Exp No. 09 : JPEG COMPRESSION USING DCT

NAME : VINAY MALKAR
REG NO. 21BEC1430

DATE : 03-10-2024

Task 01 : Compress the image and verify the MSE value

Steps :

- Read a color image, convert into gray
- Apply 2D DCT on each 8x8 block of gray image
- Quantize the DCT coefficients
- Use inverse DCT to reconstruct
- Verify the MSE value for original image and reconstruct image
- Show all the images

Task 02 : Set all the components to zero except the DC coefficient DCT and verify the output, check the quality

Task 03 – Consider a 50x50 subimage and encode it using Huffman Coding Technique

Matlab Code :

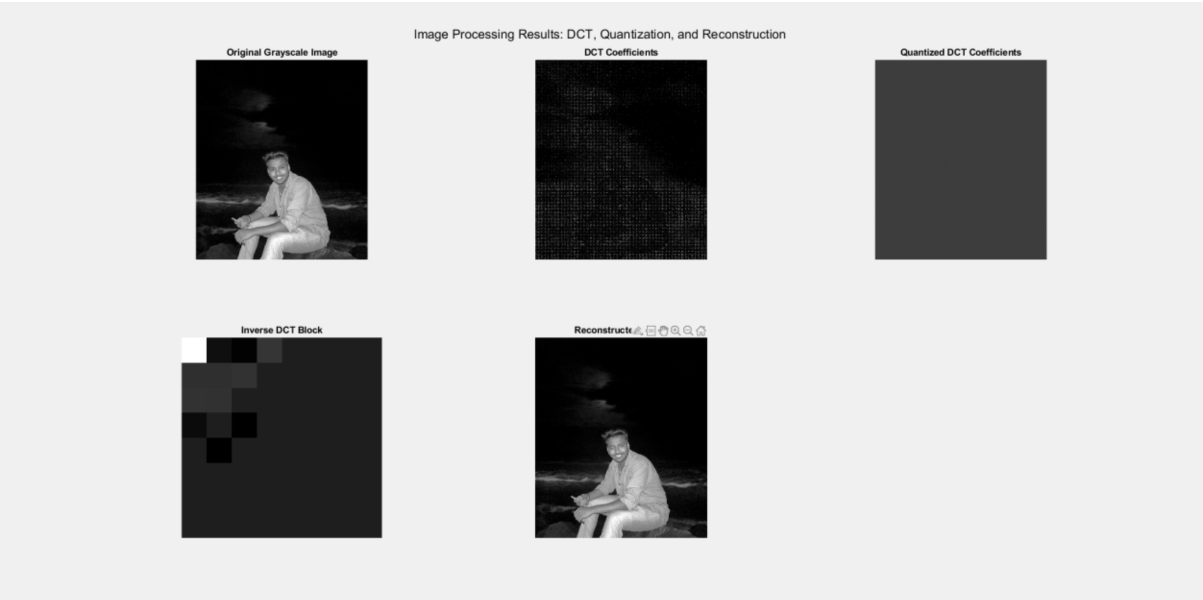
Task 01 Matlab Code :

```
% Read a color image and convert it to grayscale
original_image = imread("D:\21bec1430\SAVE_20240708_220044.JPG");  %
Replace with the image path
gray_image = rgb2gray(original_image);

% Parameters
block_size = 8;  % Block size for 8x8 DCT
Q = [16 11 10 16 24 40 51 61;
     12 12 14 19 26 58 60 55;
     14 13 16 24 40 57 69 56;
     14 17 22 29 51 87 80 62;
     18 22 37 56 68 109 103 77;
     24 35 55 64 81 104 113 92;
     49 64 78 87 103 121 120 101;
     72 92 95 98 112 100 103 99];  % Quantization matrix for
compression

% Get the size of the image
[rows, cols] = size(gray_image);

% Initialize arrays for DCT coefficients, quantized coefficients,
reconstructed image
dct_coefficients = zeros(rows, cols);
quantized_coefficients = zeros(rows, cols);
reconstructed_image = zeros(rows, cols);
```

```
% Apply block-wise DCT and Quantization
for i = 1:block_size:rows
    for j = 1:block_size:cols
        % Check if block exceeds image boundaries
        if i + block_size - 1 <= rows && j + block_size - 1 <= cols
            % Extract the 8x8 block
            block = double(gray_image(i:i + block_size - 1, j:j +
block_size - 1));

            % Apply 2D DCT on the block
            dct_block = dct2(block);

            % Quantize the DCT coefficients
            quantized_block = round(dct_block ./ Q);

            % Store the DCT coefficients and quantized coefficients
            dct_coefficients(i:i + block_size - 1, j:j + block_size
- 1) = dct_block;
            quantized_coefficients(i:i + block_size - 1, j:j +
block_size - 1) = quantized_block;

            % Inverse Quantization
            dequantized_block = quantized_block .* Q;

            % Apply inverse DCT to reconstruct the block
            reconstructed_block = idct2(dequantized_block);

            % Store the reconstructed block in the image
            reconstructed_image(i:i + block_size - 1, j:j +
block_size - 1) = reconstructed_block;
        end
    end
end

% Convert reconstructed image to uint8
reconstructed_image = uint8(reconstructed_image);

% Calculate MSE between original and reconstructed images
mse_value = immse(gray_image, reconstructed_image);

% Display MSE
fprintf('Mean Squared Error (MSE) between original and reconstructed
image: %0.4f\n', mse_value);

% Show the images for comparison
figure;

% Original Grayscale Image
subplot(2, 3, 1);
imshow(gray_image);
title('Original Grayscale Image');

% DCT Coefficients (Display absolute values for visibility)
subplot(2, 3, 2);
```

```
imshow(log(abs(dct_coefficients) + 1), []);  % Use logarithm for better visibility
title('DCT Coefficients');

% Quantized DCT Coefficients
subplot(2, 3, 3);
imshow(quantized_coefficients, []);  % Display quantized coefficients
title('Quantized DCT Coefficients');

% Inverse DCT Image (after dequantization)
subplot(2, 3, 4);
imshow(dequantized_block, []);  % Display the last dequantized block
title('Inverse DCT Block');

% Reconstructed Image
subplot(2, 3, 5);
imshow(reconstructed_image);
title('Reconstructed Image');

% Display layout adjustments
sgtitle('Image Processing Results: DCT, Quantization, and Reconstruction');
```

Matlab Output Image :



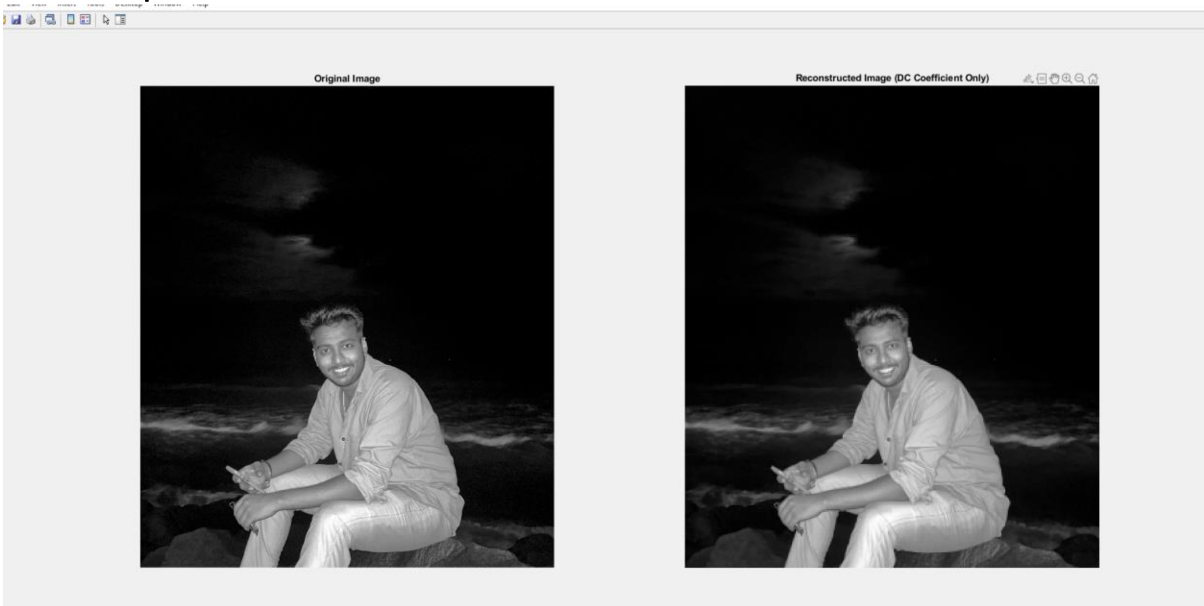

```
>> ExpNo8
Mean Squared Error (MSE) between original and reconstructed image: 11.7246
```

Task 02 Code :

```
image_path = 'D:\21bec1430\SAVE_20240708_220044.JPG';

% Read and convert the color image to grayscale
original_image = imread(image_path);
gray_image = rgb2gray(original_image);


% Parameters
block_size = 8;  % Block size for 8x8 DCT

% Get the size of the image
[rows, cols] = size(gray_image);

% Initialize an array for storing DCT coefficients and reconstructed
image
dct_coefficients = zeros(rows, cols);
reconstructed_image_dc_only = zeros(rows, cols);

% Process the image block by block
for i = 1:block_size:rows
    for j = 1:block_size:cols
        % Ensure the block fits within the image bounds
        if i+block_size-1 <= rows && j+block_size-1 <= cols
            % Extract the 8x8 block
            block = double(gray_image(i:i+block_size-1,
j:j+block_size-1));

            % Apply 2D DCT on the block
            dct_block = dct2(block);

            % Set all components to zero except the DC coefficient
(top-left element)
            dct_dc_only = zeros(block_size, block_size);
            dct_dc_only(1, 1) = dct_block(1, 1);  % Keep only the DC
coefficient

            % Store the modified DCT coefficients (for visualization
if needed)
            dct_coefficients(i:i+block_size-1, j:j+block_size-1) =
dct_dc_only;

            % Apply inverse DCT using only the DC coefficient
            reconstructed_block_dc_only = idct2(dct_dc_only);

            % Store the reconstructed block in the final image
            reconstructed_image_dc_only(i:i+block_size-1,
j:j+block_size-1) = reconstructed_block_dc_only;
        end
    end
end

% Convert the reconstructed image to uint8
reconstructed_image_dc_only = uint8(reconstructed_image_dc_only);
```

```
% Calculate MSE between original and reconstructed images
mse_dc_only = immse(gray_image, reconstructed_image_dc_only);

% Display MSE value
fprintf('MSE between original and DC-only reconstructed image:
%0.4f\n', mse_dc_only);

% Show the original and DC-only reconstructed images side by side
for quality comparison
figure;
subplot(1, 2, 1);
imshow(gray_image);
title('Original Image');

subplot(1, 2, 2);
imshow(reconstructed_image_dc_only);
title('Reconstructed Image (DC Coefficient Only)');
```

Matlab Output :



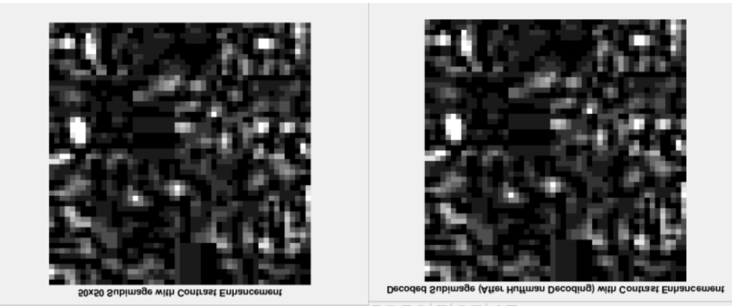

```
>> ExpNo8Task2
MSE between original and DC-only reconstructed image: 40.2459
```

Task 03 Matlab Code :

```
% Task: Extract a 50x50 subimage and encode it using Huffman coding

% Image location
image_path = 'D:\21bec1430\SAVE_20240708_220044.JPG';

% Read and convert the image to grayscale
original_image = imread(image_path);
gray_image = rgb2gray(original_image);

% Extract a 50x50 subimage (top-left corner)
sub_image = gray_image(1:50, 1:50);

% Apply contrast enhancement to brighten the image for better
visibility
sub_image = imadjust(sub_image);  % Adjust contrast of the subimage

% Display the 50x50 subimage with increased zoom
figure;
imshow(sub_image, 'InitialMagnification', 800);  % Zoom in 800%
title('50x50 Subimage with Contrast Enhancement');

% Flatten the subimage to a 1D vector
sub_image_vector = sub_image(:);

% Calculate the frequency of each unique pixel value in the range 0-
255
pixel_values = 0:255;
pixel_freq = histcounts(sub_image_vector, [pixel_values, 256]); %
Pixel frequencies

% Normalize frequencies to probabilities
probabilities = pixel_freq / sum(pixel_freq);

% Remove zero-probability values to avoid issues in Huffman tree
valid_pixel_values = pixel_values(probabilities > 0);
valid_probabilities = probabilities(probabilities > 0);

% Build the Huffman dictionary based on valid pixel values
dict = huffmandict(valid_pixel_values, valid_probabilities);

% Manually encode the subimage using Huffman codes
encoded_sub_image = [];
for k = 1:length(sub_image_vector)
    pixel = sub_image_vector(k);
    code = dict{valid_pixel_values == pixel, 2};  % Find Huffman
code for the pixel
    encoded_sub_image = [encoded_sub_image, code];  % Concatenate
Huffman code
end

% Show encoded result length (number of bits)
fprintf('Original subimage size: %d pixels\n',
numel(sub_image_vector));
```

```
fprintf('Encoded subimage length: %d bits\n',
length(encoded_sub_image));

% Manually decode the encoded subimage
decoded_sub_image_vector = zeros(size(sub_image_vector));
index = 1;  % Index to traverse the encoded string
for k = 1:length(decoded_sub_image_vector)
    % Decode one pixel at a time by matching the Huffman codes
    for j = 1:size(dict, 1)
        code = dict{j, 2};  % Get Huffman code
        len = length(code);
        if index+len-1 <= length(encoded_sub_image) && ...
           isequal(encoded_sub_image(index:index+len-1), code)
            decoded_sub_image_vector(k) = dict{j, 1};  % Assign
pixel value
            index = index + len;
            break;
        end
    end
end

% Reshape the decoded 1D vector back into a 50x50 matrix
decoded_sub_image = reshape(decoded_sub_image_vector, 50, 50);

% Apply contrast enhancement to decoded image for better visibility
decoded_sub_image = imadjust(uint8(decoded_sub_image));

% Display the decoded subimage with increased zoom
figure;
imshow(decoded_sub_image, 'InitialMagnification', 800);  % Zoom in
800%
title('Decoded Subimage (After Huffman Decoding) with Contrast
Enhancement');

% Check if original and decoded images are identical
if isequal(sub_image, uint8(decoded_sub_image))
    disp('Original and decoded subimage are identical.');
else
    disp('Original and decoded subimage are NOT identical.');
end
```

<u>Matlab Output :</u>



```
>> Expno8task3
Original subimage size: 2500 pixels
Encoded subimage length: 6444 bits
Original and decoded subimage are identical.
```