PROJECT: *Efficient Compression of 8 bit Grayscale Images of 512x512 Samples*

This project weighs 20% of the course

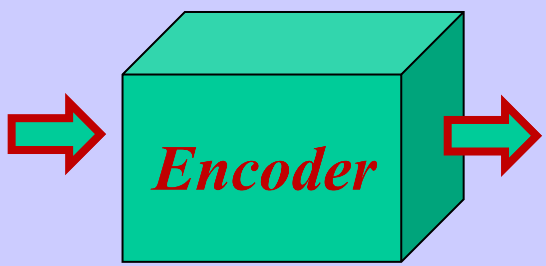
Each team can not exceed 5 students under any circumstances

Your task is to

Develop an efficient algorithm for compression of 8-bit grayscale images of 512×512 samples

You have to implement your encoder in MATLAB
Your algorithms have to be implemented from scratch
You can use any function from MATLAB toolbox,
but are not allowed to use any software parts written by others

Test grayscale image in PGM format and a quality parameter



Bitstream

These images can be read using the "imread / imshow" function of MATLAB and can be written using the "imwrite" function of MATLAB

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

Project submission:

- MATLAB code for encoder and decoder
- Presentation slides (10 slides at maximum)
- Bitstream in 2D matrix for the first X matrix (Mile Stone1 test)
- Check the encoded DC+AC bits with the uploaded samples
- Decoder to test the picture (Mile Stone2)

The presentation have to be provided in Microsoft PowerPoint format

They should represent the project report and should contain:

- Description of algorithm
- Results for test images
- Who did what

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

Milestone 1

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

The goal for your image coding algorithm is
to provide a coding efficiency that is at least comparable to that of a typical JPEG implementation. It is encouraged to develop an algorithm that is better than JPEG

Import Image → *Segmentation* → *Bits Decrementing* → *Applying DCT* → *Quantization*

You are given a gray picture of 512×512 samples
Import the images using MATLAB
Put the pictures in the MATLAB directory, in the documents folders

Six grayscale test images with a size of samples and a bit depth of 8 bit are used for testing your algorithm. They are provided in PGM format and are shown below

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

Segmentation

For the codes to work, a segmentation for the big matrix is required
Develop a code to transform the 2D matrix of 512×512 into 3D matrix of (8×8)×4096
(Search in Google to how you can transfer form 3D matrix to 2D)

Each matrix of the 4096 matrices is called matrix X

Bits Decrementing
Or level shifting

One can decrease the number of information stored in the data matrix X by simply subtracting 128 from each number
NOTE: ***before subtraction you have to transfer the matrix using Double function***

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

Applying DCT

Discrete Cosine Transform (DCT)

The DCT is in a class of mathematical operations that includes the well known Fast Fourier Transform (FFT), as well as many others

The basic purpose of these operations is to take a signal and transform it from one type of representation to another

Since an image is a two dimensional signal that is perceived by the human visual system, then

The DCT is used to convert the signal (spatial information) into numeric data ("frequency" or "spectral" information) so that the image's information can exist in a quantitative form that can be manipulated for compression

It is the key to the JPEG baseline compression process

It extracts spatial frequency information from the spatial amplitude samples

These frequency components are then quantized to eliminate the visual data from the image that is least perceptually apparent, thereby reducing the amount of information that must be stored

Finally, the redundant properties of the quantized frequency samples are exploited through Huffman coding to produce the compressed representation

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

Form 8×8 matrix A consists of elements a_{ij}, with i,j =0,………...,7, given by

$$a_{ij} = 1/\sqrt{8} \qquad i = 0, \ \ all \ j$$

$$a_{ij} = \frac{1}{4}\cos\left(\frac{\pi(2j+1)}{16} * i\right) \quad i > 0, \ \ all \ j$$

Hence, for any 8×8 matrix X of the segmented matrices that contains the data for the image, and the generated DCT matrix A, a matrix X can be transformed to a matrix of one DC value stored in position (1,1) and all of the other data are AC data using the following transformation

$$Transformed \ data \ of \ \ matrix \ \ X = DCT\,matrix \ * X * \ DCT\,matrix^{T}$$

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

Quantization

*Divide the final transformed matrix from the previous step by a **constant number** for quantization*
Increasing this number will increase the compression ratio,
but it will decrease the Picture SNR (PSNR)
NOTE: *After quantization you have to use **Round** function to cancel any decimal follows **0/1***

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

Encode DC

Encode AC

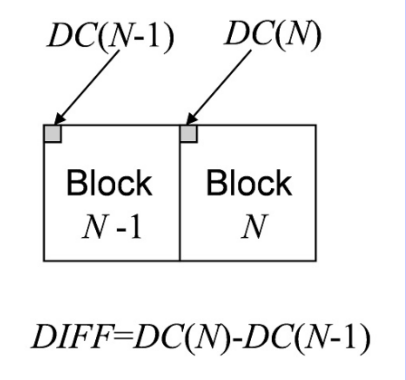
We previously transform the 2D matrix , of 512×512 into 3D matrix of 8×8×4096, i.e., it is divided into N block (N =4096), each block is 2D matrix of size 8x8

Steps of DC Encoding

1-Calculate the difference between the DC value of each block and the successive one

Ex: Consider we have two consequent matrices, the first with DC value 14, and the second of 0

$$DC_{\text{difference}} = DC_{\text{current}} - DC_{\text{previous}} = 14-0=14$$



PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

2-This difference DC is encoded into two different parts, the first one is the code of category and the second is the code of the difference

Given in the table to encode the category of different DC difference values

DC CODEBOOK

Category C	Range of *DIFF* value	Example codeword
0	0	00
1	-1, 1	010
2	-3, -2, 2, 3	011
3	-7..-4, 4..7	100
4	-15..-8, 8..15	101
5	-31..-16, 16..31	110
6	-63..-32, 32..63	1110
7	-127..-64, 64..127	11110
8	-255..-128, 128..255	111110
9	-511..-256, 256..511	1111110
10	-1023..-512, 512..1023	11111110
11	-2047..-1024, 1024..2047	111111110

It is clear that the DC difference 14 has category 4 which is encoded in binary form 101 using Variable Length Code (VLC) DC codebook

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

The difference is +14, then its binary code should be determined using JPEG code which is '1110'

Why? As the category number is 4, the difference representation has to be done in 4 bits. 4 bits can tolerate numbers from '0000' to '1111', where '0000' represents a difference of the maximum –ve of -15, and '1111' maximum difference of +15

[redacted]

The Final code is '1011110'

Category code +	Difference code
101	1110

Ex: For the second DC

$DC_{difference} = DC_{current} - DC_{previous} = 35-14=21$

It is clear that the DC difference 21 has category 5 which is encoded in binary form 110 using Variable Length Code (VLC) DC code book

The difference is +21, then its binary code should be determined using JPEG code which is '10101'

The final code is 11010101

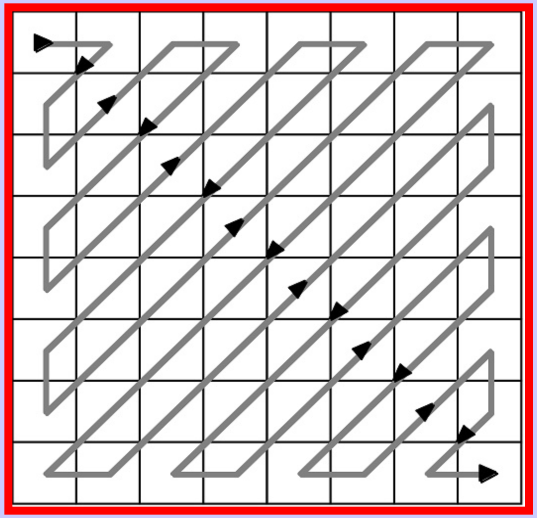
110	10101

0	0	0	0	-15
0	0	0	1	-14
0	0	1	0	-13
0	0	1	1	-12
0	1	0	0	-11
0	1	0	1	-10
0		1	0	-9
0	1	1	1	-8
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

Encode AC

Convert each block (8x8 matrix) into sequence (vector of length 64) using a zig zag scan The elements of that vector are starting by zero (which corresponds to DC value) then each value in the matrix is depicted in the vector followed by number of successive number zeros in the zig zag route



185	3	1	0	-3	-1	0	0
1	0	-1	0	-2	0	0	0
0	0	0	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

*For the first given matrix of 8*8 samples, once applying the code fulfilling the transformation using zig zag route we will get a row vector V with length 64*

V = [0 3 0 1 2 1 1 -1 6 -3 0 -1 0 -2 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

The first number is DC value, so it is ignored and equaled to zero
The 3 has no zeros after it, so it is [0 3 0], then the 1 after the 3 has 2 zeros, then the matrix will be [0 3 0 1 2], the next 1 has 1 zero after it so, the matrix will be [0 3 0 1 2 1 1], and the -1 has 6 zeros after it, so…..

Trim the extra zeros

V = [0 3 0 1 2 1 1 -1 6 -3 0 -1 0 -2 0 -1]

Reshape or convert the trimmed vector into a matrix of 2 rows, where the first row has the elements of the odd positions (number of zeros or run), and the second has the elements of the even positions (level value)

0	0	2	1	6	0	0	0
3	1	1	-1	-3	-1	-2	-1

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

After that conversion, we can see that we have pairs of numbers (run, level), which will help us in encoding the AC

(0,3), (0,1), (2,1), (1,-1), (6,-3), (0,-1), (0,-2), (0,-1)

Convert the pairs (run , level) (0,3), (0,1), (2,1), (1,-1), (6,-3), (0,-1), (0,-2), (0,-1) into (run, category) as follows:

The 3 can be represented in 2 bits, so it is category 2, hence (0,3) is converted to (0,2) followed by the level alone, i.e., (0,2) 3

The sequence can be then take the form

(0,2) 3 (0,1) 1 (2,1) 1 (1 , 1) -1 (6,2) -3 (0,1) -1 (0,2) -2 (0,1) -1

This is encoded using the same as we encode the DC difference

This is encoded using the AC codebook, its input is the zeros and category , its output is the category code, its code is 01

0	0	-3
0	1	-2
1	0	2
1	1	3

PROJECT: *Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples*

DC CODEBOOK

run/category	codeword	run/category	codeword
EOB	1010	2/1	11100
0/1	00	2/2	11111001
0/2	01	[illegible]/3	1111110111
0/3	100	2/4	111111110100
0/4	1011	2/5	1111111110001001
0/5	11010	2/6	1111111110001010
0/6	1111000	2/7	1111111110001011
0/7	11111000	2/8	1111111110001100
0/8	1111110110	2/9	1111111110001101
0/9	1111111110000010	2/10	1111111110001110
0/10	[illegible]	[illegible]	[illegible]0
1/1	[illegible]	[illegible]	[illegible]0111
1/2	[illegible]	[illegible]	[illegible]1110101
1/3	1111001	3/4	1111111110001111
1/4	[illegible]	[illegible]	[illegible]1111110010000
1/5	[illegible]	[illegible]	[illegible]1111110010001
1/6	1111111110000100	3/7	1111111110010010
1/7	1111111110000101	3/8	1111111110010011
1/8	1111111110000110	3/9	1111111110010100
1/9	1111111110000111	3/10	1111111110010101
1/10	1111111110001000	...	...

Check the coder output DC+AC for the first (first block) and get mark for the first and second milestones

26th ,28th of November in lecture time

The sequence can be then encoded as

01 [11] 00 [1] 11100 [1] 1100 [0] 1111111110110 [00] 00 [0] 01 [01] 00 [0]

Then add the end of block code 1010

01 [11] 00 [1] 11100 [1] 1100 [0] 1111111110110 [00] 00 [0] 01 [01] 00 [0]1010

PROJECT: Efficient Compression of 8 Bit Grayscale Images of 512x512 Samples

Milestone 3

Implement the decoder and apply it for your encoder get the picture

Check the picture quality and the PSNR and get the last mark

3rd , 5th of December in lecture time