جامعة الإسكندرية
ALEXANDRIA UNIVERSITY

EEC-371 Digital Signal Processing

Lab3 Report

(DCT transform) ECE LEVEL 3

Team Members:

Ali Emad Abdel-Latif	19016028
Mayar Mohamed Mahmoud El-Prince	19016749
Noha Mohamed Mahmoud El-Nemr	19016798
Hythem Ahmed Shaaban	19016856
Yara Mohamed Nasser Hassan	19016873

Implementation of the DCT transform with application to the JPEG transform

The Process:

The following is a general overview of the JPEG process. Later, we will take the reader through a detailed tour of JPEG's method so that a more comprehensive understanding of the process may be acquired.

1. The image is broken into 8x8 blocks of pixels.

2. Working from left to right, top to bottom, the DCT is applied to each block.

3. Each block is compressed through quantization.

4. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space.

5. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).

1. DCT transform in 2-D

```
N = 8;

C_8 = zeros(8);
for k=0 : N-1
    for r=0 : N-1
       if k>0
           uk = sqrt(2/N);
       else
           uk = sqrt(1/N);
```

```
        end
        C_8(k+1,r+1)=
uk*cos((pi/N)*k*(r+0.5));
    end
end
```

2. JPEG encoding

2.1.Block divide

```
%Block divide
img = imread('The World
Champion.jpg');    %Reading image

info_input = dir('The World
Champion.jpg'); % get information
about the image file
input_img_size = info_input.bytes %
get the file size in bytes


if size(img, 3) == 3
    grayImg = rgb2gray(img);
else
    grayImg = img;
end


figure; imshow(grayImg); title('read
image');

[rows, cols] = size(grayImg);
%Get number of rows and columns of the
image
```

```
paddedRows = N*ceil(rows/N);
%Number of rows divisible by 8
paddedCols = N*ceil(cols/N);
%Number of columns divisible by 8

paddedImg=zeros(paddedRows,
paddedCols);
paddedImg(1:rows, 1:cols)= grayImg;
%Divisible by 8 image with zero
padding

figure; imshow(uint8(paddedImg));
title('padded image');
% ---->>>>>reshape fn needs low level
implementation <<<<<---------
block8by8 = split_image(paddedImg,[N
N]);  %The 8x8 blocks of the image
%block8by8(5,5,1450)
%An example for the block 1450 fifth
row fifth column

[x ,y
,numberOfBlocks]=size(block8by8);
DctOfTheBlock
=zeros(N,N,numberOfBlocks); %could be
discarded and use the varaible
"block8by8" directly
```

2.2. DCT block

```
%loop on all blocks to apply DCT with
C8 on them
for i=1:numberOfBlocks
```

```
    DctOfTheBlock(:,:,i)=
C_8*block8by8(:,:,i)*transpose(C_8);
%A^=CN*A*CN(transpose)
end
```

Quantization:

Our 8x8 block of DCT coefficients is now ready for compression by quantization. A remarkable and highly useful feature of the JPEG process is that in this step, varying levels of image compression and quality are obtainable through selection of specific quantization matrices. This enables the user to decide on quality levels ranging from 1 to 100, where 1 gives the poorest image quality and highest compression, while 100 gives the best quality and lowest compression.

q_mtx is a quantization matrix with a quality level of 50, this matrix renders both high compression and excellent decompressed image quality.

2.3. Quantization

```
%Quantization
%A Standard Quantization Matrix for
50% quality
r=input('Enter Scaling factor  ');
%used  to change quantization level 1
- 100
q_mtx = [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];
```

```
q =
rescale(q_mtx,r,numberOfBlocks,DctOfTh
eBlock);
```

3. JPEG decoding

3.1.Rescaling the data block

```
%Rescaling
R =
rescaling(q,r,q_mtx,numberOfBlocks);
```

3.2. DCT block

```
%IDCT
IDCT_block =
zeros(8,8,numberOfBlocks);
for i=1:numberOfBlocks

IDCT_block(:,:,i)=transpose(C_8)*R(:,:
,i)*C_8; %A=CN(transpose)*A**CN
end
```

3.3. Merging the blocks

```
%Merging
newImage = merge(IDCT_block,
paddedRows, paddedCols);
%disp(newImage);

imwrite(newImage,
'compressedImg.jpg'); % save the image
to a JPEG file
info = dir('compressedImg.jpg'); % get
information about the image file
```

```
comp_img_size = info.bytes % get the
file size in bytes

figure;
imshow(uint8(newImage));
title('output image');
```

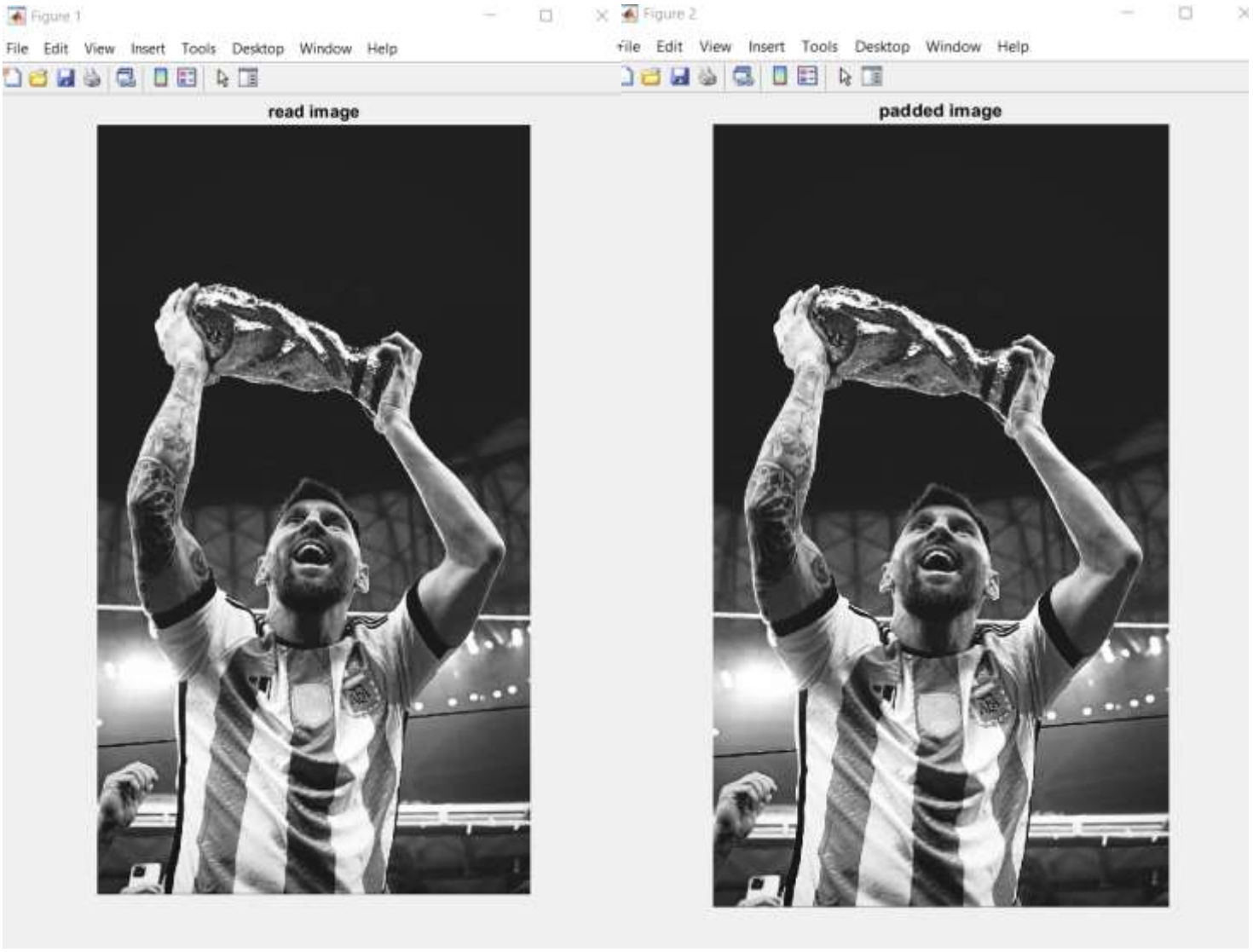
functions:

```
function quantized_dct = rescale(x,y,n,dct)
    T=y*x;
    quantized_dct = zeros(8,8,n);
    for k=1:n
        quantized_dct(:,:,k)= round(dct(:,:,k) ./ T);
    end
    %disp(quantized_dct);
end

function rescaled_block8by8 = rescaling(y,r,q,n)
    T=r*q;
    rescaled_block8by8 = zeros(8,8,n);
    for k=1:n
    rescaled_block8by8(:,:,k) = y(:,:,k) .* T;
    end
    %disp(rescaled_block8by8);
end

function blocks = split_image(I, block_size)
%splits the image matrix I into small blocks of size
BLOCK_SIZE, and returns the blocks as a 3D matrix.

    [nrows, ncols] = size(I);       % Get the number
of rows and columns in the image

    % Get the number of blocks in each direction
    nblocks_row = ceil(nrows/block_size(1));
    nblocks_col = ceil(ncols/block_size(2));

    % Initialize the output matrix
    blocks = zeros(block_size(1), block_size(2),
nblocks_row*nblocks_col);

    % Split the image into blocks
```

```
    for i = 1:nblocks_row
        for j = 1:nblocks_col
            % Get the current block
            block = I((i-
1)*block_size(1)+1:i*block_size(1), (j-
1)*block_size(2)+1:j*block_size(2));

            % Store the block in the output matrix
            blocks(:,:,(i-1)*nblocks_col+j) = block;
        end
    end
    %disp(blocks);
end
function newImage = merge(IDCT_block, paddedRows,
paddedCols)
    newImage = zeros(paddedRows, paddedCols);
    row_blocks = paddedRows/8;
    col_blocks = paddedCols/8;
    for k=1:row_blocks
        for j=1 : col_blocks
            rmin=(k-1)*8+1;
            rmax=k*8;
            cmin=(j-1)*8+1;
            cmax=j*8;
            newImage(rmin:rmax,cmin:cmax)=
IDCT_block(:,:,(k-1)*col_blocks+j);
        end
    end
end
```

we will notice that when r increases , the quality of the picture decreases:

original image:



r=1:

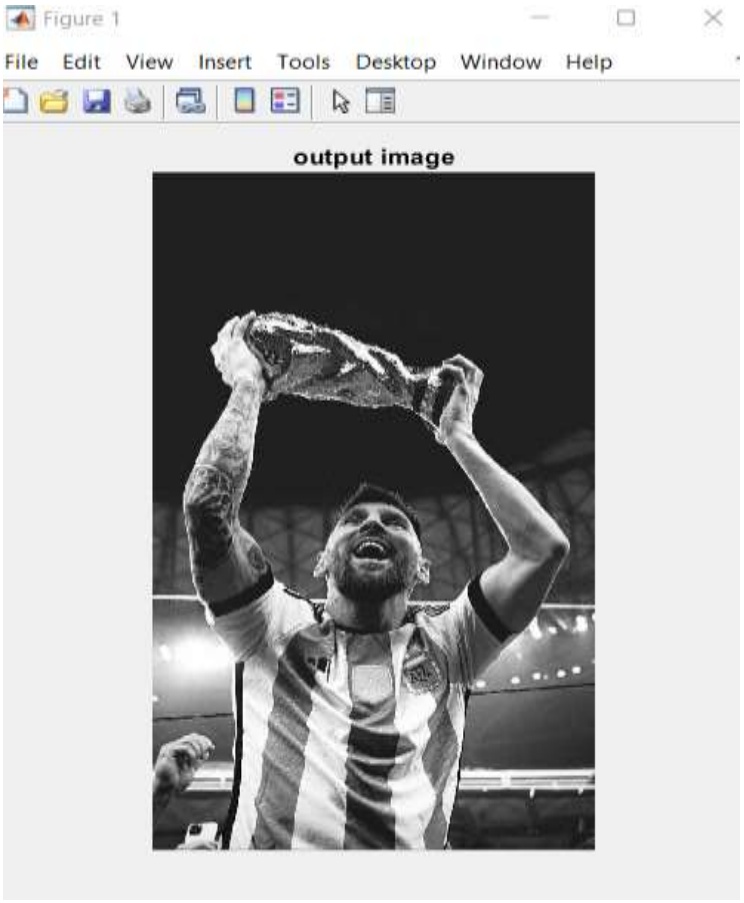
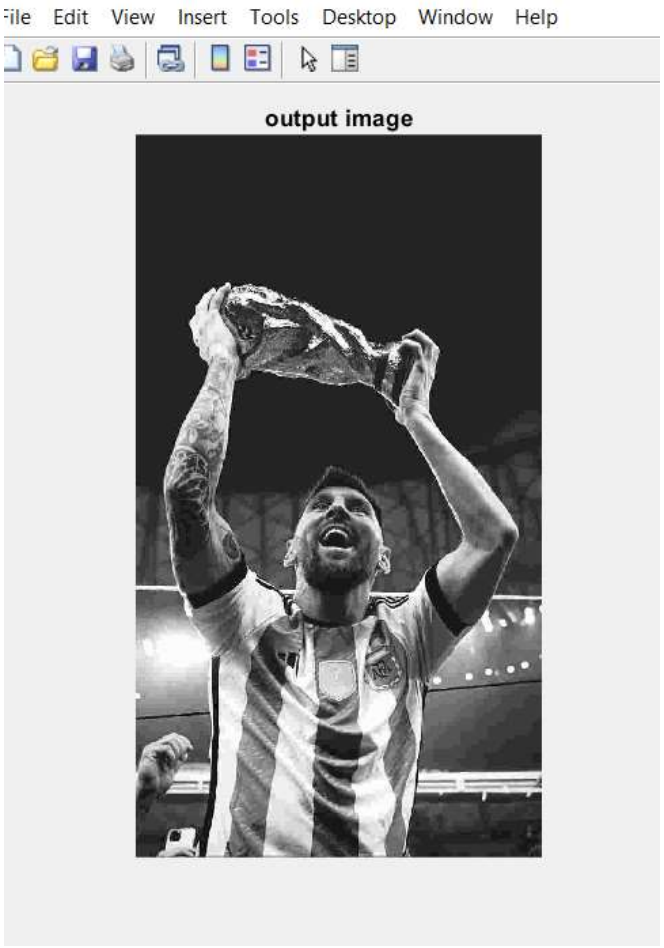
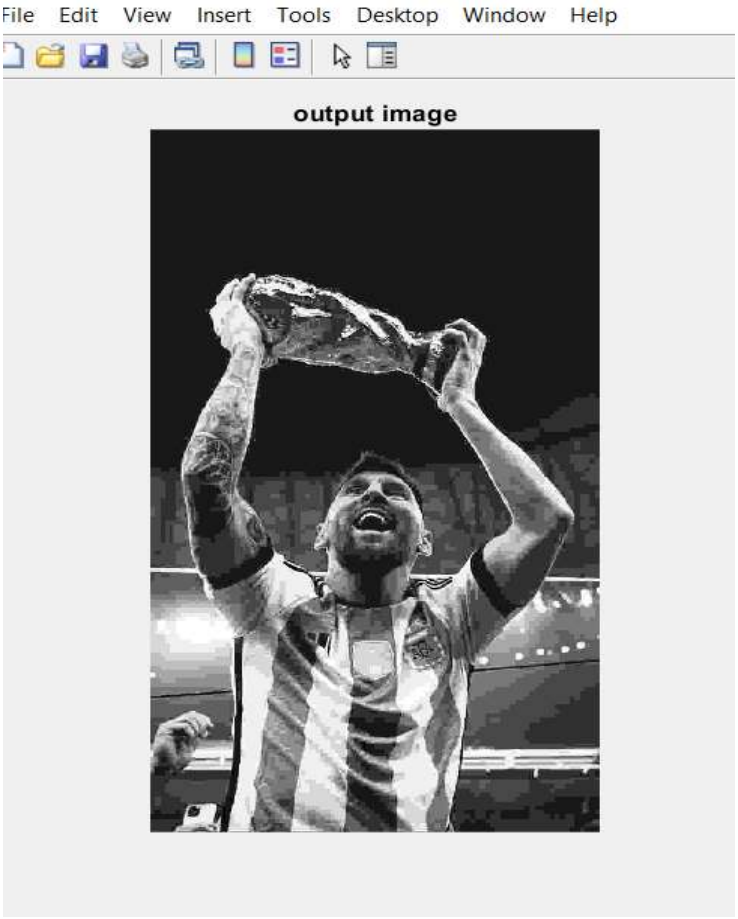


Figure 1
File Edit View Insert Tools Desktop Window Help
output image

r=6:


File Edit View Insert Tools Desktop Window Help
output image

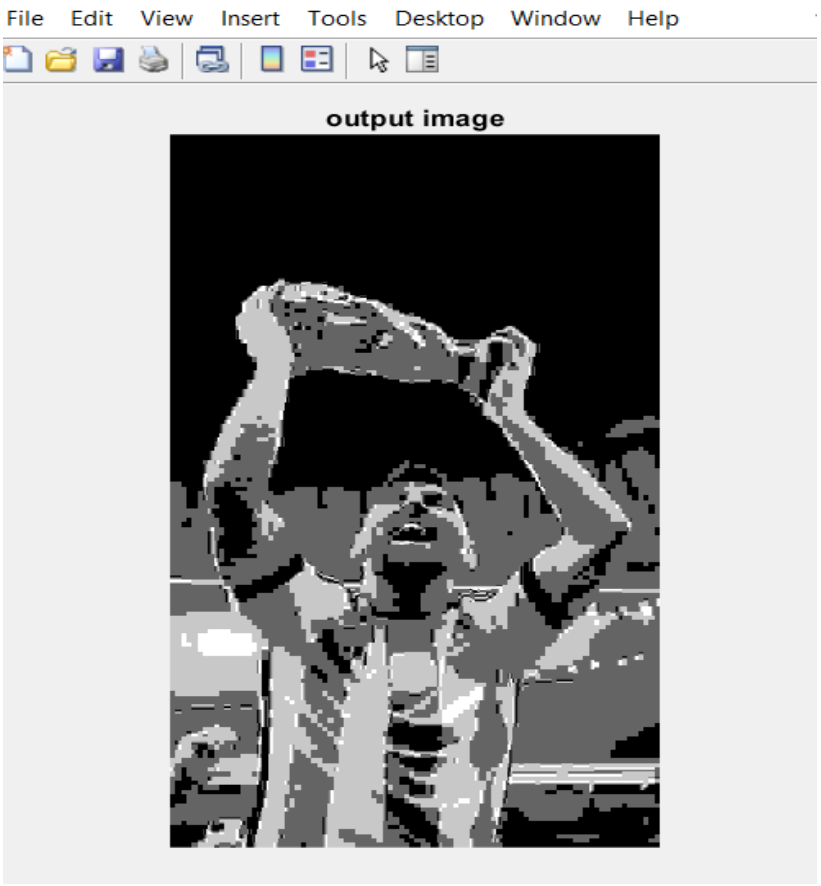
r=12:


File Edit View Insert Tools Desktop Window Help
output image

r=25:


File Edit View Insert Tools Desktop Window Help
output image

r=50:


File Edit View Insert Tools Desktop Window Help
output image

r=100:


ile Edit View Insert Tools Desktop Window Help
output image