LAB-1

```
% Huffman Encoding and Decoding in MATLAB (for MATLAB 2014a)
% This version keeps left child 0 and right child 1 for Huffman
Encoding.

function huffman_main()
    % Input text
    inputText = input('Enter the text to encode and decode using
Huffman coding: ', 's');

    % Huffman Encoding
     [encodedText, codeTable] = huffmanEncoding(inputText);
    disp(['Encoded Text: ', encodedText]);

    % Display Huffman Codes
    disp('Huffman Codes:');
    disp(codeTable);

    % Huffman Decoding
    decodedText = huffmanDecoding(encodedText, codeTable);
    disp(['Decoded Text: ', decodedText]);

    % Efficiency Calculation
    efficiency = calculateEfficiency(inputText, codeTable);
    disp(['Efficiency: ', num2str(efficiency)]);
end

function [encodedText, codeTable] = huffmanEncoding(inputText)
    % Calculate frequencies
    symbols = unique(inputText);
    freq = zeros(1, length(symbols));
```

```
    for i = 1:length(symbols)
        freq(i) = sum(inputText == symbols(i));
    end

    % Create a priority queue (min-heap)
    pq = cell(length(symbols), 1);
    for i = 1:length(symbols)
        pq{i} = struct('symbol', symbols(i), 'freq', freq(i),
'left', [], 'right', []);
    end

    % Build Huffman tree (lowest frequency first)
    while length(pq) > 1
        % Find two nodes with minimum frequency
        [~, idx1] = min(cellfun(@(x) x.freq, pq));
        left = pq{idx1};
        pq(idx1) = [];

        [~, idx2] = min(cellfun(@(x) x.freq, pq));
        right = pq{idx2};
        pq(idx2) = [];

        % Create a new node with the sum of frequencies
        newNode = struct('symbol', [], 'freq', left.freq +
right.freq, 'left', left, 'right', right);
        pq = [pq; newNode];  % Add the new node back to the
priority queue
    end
    huffmanTree = pq{1};  % The root of the Huffman tree

    % Generate Huffman codes
    codeTable = generateCodes(huffmanTree, '');
```

```
    % Encode text using the Huffman codes
    encodedText = '';
    for i = 1:length(inputText)
        idx = find(strcmp(codeTable(:, 1), inputText(i)));
        encodedText = strcat(encodedText, codeTable{idx, 2});
    end
end

% Recursive function to generate Huffman codes
function codes = generateCodes(node, code)
    if isempty(node.symbol)  % Internal node
        % Traverse left child (0) and right child (1)
        leftCodes = generateCodes(node.left, strcat(code, '1'));
        rightCodes = generateCodes(node.right, strcat(code,
'0'));
        codes = [leftCodes; rightCodes];
    else  % Leaf node
        codes = {node.symbol, code};  % Store the symbol and its
corresponding code
    end
end

function decodedText = huffmanDecoding(encodedText, codeTable)
    symbols = codeTable(:, 1);
    codes = codeTable(:, 2);
    currentCode = '';
    decodedText = '';

    % Decode the encoded text using the Huffman codes
    for i = 1:length(encodedText)
        currentCode = strcat(currentCode, encodedText(i));
```

```
        matchIdx = find(strcmp(codes, currentCode));
        if ~isempty(matchIdx)
            decodedText = strcat(decodedText,
symbols{matchIdx});
            currentCode = '';  % Reset current code for next
symbol
        end
    end
end

function efficiency = calculateEfficiency(inputText, codeTable)
    originalBits = length(inputText) * 8;  % ASCII encoding
    huffmanBits = 0;
    symbols = codeTable(:, 1);
    codes = codeTable(:, 2);

    % Calculate the number of bits used in the Huffman encoding
    for i = 1:length(inputText)
        idx = find(strcmp(symbols, inputText(i)));
        huffmanBits = huffmanBits + length(codes{idx});
    end

    efficiency = (originalBits - huffmanBits)/originalBits*100;
% Efficiency calculation
end
```

LAB-2

```
% LZ78 Encoding and Huffman Encoding for Text Data in MATLAB

function lz78_main()
    % Input text
    inputText = 'lempel ziv coding algorithm comparison';

    % Huffman Encoding
    [huffmanEncodedText, huffmanCodeTable] = huffmanEncoding(inputText);
    disp(['Huffman Encoded Text: ', huffmanEncodedText]);

    % LZ78 Encoding
    [lz78EncodedText, lz78Dictionary] = lz78Encoding(inputText);
    disp(['LZ78 Encoded Text: ', lz78EncodedText]);

    % Calculate Efficiency
    huffmanEfficiency = calculateEfficiency(inputText, huffmanCodeTable);
    lz78Efficiency = calculateLZ78Efficiency(inputText, lz78EncodedText);

    % Display Results
    disp(['Huffman Efficiency: ', num2str(huffmanEfficiency)]);
    disp(['LZ78 Efficiency: ', num2str(lz78Efficiency)]);
end

% Function to implement Huffman Encoding
function [encodedText, codeTable] = huffmanEncoding(inputText)
```

```
    % Calculate frequencies
    symbols = unique(inputText);
    freq = zeros(1, length(symbols));
    for i = 1:length(symbols)
        freq(i) = sum(inputText == symbols(i));
    end

    % Create a priority queue (min-heap)
    pq = cell(length(symbols), 1);
    for i = 1:length(symbols)
        pq{i} = struct('symbol', symbols(i), 'freq', freq(i),
'left', [], 'right', []);
    end

    % Build Huffman tree
    while length(pq) > 1
        [~, idx1] = min(cellfun(@(x) x.freq, pq));
        left = pq{idx1};
        pq(idx1) = [];

        [~, idx2] = min(cellfun(@(x) x.freq, pq));
        right = pq{idx2};
        pq(idx2) = [];

        newNode = struct('symbol', [], 'freq', left.freq +
right.freq, 'left', left, 'right', right);
        pq = [pq; newNode];
    end
    huffmanTree = pq{1};

    % Generate Huffman codes
    codeTable = generateCodes(huffmanTree, '');
```

```
    % Encode text using the Huffman codes
    encodedText = '';
    for i = 1:length(inputText)
        idx = find(strcmp(codeTable(:, 1), inputText(i)));
        encodedText = strcat(encodedText, codeTable{idx, 2});
    end
end

% Function to recursively generate Huffman codes
function codes = generateCodes(node, code)
    if isempty(node.symbol)
        leftCodes = generateCodes(node.left, strcat(code, '0'));
        rightCodes = generateCodes(node.right, strcat(code,
'1'));
        codes = [leftCodes; rightCodes];
    else
        codes = {node.symbol, code};
    end
end

% LZ78 Encoding Implementation
function [encodedText, dictionary] = lz78Encoding(inputText)
    dictionary = {};  % Initialize empty dictionary
    encodedText = '';  % Output encoded text
    idx = 1;  % Index for dictionary
    i = 1;  % Pointer to traverse the input text

    while i <= length(inputText)
        % Find the longest substring starting at position i
        substring = '';
```

```
        while i <= length(inputText) && ismember(substring,
dictionary)
            substring = [substring, inputText(i)];
            i = i + 1;
        end

        % Add the current substring to the dictionary if it is
new
        dictionary{idx} = substring;
        encodedText = strcat(encodedText, num2str(idx), ',',
substring, ' '); % Store (index, substring) pairs
        idx = idx + 1;
    end
end

% Function to calculate Huffman Efficiency
function efficiency = calculateEfficiency(inputText, codeTable)
    originalBits = length(inputText) * 8; % ASCII encoding
    huffmanBits = 0;
    symbols = codeTable(:, 1);
    codes = codeTable(:, 2);

    for i = 1:length(inputText)
        idx = find(strcmp(symbols, inputText(i)));
        huffmanBits = huffmanBits + length(codes{idx});
    end

    efficiency = (originalBits - huffmanBits)/originalBits*100;
% Efficiency calculation
end

% Function to calculate LZ78 Efficiency
```

```
function efficiency = calculateLZ78Efficiency(inputText,
lz78EncodedText)
    originalBits = length(inputText) * 8;  % ASCII encoding
    lz78Bits = length(lz78EncodedText) ; % Number of bits in
LZ78 encoded text (each symbol and index is encoded as bits)
    efficiency = (originalBits/ lz78Bits);  % Efficiency
calculation
end
```

LAB-3

```
function compareFT_DCT()
    % Read an example image (e.g., 'pial.jpg')
    [file, path] = uigetfile({'*.jpg;*.png;*.bmp;*.tif', 'Image
Files (*.jpg, *.png, *.bmp, *.tif)'}, 'Select a Color Image');

    img = imread(fullfile(path, file));
    img = rgb2gray(img);  % Convert to grayscale if it's in
color
    subplot(1,3,1);
    imshow(img);

    % Normalize the image to [0, 1] for computation
    img = double(img) / 255;

    % Image Compression using Fourier Transform (FT)
    [ftCompressedImg, ftCompressionRatio] = ftCompression(img);
    subplot(1,3,2);
    imshow(ftCompressedImg, []);
    title('FT Compressed Image');
    disp(['FT Compression Ratio: ',
num2str(ftCompressionRatio)]);

    % Image Compression using Discrete Cosine Transform (DCT)
    [dctCompressedImg, dctCompressionRatio] =
dctCompression(img);
    subplot(1,3,3);
    imshow(dctCompressedImg, []);
    title('DCT Compressed Image');
    disp(['DCT Compression Ratio: ',
num2str(dctCompressionRatio)]);
```

```
    % Calculate MSE or PSNR for quality comparison
    ftMSE = mean((img(:) - ftCompressedImg(:)).^2);
    dctMSE = mean((img(:) - double(dctCompressedImg(:))).^2); %
Convert to double
    disp(['FT MSE: ', num2str(ftMSE)]);
    disp(['DCT MSE: ', num2str(dctMSE)]);

    % Display PSNR
    ftPSNR = 10 * log10(1 / ftMSE);
    dctPSNR = 10 * log10(1 / dctMSE);
    disp(['FT PSNR: ', num2str(ftPSNR)]);
    disp(['DCT PSNR: ', num2str(dctPSNR)]);
end

% Fourier Transform Compression Function
function [compressedImg, compressionRatio] = ftCompression(img)
    % Perform 2D FFT
    imgFFT = fft2(img);

    % Retain only the top 10% most significant coefficients
    threshold = 0.1;
    magnitude = abs(imgFFT);
    sortedMagnitude = sort(magnitude(:), 'descend');
    thresholdValue = sortedMagnitude(round(threshold *
numel(sortedMagnitude)));

    imgFFT(abs(imgFFT) < thresholdValue) = 0;

    % Inverse FFT to get compressed image
    compressedImg = real(ifft2(imgFFT));
```

```
    % Calculate compression ratio
    originalSize = numel(img);
    compressedSize = sum(abs(imgFFT(:)) > thresholdValue);
    compressionRatio = originalSize / compressedSize;
end

% DCT Compression Function
function [compressedImg, compressionRatio] = dctCompression(img)
    % Perform 2D DCT
    imgDCT = dct2(img);

    % Retain only the top 10% most significant coefficients
    threshold = 0.1;
    magnitude = abs(imgDCT);
    sortedMagnitude = sort(magnitude(:), 'descend');
    thresholdValue = sortedMagnitude(round(threshold *
numel(sortedMagnitude)));

    imgDCT(abs(imgDCT) < thresholdValue) = 0;

    % Inverse DCT to get compressed image
    compressedImg = idct2(imgDCT); % Keep as double for accuracy

    % Calculate compression ratio
    originalSize = numel(img);
    compressedSize = sum(abs(imgDCT(:)) > thresholdValue);
    compressionRatio = originalSize / compressedSize;
end
```

LAB-4

```
% Step 1: Prompt the user to select an image file
[filename, pathname] = uigetfile({'*.jpg;*.png;*.tif', 'Image
Files (*.jpg, *.png, *.tif)';},'Select an Image');

% Load the selected image
img = imread(fullfile(pathname, filename));
if size(img, 3) == 1  % If grayscale, replicate channels to
create RGB
    img = repmat(img, 1, 1, 3);
end
original_img = img;
img = rgb2ycbcr(img);  % Convert to YCbCr color space
img = double(img);

% Step 2: Define the standard quantization matrix for luminance
(Y channel)
quant_matrix = [
    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];

% Step 3: JPEG Encoding
[m, n, ~] = size(img);
compressed_img = zeros(m, n, 3);
block_size = 8;
```

```
% Process each channel separately (Y, Cb, Cr)
for channel = 1:3
    for i = 1:block_size:m-block_size+1  % Ensure within bounds
        for j = 1:block_size:n-block_size+1  % Ensure within
bounds
            block = img(i:i+block_size-1, j:j+block_size-1,
channel);  % Get 8x8 block
            dct_block = dct2(block);  % Apply DCT to the block
            % Quantize the DCT coefficients
            quantized_block = round(dct_block ./ quant_matrix);
            % Store quantized block in the compressed image
            compressed_img(i:i+block_size-1, j:j+block_size-1,
channel) = quantized_block;
        end
    end
end

% Step 4: Save only the quantized DCT coefficients (this
simulates compression)
compressed_filename = fullfile(tempdir, 'compressed_data.mat');
% Temporary directory
save(compressed_filename, 'compressed_img', 'quant_matrix');

% Step 5: JPEG Decoding
load(compressed_filename, 'compressed_img', 'quant_matrix');
reconstructed_img = zeros(m, n, 3);

% Process each channel separately (Y, Cb, Cr)
for channel = 1:3
    for i = 1:block_size:m-block_size+1  % Ensure within bounds
```

```
        for j = 1:block_size:n-block_size+1  % Ensure within
bounds
            quantized_block = compressed_img(i:i+block_size-1,
j:j+block_size-1, channel);  % Get 8x8 block
            % Dequantize the DCT coefficients
            dequantized_block = quantized_block .* quant_matrix;
            % Apply inverse DCT
            idct_block = idct2(dequantized_block);
            % Store reconstructed block in the image
            reconstructed_img(i:i+block_size-1, j:j+block_size-
1, channel) = idct_block;
        end
    end
end

% Ensure pixel values are in the range [0, 255] for display
reconstructed_img = uint8(min(max(reconstructed_img, 0), 255));

% Convert back to RGB color space
reconstructed_img = uint8(ycbcr2rgb(reconstructed_img));

% Step 6: Display the Original and Reconstructed Image
subplot(1,2,1);
imshow(original_img);
title('Original Image');

subplot(1,2,2);
imshow(reconstructed_img);
title('Reconstructed Image');

% Step 7: Calculate Metrics (MSE, PSNR, Compression Ratio)
```

```
% Calculate MSE (Mean Squared Error)
mse_value = immse(original_img, reconstructed_img);
fprintf('Mean Squared Error (MSE): %.4f\n', mse_value);

% Calculate PSNR (Peak Signal-to-Noise Ratio)
psnr_value = psnr(reconstructed_img, original_img);
fprintf('Peak Signal-to-Noise Ratio (PSNR): %.2f dB\n',
psnr_value);

% Calculate the original image size (in bytes)
original_size = numel(original_img) * 8 / 1024;  % Size in KB
fprintf('Original Image Size: %.2f KB\n', original_size);

% Calculate the compressed image size (size of quantized
coefficients)
compressed_file_info = dir(compressed_filename);
compressed_size = compressed_file_info.bytes / 1024;  % Size in
KB

fprintf('Compressed Image Size (Actual): %.2f KB\n',
compressed_size);

% Calculate Compression Ratio
compression_ratio = original_size / compressed_size;
fprintf('Compression Ratio: %.2f\n', compression_ratio);

% Calculate Compression Efficiency
efficiency = (1 - (compressed_size / original_size)) * 100;  %
in percentage
fprintf('Compression Efficiency: %.2f%%\n', efficiency);
```

LAB-5 Audio

```
% Audio Compression Example with User Input

% Prompt user for the input audio file
[inputAudioFile, audioPath] = uigetfile('*.wav', 'Select an
audio file');
if isequal(inputAudioFile, 0)
    disp('User canceled the selection.');
    return;
else
    audioFile = fullfile(audioPath, inputAudioFile);
end

% Read the input audio file
[audioData, fs] = audioread(audioFile);

% Prompt the user for a threshold value for frequency reduction
thresholdPercentage = input('Enter threshold percentage for
frequency reduction (e.g., 1 for 1%): ');

% Perform Fourier Transform to convert the signal into the
frequency domain
audioFreq = fft(audioData);

% Set a threshold to remove high-frequency components
threshold = (thresholdPercentage / 100) *
max(abs(audioFreq(:)));  % Use max across all elements for
threshold

% Create the frequency mask: apply the thresholding condition
element-wise
```

```
freqMask = abs(audioFreq) > threshold;

% Apply the mask to the audio frequency components
compressedAudioFreq = audioFreq .* freqMask;

% Apply inverse Fourier Transform to get back to the time domain
compressedAudio = ifft(compressedAudioFreq);

% Save the compressed audio to a new file
[outputAudioFile, audioSavePath] =
uiputfile('compressed_audio.wav', 'Save compressed audio as');
if isequal(outputAudioFile, 0)
    disp('User canceled the file save.');
else
    audiowrite(fullfile(audioSavePath, outputAudioFile),
real(compressedAudio), fs);
    disp(['Compressed audio saved as: ', fullfile(audioSavePath,
outputAudioFile)]);
end

% Play the original and compressed audio
disp('Playing original audio...');
sound(audioData, fs);
pause(length(audioData)/fs + 2);

disp('Playing compressed audio...');
sound(real(compressedAudio), fs);
```

LAB-5 Video

```
% Video Compression Example in MATLAB (without using NumFrames)

% Prompt user for the input video file
[inputVideoFile, videoPath] = uigetfile('*.mp4;*.avi;*.mov',
'Select a video file');
if isequal(inputVideoFile, 0)
    disp('User canceled the selection.');
    return;
else
    videoFile = fullfile(videoPath, inputVideoFile);
end

% Create a VideoReader object to read the input video
videoObj = VideoReader(videoFile);

% Get video properties
originalFrameRate = videoObj.FrameRate;
originalResolution = [videoObj.Width, videoObj.Height];

% Prompt the user for a frame rate reduction factor
frameRateReductionFactor = input('Enter the frame rate reduction
factor (e.g., 2 for half the original frame rate): ');

% Prompt the user for resolution reduction factor
resolutionReductionFactor = input('Enter the resolution
reduction factor (e.g., 2 for half the original resolution): ');

% Calculate the new frame rate and resolution
newFrameRate = originalFrameRate / frameRateReductionFactor;
newResolution = originalResolution / resolutionReductionFactor;
```

```
% Create a VideoWriter object for writing the compressed video
[outputVideoFile, videoSavePath] =
uiputfile('compressed_video.mp4', 'Save compressed video as');
if isequal(outputVideoFile, 0)
    disp('User canceled the file save.');
    return;
else
    compressedVideoFile = fullfile(videoSavePath,
outputVideoFile);
end

% Create a VideoWriter object with new frame rate
outputVideoObj = VideoWriter(compressedVideoFile, 'MPEG-4');
outputVideoObj.FrameRate = newFrameRate;
open(outputVideoObj);

% Read and write frames with reduced resolution and frame rate
frameCount = 0;
while hasFrame(videoObj)
    frame = readFrame(videoObj);    % Read one frame
    frameCount = frameCount + 1;

    % Resize the frame based on the resolution reduction factor
    resizedFrame = imresize(frame, 1 /
resolutionReductionFactor);

    % Write every nth frame, where n is the frame rate reduction
factor
    if mod(frameCount, frameRateReductionFactor) == 0
        writeVideo(outputVideoObj, resizedFrame);  % Write the
frame
```

```
    end
end

% Close the video writer object
close(outputVideoObj);

disp(['Compressed video saved as: ', compressedVideoFile]);
```