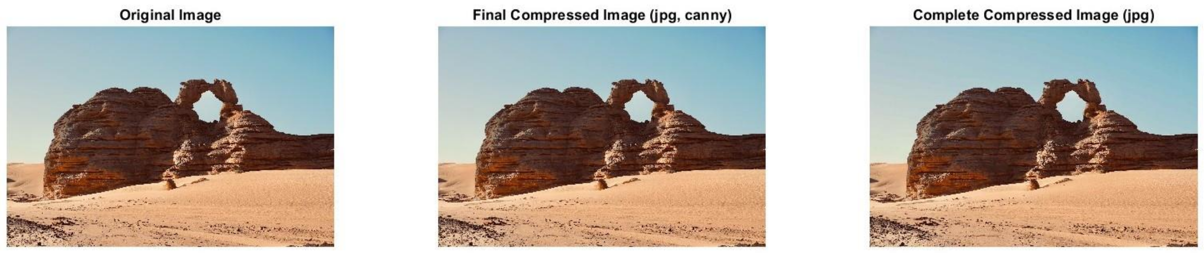
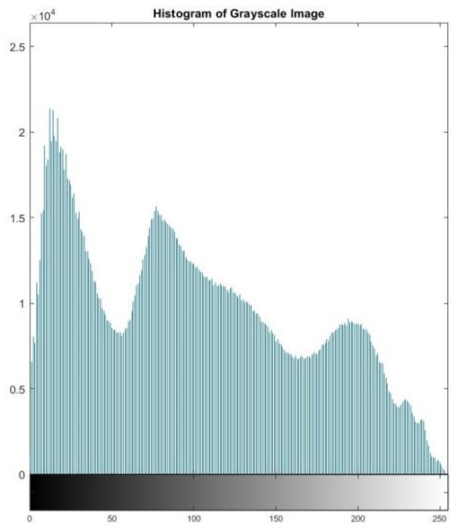

Original Image
Final Compressed Image (jpg, canny)
Complete Compressed Image (jpg)

Fig.19- Original image1, selectively compressed using canny edge detection and jpg for image compression and completely compressed image

The canny edge detection worked better compared to thresholding for the Complex Image and selective compression showed better performance than complete compression. Therefore, demonstrating that canny edge detection can be used on non-bimodal histograms.



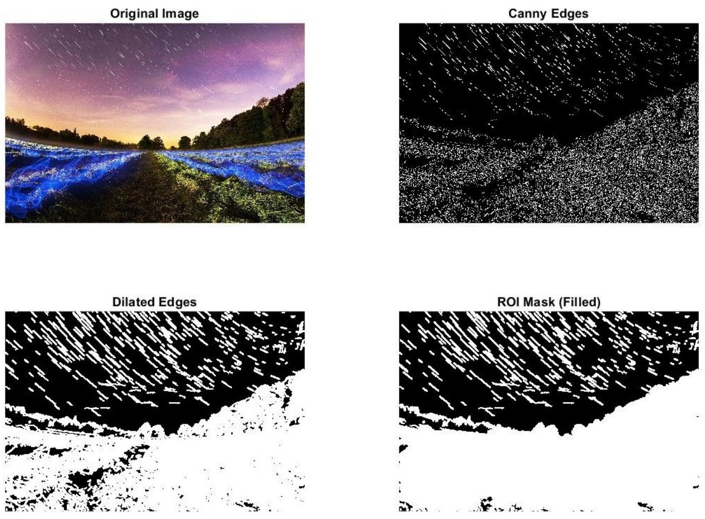


Fig.20- Histogram of grayscale image 2

Fig.21- Original image, Canny edges, Edges after dilation and ROI mask

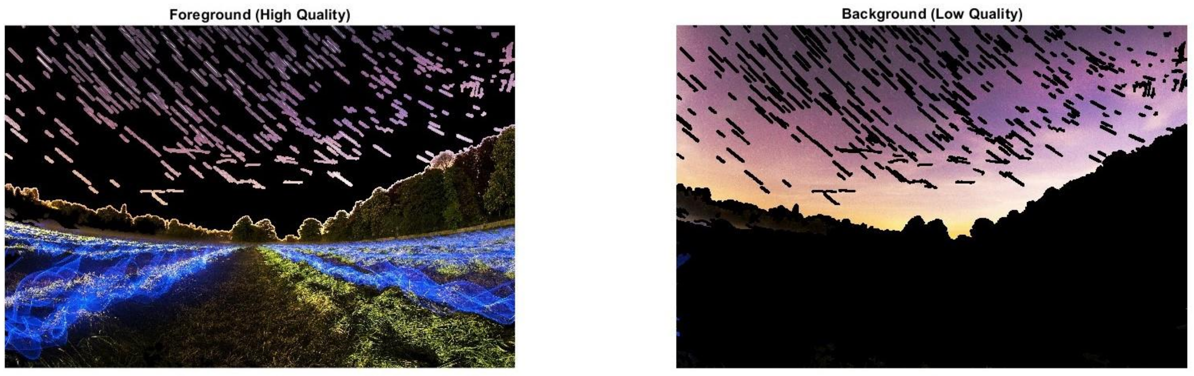


Fig.22- Foreground and Background for image 2

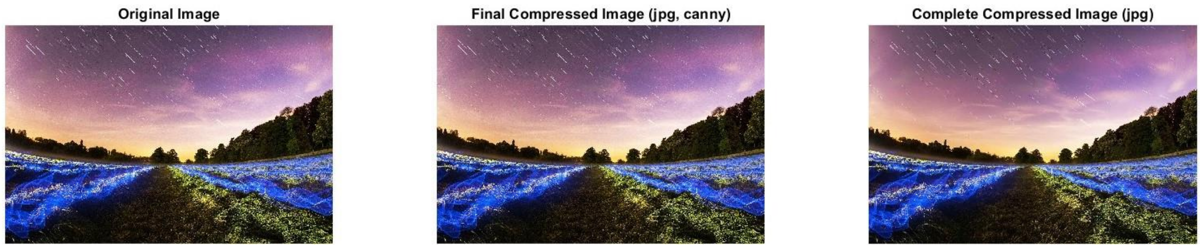
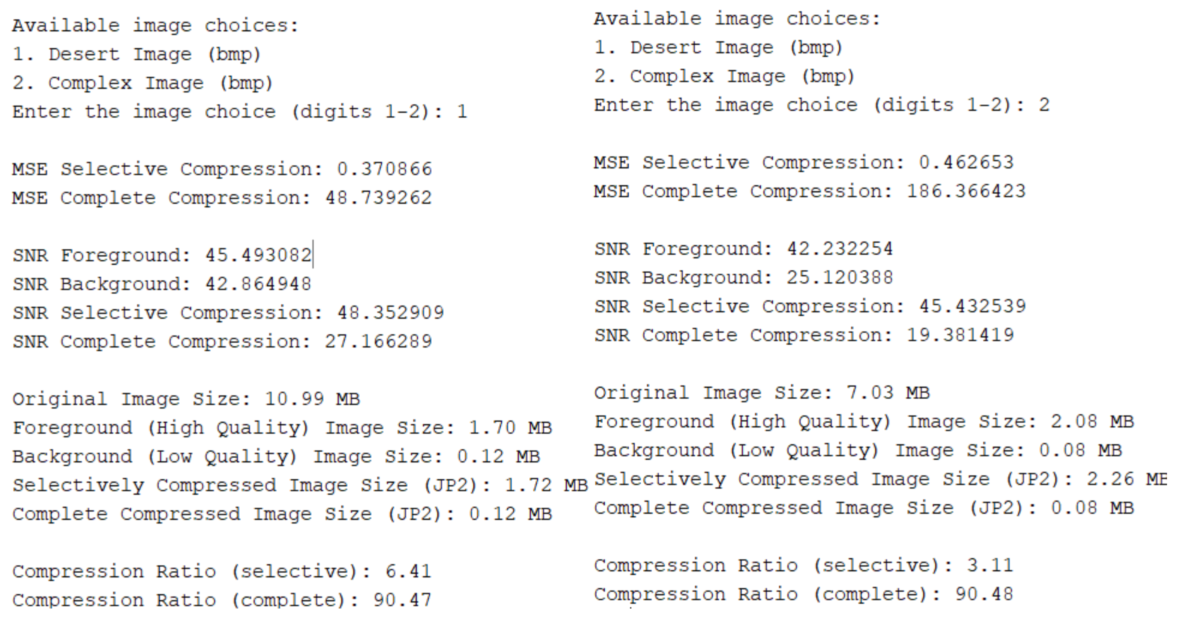

Original Image
Final Compressed Image (jpg, canny)
Complete Compressed Image (jpg)

Fig.23- Original image 2, selectively compressed using canny edge detection and jpg for image compression and completely compressed image

## IV. Canny Edge Detection for Segmentation and Wavelet Transform for Lossless Compression (JP2)

In this case, Canny edge detection was used for segmentation, while the JPEG 2000 (JP2) format, utilizing wavelet transform, was used for compression. This method was applied to both the Desert Image and the Complex Image, with selective (ROI-based) and complete compression scenarios.



```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 1

MSE Selective Compression: 0.370866
MSE Complete Compression: 48.739262

SNR Foreground: 45.493082
SNR Background: 42.864948
SNR Selective Compression: 48.352909
SNR Complete Compression: 27.166289

Original Image Size: 10.99 MB
Foreground (High Quality) Image Size: 1.70 MB
Background (Low Quality) Image Size: 0.12 MB
Selectively Compressed Image Size (JP2): 1.72 MB
Complete Compressed Image Size (JP2): 0.12 MB

Compression Ratio (selective): 6.41
Compression Ratio (complete): 90.47
```

```
Available image choices:
1. Desert Image (bmp)
2. Complex Image (bmp)
Enter the image choice (digits 1-2): 2

MSE Selective Compression: 0.462653
MSE Complete Compression: 186.366423

SNR Foreground: 42.232254
SNR Background: 25.120388
SNR Selective Compression: 45.432539
SNR Complete Compression: 19.381419

Original Image Size: 7.03 MB
Foreground (High Quality) Image Size: 2.08 MB
Background (Low Quality) Image Size: 0.08 MB
Selectively Compressed Image Size (JP2): 2.26 MB
Complete Compressed Image Size (JP2): 0.08 MB

Compression Ratio (selective): 3.11
Compression Ratio (complete): 90.48
```

Fig.24- MATLAB output for canny edge detection used for segmentation and JP2 (Wavelet transform) used for image compression

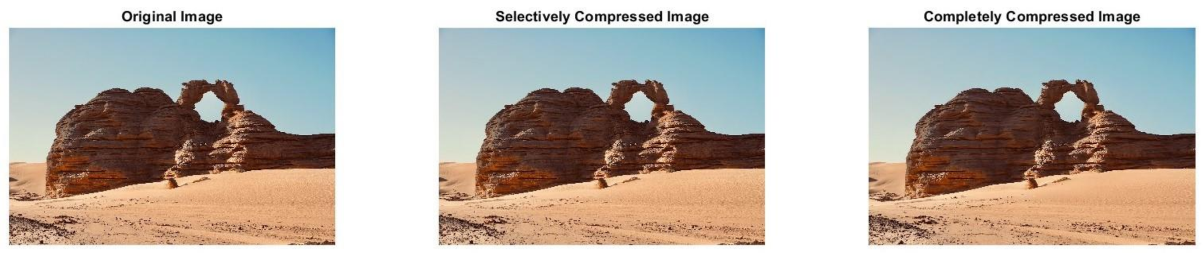


Fig.25- Original image 1, Selectively compressed image using canny edge detection and jp2 for compression, and completely compressed image

For the desert image, the MSE for selective compression was substantially lower (0.371 vs. 48.739), indicating better preservation of details, especially in the foreground. The SNR values for selective compression were higher, confirming this improved quality. Although selective compression resulted in a larger file size (1.72 MB vs. 0.12 MB for complete compression), it achieved a significantly better quality-size trade-off.

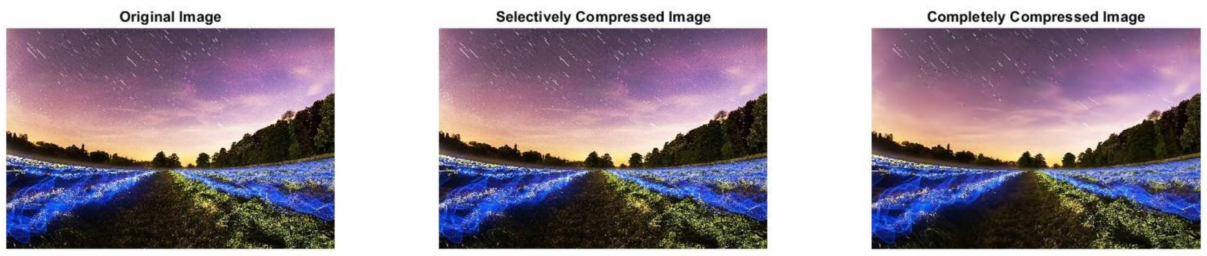


Fig.26- Original image 2, Selectively compressed image using canny edge detection and jp2 for compression, and completely compressed image

For the Complex Image, selective compression performed significantly better in terms of MSE and SNR compared to complete compression. The selective approach yielded a compression ratio of 3.11, which, although smaller, ensured much better-quality preservation due to lossless compression of foreground and lossy compression of background.

## 7. Applications:

Applications of Region-Based Image Compression Using Image Segmentation

1. Medical Imaging
   In medical imaging, critical regions like tumors require higher quality, while surrounding areas can be compressed more aggressively. This method reduces file sizes while ensuring diagnostic accuracy, providing clearer views in scans like MRIs.

2. Remote Sensing and Satellite Imaging
   Region-based compression enables efficient data management in satellite imaging by applying higher compression to less significant areas, such as oceans, while preserving urban or disaster-affected regions. This improves storage and transmission efficiency for large datasets.

3. Surveillance Systems
   In video surveillance, region-based compression prioritizes important areas, like moving objects, allowing them to maintain higher quality. This technique ensures efficient storage and real-time transmission while preserving relevant details in the footage.

4. Multimedia and Video Streaming
   In video streaming, region-based compression selectively compresses less important areas, optimizing bandwidth while maintaining high quality for key regions like faces or objects. This approach enhances the viewing experience without overwhelming the network.

## 8. Limitations and Future scope:

### Limitations

1. Segmentation Accuracy: The accuracy of image segmentation algorithms can significantly impact the performance of region-based compression. Inaccurate segmentation can lead to suboptimal compression results. Thresholding used in the project can struggle with images containing complex textures or varying illumination, leading to inaccurate segmentation and loss of detail in important regions. Also, in case of canny edge detection, it may miss subtle edges or create false positives in noisy images, affecting the reliability of the identified ROIs.
2. Computational Complexity: Region-based compression techniques, especially those involving complex segmentation algorithms, can be computationally expensive, limiting their applicability in real-time applications or on devices with limited processing power.
3. Compression Algorithm Limitations: JPEG compression, particularly with Discrete Cosine Transform (DCT), can introduce noticeable artifacts, especially in regions with significant texture, which can compromise image quality. Similarly, wavelet-based compression, while offering advantages in handling various image types, can be computationally intensive and may require more processing power, making it less suitable for real-time applications. The increased complexity of wavelet transforms can also complicate implementation, particularly in systems with limited computational resources.

### Future Scope

1. Enhancing Segmentation Accuracy:

Integrating advanced machine learning and deep learning techniques can help to improve the precision of region identification. The algorithms can be developed that effectively handle complex textures and varying illumination, ensuring accurate segmentation.

2. Reducing Computational Complexity:

To reduce the computational complexity, more efficient algorithms can be developed that optimize processing time while maintaining segmentation quality. For that, parallel processing and hardware acceleration (e.g., GPUs) can help to lighten computational burdens, making real-time applications feasible even on low-power devices.

3. Improving Compression Algorithms:

For improving compression, the hybrid approaches that combine the strengths of JPEG and wavelet compression can be utilized to enhance performance. Adaptive algorithms can be developed that dynamically select the best compression technique based on image content, minimizing artifacts in textured areas.

4. Implementing User-Driven Adaptive Compression

Allowing users to define priorities for specific regions can facilitate tailored compression strategies. It will help to enhance overall image quality by aligning compression methods with specific application requirements and user preferences.
By addressing these limitations and exploring future research directions, region-based image compression can continue to evolve and find broader applications in various fields.

## 9. References:

[1] B. Brindha and G. Raghuraman, "Region based lossless compression for digital images in telemedicine application," in 2013 International Conference on Communication and Signal Processing, Melmaruvathur, India, 2013, pp. 1-5.

[2] V.-I. Ungureanu, P. Negirla, and A. Korodi, "Image-compression techniques: Classical and 'region-of-interest-based' approaches presented in recent papers," *Sensors*, vol. 24, no. 3, p. 791, 2024.

[3] E. Sophia and J. Anitha, "Implementation of region based medical image compression for telemedicine application," *Proceedings of the International Conference on Computational Intelligence and Communication*, vol. 2015.

[4] MathWorks, "Region of interest based image compression - image analysis," 2024. https://in.mathworks.com/matlabcentral/answers/379804-region-of-interest-based-image-compression-image-analysis.

[5] A. Tiwari and S. Tiwari, "Image compression using MATLAB," *Department of Electronics and Communication Engineering, Rewa Institute of Technology, Ratahara, Rewa, Madhya Pradesh, India*, 2023.

[6] T. Gaur and A. Khanna, "Image compression in MATLAB," International Journal of Scientific & Engineering Research, vol. 8, no. 4, pp. 86-91, Apr. 2017. ISSN 2229-5518.

[7] Pantech Solutions, "Region of interest-based image compression," 2024. https://www.pantechsolutions.net/region-of-interest-based-image-compression.

[8] A. P. Mukherjee, "Region of Interest Based Medical Image Compression," GitHub,2023. https://github.com/ArkaPM/Region-of-Interest-Based-Medical-Image-Compression/blob/master/mm.m.

[9] MathWorks, "Image Thresholding," MathWorks, 2024. https://in.mathworks.com/discovery/image-thresholding.html.

[10] MathWorks, "Edge Detection," MathWorks, 2024. https://in.mathworks.com/help/images/edge-detection.html.

[11] MathWorks, "Reading, Writing, and Querying Graphics Image Files," MathWorks, 2024. [Online]. Available: https://in.mathworks.com/help/matlab/creating_plots/reading-writing-and-querying-graphics-image-files.html.

[12] MathWorks, "Save image with compression (JPEG 2000)," MathWorks, 2024. https://in.mathworks.com/matlabcentral/answers/529003-save-image-with-compression-jpeg-2000.