```
%date 26/01/2026
%created by - Kaushik Kumar Umar

clc;
clear;
close all;

% Read a color image
rgb_img = imread('input.jpg');
rgb_img=imrotate(rgb_img,90);

% Resize image to reduce computation

figure;
imshow(rgb_img);
title('Original Color Image');

% Convert RGB image to grayscale
gray_img = rgb2gray(rgb_img);
[rows, cols] = size(gray_img);

figure;
imshow(gray_img);
title('Grayscale Image');

% Convert grayscale image into column vector
img_vec = gray_img(:);
img_data = img_vec;
% Find unique gray levels
symbols = unique(img_vec);

% Calculate frequency of each symbol
freq = histcounts(img_vec, [symbols; max(symbols)+1]);

% Calculate probability
prob = freq / sum(freq);

% Sort symbols based on probability
[prob_sorted, idx] = sort(prob, 'descend');
symbols_sorted = symbols(idx);

% Shannon code length calculation
code_len = ceil(-log2(prob_sorted));

% Generate Shannon codes
shannon_code = cell(length(symbols_sorted),1);
cum_prob = 0;

for i = 1:length(symbols_sorted)
    shannon_code{i} = dec2bin(floor(cum_prob * 2^code_len(i)), code_len(i));
    cum_prob = cum_prob + prob_sorted(i);
end

% Encoding
encoded_bits = cell(length(img_vec),1);

for i = 1:length(img_vec)
    symbol_index = find(symbols_sorted == img_vec(i), 1);
    encoded_bits{i} = shannon_code{symbol_index};
end

% Final encoded bitstream
encoded_stream = [encoded_bits{:}];

% Reconstructed image
reconstructed_img = reshape(img_vec, rows, cols);

figure;
imshow(reconstructed_img);
title('Reconstructed Image after Shannon Coding');

% Compression calculation
original_bits   = numel(img_vec) * 8;
compressed_bits = length(encoded_stream);
compression_ratio = original_bits / compressed_bits;
fprintf('By using Shannon Method\n');
fprintf('Original size (bits)   : %d\n', original_bits);
fprintf('Compressed size (bits) : %d\n', compressed_bits);
fprintf('Compression Ratio      : %.2f\n', compression_ratio);


% Huffman Coding

% Find pixel values and their probabilities
pixel_values = unique(img_data);
pixel_count  = histc(img_data, pixel_values);
pixel_prob   = pixel_count / sum(pixel_count);

% Create Huffman dictionary based on probabilities
```

```
huff_dict = huffmandict(pixel_values, pixel_prob);

% Encode the grayscale image using Huffman coding
encoded_stream = huffmanenco(img_data, huff_dict);

% Decode the Huffman encoded data
decoded_data = huffmandeco(encoded_stream, huff_dict);

% Convert decoded data back into image form
decoded_img = uint8(reshape(decoded_data, rows, cols));

figure;
imshow(decoded_img);
title('Reconstructed Image after Huffman Coding');

% Calculate compression ratio
original_bits   = numel(img_data) * 8;
compressed_bits = length(encoded_stream);

compression_ratio = original_bits / compressed_bits;
fprintf('By using Huffman Method\n');
fprintf('Original size (bits)   : %d\n', original_bits);
fprintf('Compressed size (bits) : %d\n', compressed_bits);
fprintf('Compression Ratio      : %.2f\n', compression_ratio);
```

```
By using Shannon Method
Original size (bits)   : 7701504
Compressed size (bits) : 7923467
Compression Ratio      : 0.97
By using Huffman Method
Original size (bits)   : 7701504
Compressed size (bits) : 7374582
Compression Ratio      : 1.04
```

Original Color Image

Grayscale Image

Reconstructed Image after Shannon Coding

Reconstructed Image after Huffman Coding

Published with MATLAB® R2021a