

Alexandria University

Faculty of Engineering

Communication Department

**Assignment: Lab 2**

**Digital Signal Processing**

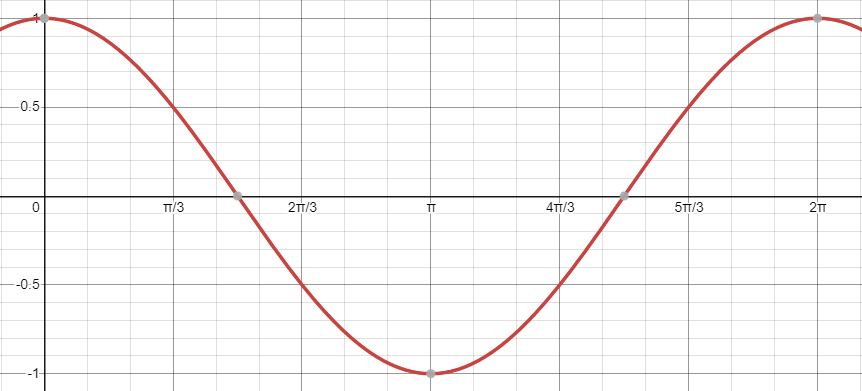
By:

|  |  |
| --- | --- |
| 18010198 | احمد محمد احمد عبدالجواد |
| 18010200 | احمد محمد احمد ابوجبل |
| 18010299 | آسر محمد عاطف زايد |
| 18011084 | علي محمد طعيمة حمص |
| 18011505 | محمد طارق محمد عامر |

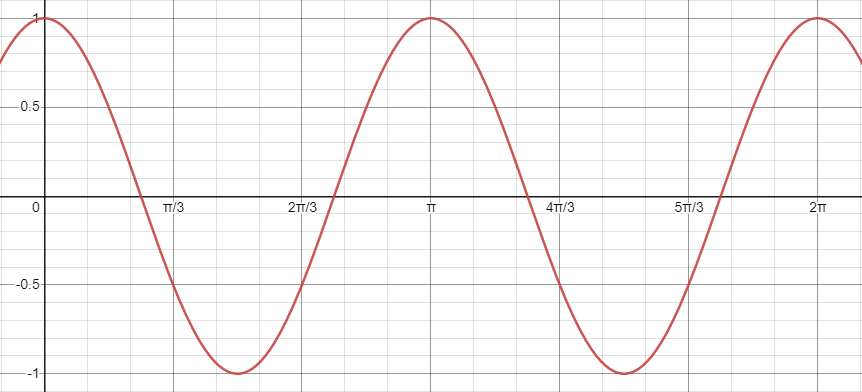
3rd year 1st semester

**Introduction to DCT**

First we know that standard cos looks like this between -1 to 1



If we see another cos with higher frequency, it will look like that



If we took the average of the 2 cos waves above it will look like this



What are we trying to be to represent our image in terms of summation of a lot of cos wave.

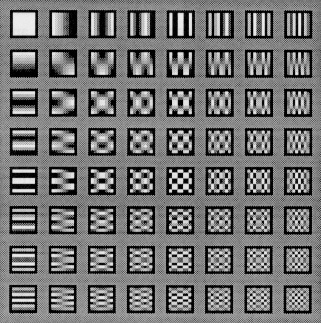
We want to remove the high frequencies of the image without distorting it.

If we have each signal with 8 components that means we need to have 8 cosines to represent it, and in JPEG we use 8x8 blocks of the image at a time.

So each 8x8 block can be represented by 8x8 cosines waves.

Now we need to calculate the DCT basis matrix that show the 8x8 cosines blocks while increasing its frequency

This will look like this



As we can see as we go down or right the frequencies of the block increase.

**MATLAB function to calculate C8:**

This is a function which takes the block size and gets CN

function CN = compute\_CN(N)

if N>1

CN = zeros(N,N);

CN(1,:) = sqrt(1/N);

for i=2:N

for j=1:N

CN(i,j) = sqrt(2/N) \* cos(pi\*(i-1)\*(j-1+0.5)/N);

end

end

elseif N == 1

CN = (1);

else

error('unvaild size, size must be 1 or more');

end

end

We can call like this

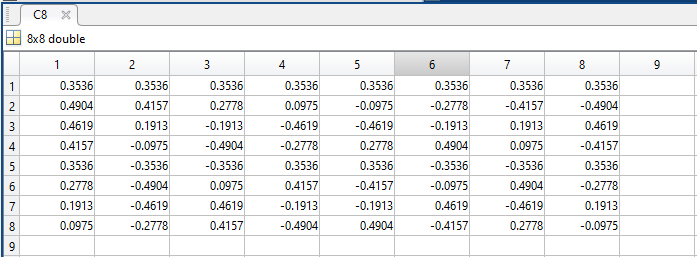
C8 = compute\_CN(8);

%check\_transpose = transpose(C8);

%check\_inv = inv(C8);

save C8.mat C8;

**Output:**



**JPEG encoding and decoding**

First we need to pad the image which doesn’t have number of rows or columns divisible by block size which is 8.

function padded\_image = padding\_by\_block\_size(gray\_image,block\_size)

[rows, columns] = size(gray\_image);

%%%%%%%%%%%%%%%

%%% padding %%%

%%%%%%%%%%%%%%%

% start from first row and skip blocksize pixels till reaching

% the end of all row

for row=1 :block\_size:rows

% start from first column and skip blocksize pixels till

% reaching the end of all columns

for column=1 :block\_size:columns

% at the end of the loops they will store the right size of the

% padded image

row\_end = row + block\_size -1;

column\_end = column + block\_size -1;

% if the right size excceds the actual size, pad the rest with

% 0s

if row\_end > rows

gray\_image(rows+1:row\_end,:) = 0;

end

if column\_end > columns

gray\_image(:,columns+1:column\_end) = 0;

end

end

end

padded\_image = gray\_image;

end

Then we need to split the image to blocks of block size (8)

function blocked\_image = spliiting\_image(image,block\_size)

[rows, columns] = size(image);

% claculate the row and the columns if each block is 8x8

number\_of\_blocks\_horzintally = ceil(columns/block\_size); %220/8=27.5=28

number\_of\_blocks\_vertically = ceil(rows/block\_size);

%%%%%%%%%%%%%%%%%%%%%%%

%%% splitting image %%%

%%%%%%%%%%%%%%%%%%%%%%%

blocked\_image = zeros(block\_size,block\_size,...

number\_of\_blocks\_horzintally,number\_of\_blocks\_vertically);

for row=1:number\_of\_blocks\_horzintally

for column=1:number\_of\_blocks\_vertically

blocked\_image(:,:,column,row) = ...

image(row+(row-1)\*block\_size-(row-1):...

row+(row-1)\*block\_size-(row-1)+block\_size-1,...

column+(column-1)\*block\_size-(column-1):...

column+(column-1)\*block\_size-(column-1)+block\_size-1);

end

end

end

**Encoding:**

Now we need to take each 8x8 block and subtract 128 from it because cosines waves around 0 accesses while image is ranged from 0 to 255 which centers at 128.

Then we will take the DCT of the block and get the quantized block.

**DCT function**

function DCT\_image = DCT\_block(splitted\_image,block\_size)

if block\_size == 8

load C8.mat C8

end

% the values are from range 0-255 which is center at 128

% we want them to be center at 0 so we will subtract 128 from all

% elements

shifted\_image = int16(splitted\_image) -128;

DCT\_image = dct2(shifted\_image);

end

And the output of the DCT represents the weight or the amount of C8 that has effect on the image.

Now we just need to remove the high frequencies of the image and this process called quantization which represented by a table known at the transmitter and the receiver.

Quantization table is just dividing the value of each pixel to the corresponding one in the quantization table and round them to the nearest integer.

function Q = quantization(splitted\_image,r)

%load the quantization matrix and

%round it off

quantization\_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];

T = r.\*quantization\_matrix;

Q=round(splitted\_image./T);

end

r represent how much of contribution each block will effect to our image in the quantization process.

The greater the r the high the contribution of removing the high frequencies blocks.

**Decoding:**

We first dequantize each block with same quantization table and r factor

function Q = dequantization(quantized\_image,r)

%load the quantization matrix and

%round it off

quantization\_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];

T = r.\*quantization\_matrix;

Q= quantized\_image.\*T;

end

Then we take the IDCT with adding 128 to regain the image in the correct range

function inv\_DCT\_image = inv\_DCT\_block(splitted\_image,block\_size)

if block\_size == 8

load C8.mat C8

end

%inv\_DCT\_image = (transpose(C8)\*splitted\_image)\*C8 +128;

% we re-add 128 to return to the normal range of image which is from

% 0-255

inv\_DCT\_image = idct2(splitted\_image) + 128;

end

We just need to reconstruct blocks again to form our image

function image = reconstruct(image\_4d,block\_size)

[temp,temp2,columns,rows] = size(image\_4d);

image = zeros(rows\*block\_size,columns\*block\_size);

for row=1:rows

for column=1:columns

image(row+(row-1)\*block\_size-(row-1):...

row+(row-1)\*block\_size-(row-1)+block\_size-1,...

column+(column-1)\*block\_size-(column-1):...

column+(column-1)\*block\_size-(column-1)+block\_size-1) = ...

image\_4d(:,:,column,row);

end

end

end

Let’s take an example on an image

% read the image

original\_image = imread('Lenna.png');

% get number of rows, columns of the image

% in addition too number of channels to if it's color or gray

[rows, columns, number\_of\_color\_channels] = size(original\_image);

pixels = rows\*columns;

% check if the image is color or gray

% if it's color convert it to gray

% if it's gray we are ready

if number\_of\_color\_channels == 3

gray\_image = rgb2gray(original\_image);

else

gray\_image = original\_image;

end

r = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Encoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

block\_size = 8;

%%%%%%%%%%%%%%%

%%% padding %%%

%%%%%%%%%%%%%%%

padded\_image = padding\_by\_block\_size(gray\_image,block\_size);

%%%%%%%%%%%%%%%%%%%%%%%

%%% splitting image %%%

%%%%%%%%%%%%%%%%%%%%%%%

splitted\_image = spliiting\_image(padded\_image,block\_size);

% claculate the row and the columns if each block is 8x8

number\_of\_blocks\_horzintally = ceil(columns/block\_size); %220/8=27.5=28

number\_of\_blocks\_vertically = ceil(rows/block\_size);

% temporery spltted image to store the DCT of the image

DCT\_image = zeros(block\_size,block\_size,number\_of\_blocks\_vertically,number\_of\_blocks\_horzintally);

% temporery spltted image to store the quantization of the image after

% takin its DCT

quantized\_image = zeros(block\_size,block\_size,number\_of\_blocks\_vertically,number\_of\_blocks\_horzintally);

for row=1:number\_of\_blocks\_horzintally

for column=1:number\_of\_blocks\_vertically

%%%%%%%%%%%%%%%%%

%%% DCT Block %%%

%%%%%%%%%%%%%%%%%

DCT\_image(:,:,column,row) = DCT\_block(splitted\_image(:,:,column,row),block\_size);

%%%%%%%%%%%%%%%%%%%%

%%% quantization %%%

%%%%%%%%%%%%%%%%%%%%

quantized\_image(:,:,column,row) = quantization(DCT\_image(:,:,column,row),r);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% decoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% temporery spltted image to store the dequantization of the image

dequantized\_image = zeros(block\_size,block\_size,number\_of\_blocks\_vertically,number\_of\_blocks\_horzintally);

% temporery spltted image to store the IDCT of the image after getting

% its dequantization

inv\_DCT\_image = zeros(block\_size,block\_size,number\_of\_blocks\_vertically,number\_of\_blocks\_horzintally);

for row=1:number\_of\_blocks\_horzintally

for column=1:number\_of\_blocks\_vertically

%%%%%%%%%%%%%%%%%%%%%%

%%% dequantization %%%

%%%%%%%%%%%%%%%%%%%%%%

dequantized\_image(:,:,column,row) = dequantization(quantized\_image(:,:,column,row),r);

%%%%%%%%%%%%%%%%%%%%%%%%%

%%% inverse DCT Block %%%

%%%%%%%%%%%%%%%%%%%%%%%%%

inv\_DCT\_image(:,:,column,row) = inv\_DCT\_block(dequantized\_image(:,:,column,row),block\_size);

end

end

decoded\_image = uint8(reconstruct(uint8(inv\_DCT\_image),block\_size));

subplot(1,2,1)

imshow(original\_image)

title('Original Image')

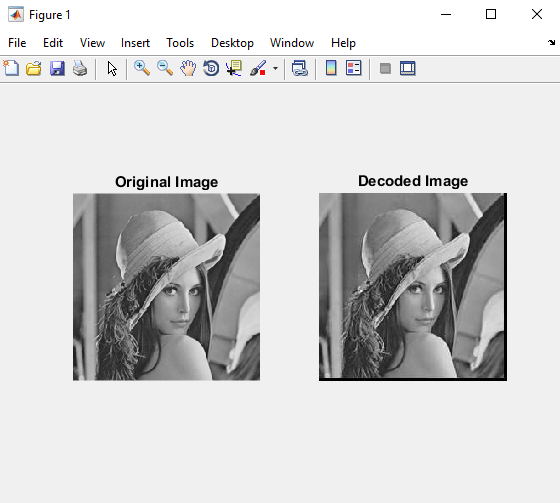
subplot(1,2,2)

imshow(decoded\_image)

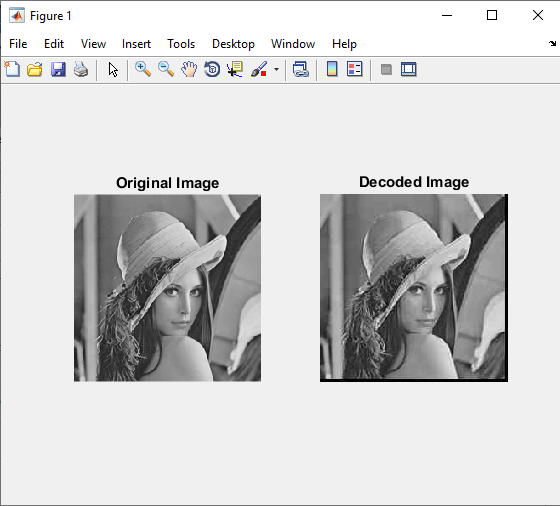
title('Decoded Image')

Output:

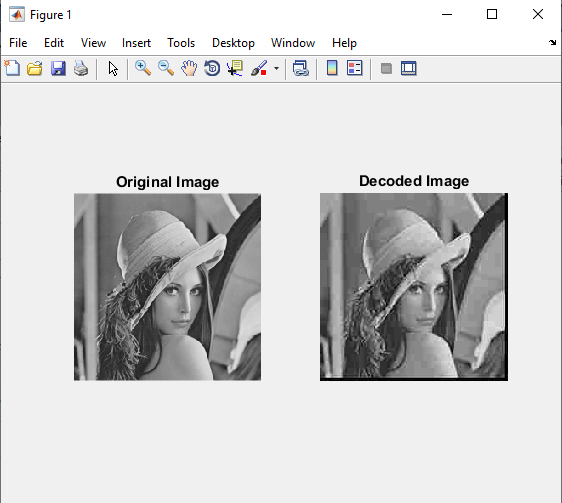
At r=1



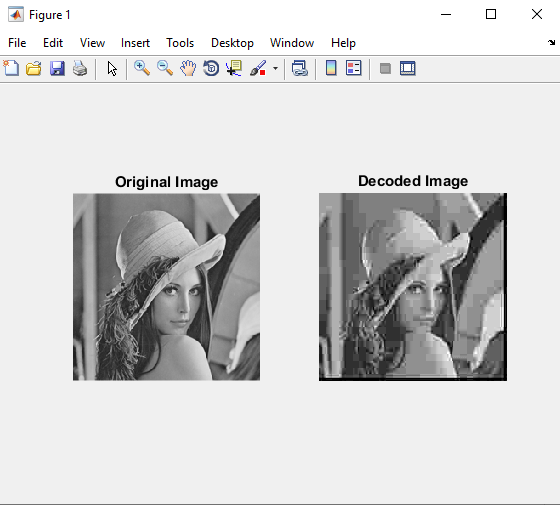
At r=2



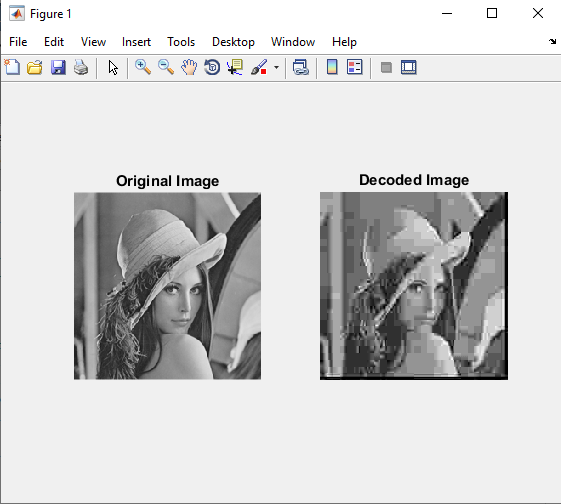
At r=4



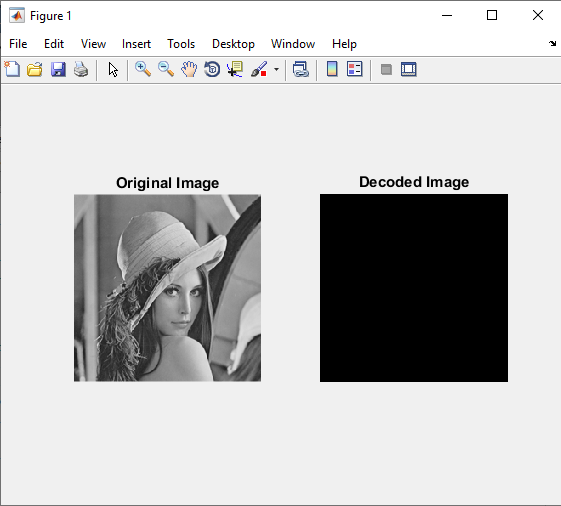
At r=8



At r=10



At r=0



As we can see by increasing r we remove more high frequencies part og the image as r affects the quantization table.

So the smaller r is the exact image I get but if we reach 0 we make the quantization 0s so the image gets black and I can’t redo that.