**JPEG transform project**

**Graphical user interface, application

Description automatically generated**

|  |  |  |
| --- | --- | --- |
| **اسم الطالب (رباعي)** | **الرقم الجامعي** | **section** |
| 1.أحمد هاني أنور محمود | 18010265 | 3 |
| 2.محمد عبد السلام عبد المطلب قطب | 18011535 | 3 |
| 3.مؤمن صالح عبد الرحمن أحمد | 18011902 | 3 |
| 4.مؤمن أشرف محمد علي | 18011901 | 3 |

**MAIN FILE**

**Global variable**

image = imread('cat.jpg');

%image =imread('D:\Python\Photo\Ahmed.JPG');

%image =imread('prefecto\prefecto.jpeg');%% 8

%image =imread('cat\cat.jpeg');

saveNameBefore ='Before.jpeg';

saveNameAfter = 'After.jpeg';

scaling=3;

**1.Find C8**

C8=findDCTMatrix(); %call function

inve=inv(C8);

transpo=C8';

transpo-inve; %%Check if inverse == transpose (10^-15 ~=0)

**2. JPEG encoding**

grayImage = rgb2gray(image);

imwrite(grayImage,saveNameBefore );

grayImage = pading(grayImage);

**2.1.Block divide**

mySplit = SplitImage(grayImage); %call function

**2.2. DCT block**

blocksDCT = DCTBlock(mySplit,C8,0); %call function

**2.3. Quantization**

load 'DCTQ'

JPEGRes = QuantJPEG(blocksDCT,DCTQ,scaling); %call function

**3. JPEG decoding**

**3.1.Rescaling the data blocks**

rescaleIM=rescaling(JPEGRes,DCTQ,scaling);%call function

**3.2. DCT block inverse**

blocksIDCT = DCTBlock(rescaleIM,C8,1); %call function

**3.3. Merging the blocks**

JPEGImage =recombinesBlocks(blocksIDCT);%call function

**4. Save the compressed image**

imwrite(JPEGImage, saveNameAfter);

imshow(image)

title('Color image before compression','FontSize',16,'color','red')

figure;

imshow(grayImage)

title('gray image before compression','FontSize',16,'color','blue')

figure;

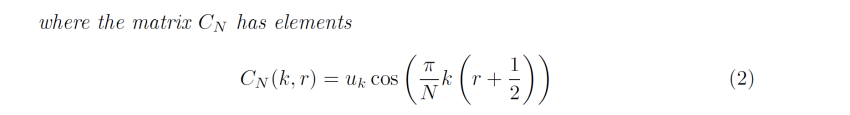
imshow(JPEGImage)

title('compression image','FontSize',16,'color','green')

**NOTE:**

Our (main file) contain call of functions that will be discussed below step by step :

1. **find DCTMatrix**



Construct ( C8 ) where :

 && 

function C8 = findDCTMatrix()

r=[0:7];

K=[1:7]';

u0=sqrt(1/8);

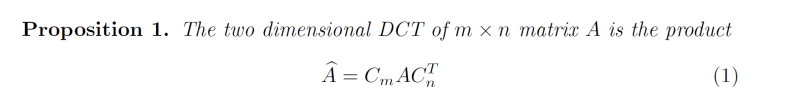
C0=[u0 u0 u0 u0 u0 u0 u0 u0]; %this in case of k=0

C7 = sqrt(2/8).\*cos((pi/8)\*(K\*(r+.5)));

C8 = [C0;C7];

end

1. **DCT Block**



Here :

Parameter to choose if you want dct() or idct():

-If parameter =0 then blockDCT contain the dct matrix of each block.

-But if parameter ~=1 then blockDCT contain the inverse dct matrix of each block.

function blockDCT = DCTBlock(splits,C8,paramter)

[l m row col]=size(splits);

if paramter~=0

C8=C8';

end

for i=1:row

for j=1:col

subIm=double(splits(:,:,i,j));

blockDCT(:,:,i,j) =C8\*subIm\*C8';

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

% if paramter==0

% blockDCT(:,:,i,j) =C8\*subIm\*C8';

% else

% blockDCT(:,:,i,j) =C8'\*subIm\*C8;

% end

end

end

end

1. **Split Image**

Here we split image into blocks of Size (8\*8):

Result is a 4-D matrix 8x8xx:

So result(:,:,i,j) this indicates to the ith &jth block, which size id 8x8:

function result = SplitImage(grayImage)

blockSize=8;

[Row Col]=size(grayImage);

for i =1:Row/blockSize

for j=1:Col/blockSize

result(:,:,i,j) = grayImage( (((i-1)\*blockSize)+1) :(i\*blockSize) ,(((j-1)\*blockSize)+1):(j\*blockSize) );

end

end

end

1. **Pad zeros**

If image size is not divisibe by (8) then pad rows and columns by zeros until it's divisible.

function padGray = pading(grayIm)

[row colum]=size(grayIm);

padrow=0;

padclo=0;

if(mod(row,8))

num=floor(row/8)+1;

padrow=num\*8-row;

end

if(mod(colum,8))

num=floor(colum/8)+1;

padclo=num\*8-colum;

end

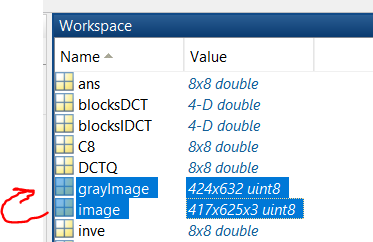
if((mod(colum,8))&(mod(row,8)))

padGray= padarray(grayIm,[padrow padclo],0,'post');

else

padGray=grayIm;

* And here how it works if image is not divisible by 8

so here our image size before padding =417x625.

417 &625 don’t accept divide by 8 so we pad it until the first big number which accept divide by 8,

So, our image size after padding =424x632

424/8= 53 &632/8=79.

1. **Quantization JPEG**

Here we will multiply DCTQ (standard matrix for jpeg) by r

(T=scale\*DCTQ):

Then we get round () by divide element by element our subblock dct matrices (8x8) by Quantization matrix (T 8x8) to block high frequency and get real data that have been compressed in low frequencies:

function JPEGRes = QuantJPEG(splitDCT,DCTQ,scaling)

T=scaling\*DCTQ;

[l m row col]=size(splitDCT);

for i=1:row

for j=1:col

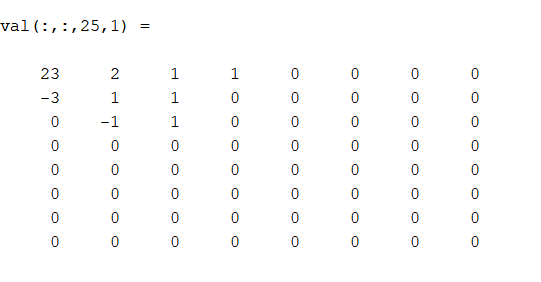
subIm=double(splitDCT(:,:,i,j));

JPEGRes(:,:,i,j) =round(subIm./T);

end

end

end

* **This is for example the 25th,1st block (8x8 ) after multiplying by factor and perform quantization** : 

1. **Rescaling**

In this function we multiply quantized blocks by Samling factor T to make it ready for decoding.

So, any value below zero will be zero because of round (), the result will be like the result of blockDCT, with ignoring the values which contain low information:

function rescaleIm = rescaling(QuntBlock,scaling,DCTQ)

T=scaling\*DCTQ;

[l m row col]=size(QuntBlock);

for i=1:row

for j=1:col

subIm=double(QuntBlock(:,:,i,j));

rescaleIm(:,:,i,j) =subIm.\*T;

end

end

end

For example:

This is the dct output of 1st,1st block: (it’s multiplied by 10^3)

Table

Description automatically generated

After rescaling the 1st ,1st block will be:

A picture containing table

Description automatically generated

1. **recombines Blocks**

after resampling and getting (IDCT) we merge sub-blocks again to recombine our image again.

Note: we get IDCT by using the same function of DCT but now parameter =1:

**The IDCT using DCTBlock:**

function blockDCT = DCTBlock(splits,C8,paramter)

[l m row col]=size(splits);

if paramter~=0

C8=C8';

end

for i=1:row

for j=1:col

subIm=double(splits(:,:,i,j));

blockDCT(:,:,i,j) =C8\*subIm\*C8';

end

end

end

now we will merge blocks using the inverse of the split function:

function JPEGImage = recombinesBlocks(resIDCT)

blockSize=8;

[l m row col]=size(resIDCT);

for i =1:row

for j=1:col

JPEGImage( (((i-1)\*blockSize)+1) :(i\*blockSize) ,(((j-1)\*blockSize)+1):(j\*blockSize) )=resIDCT(:,:,i,j);

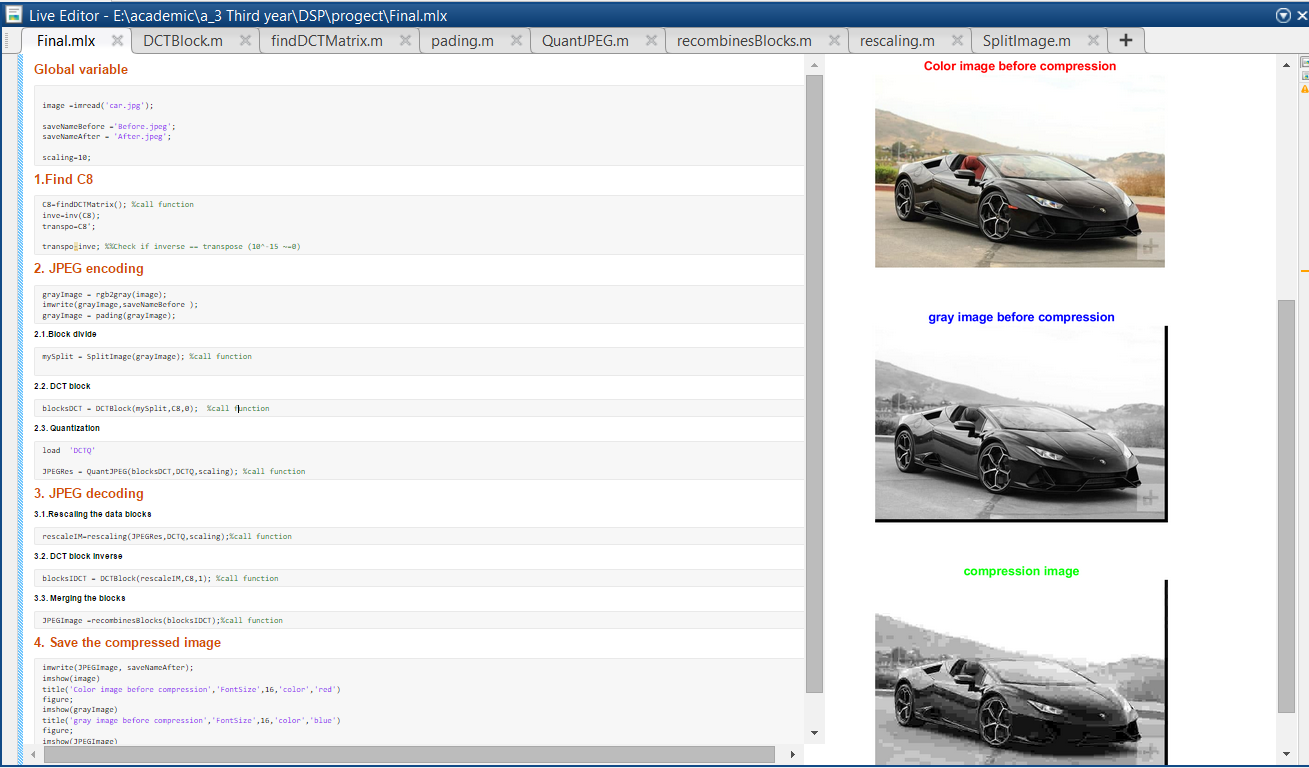
end

end

JPEGImage=uint8(JPEGImage);

end

**Change Scaling factor:**

Image before



Scaling factor =3

Scaling factor =5



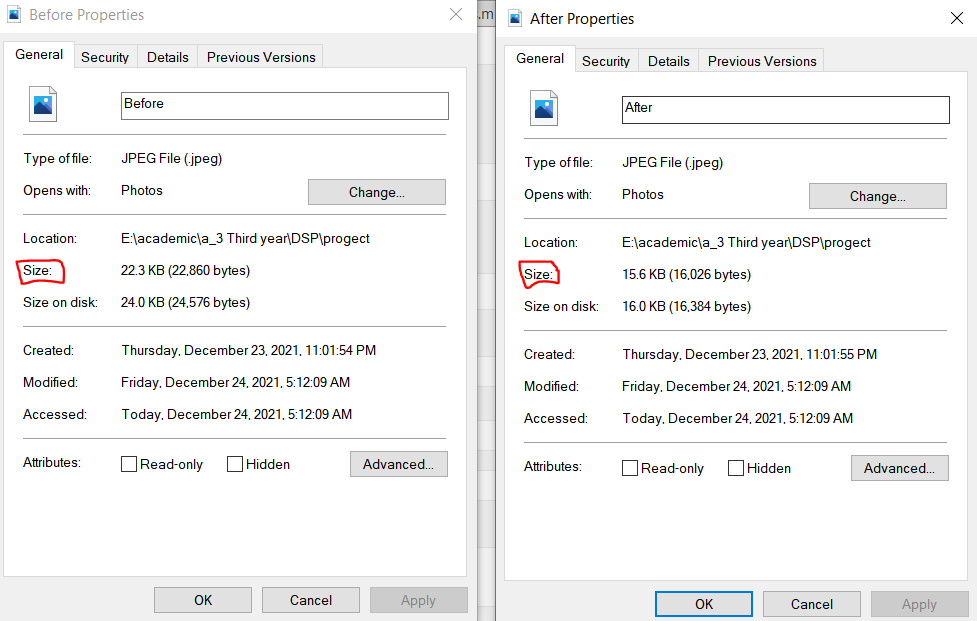
Scaling factor =10



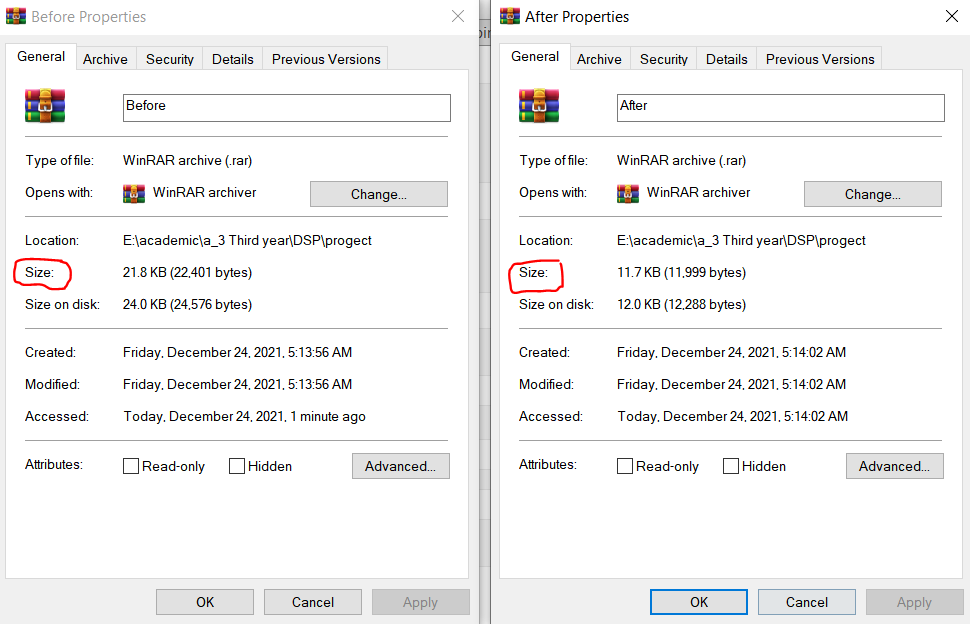
Scaling factor =20



And here we notice that size after compression is less than before:



And this if we get ( .rar file)



**Change Scaling factor:**

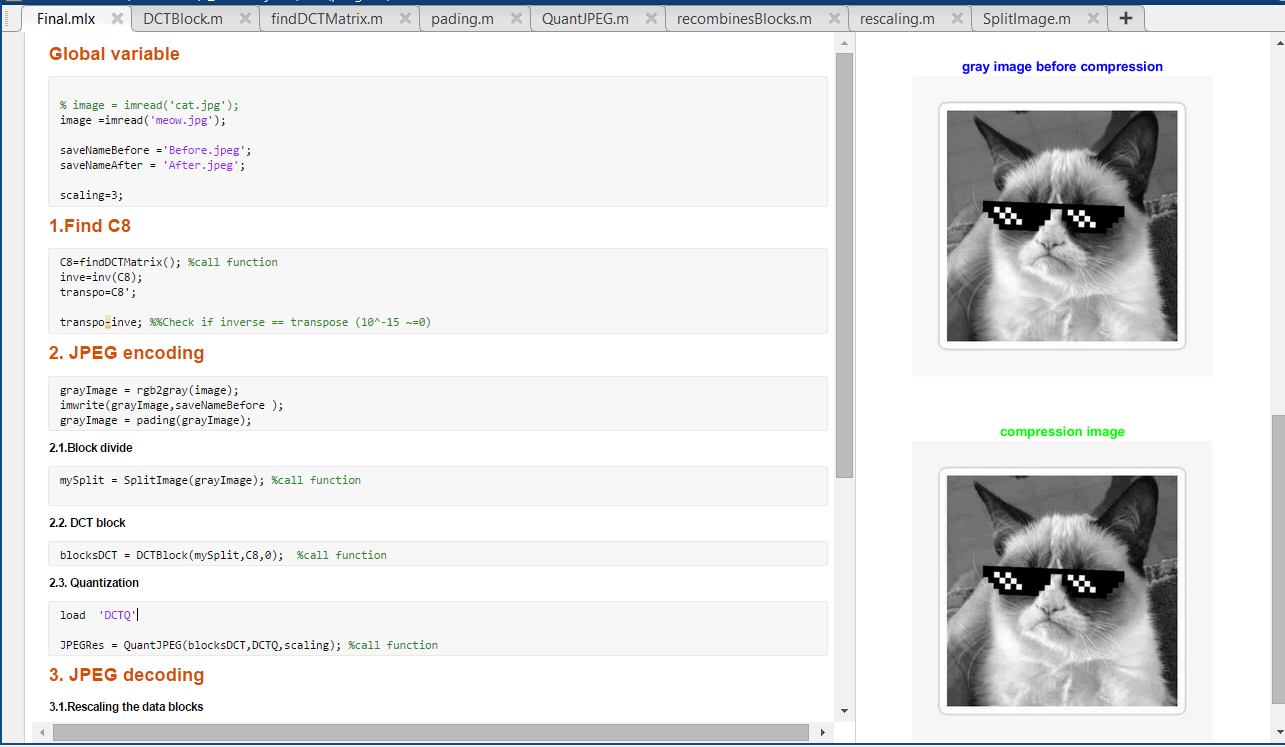
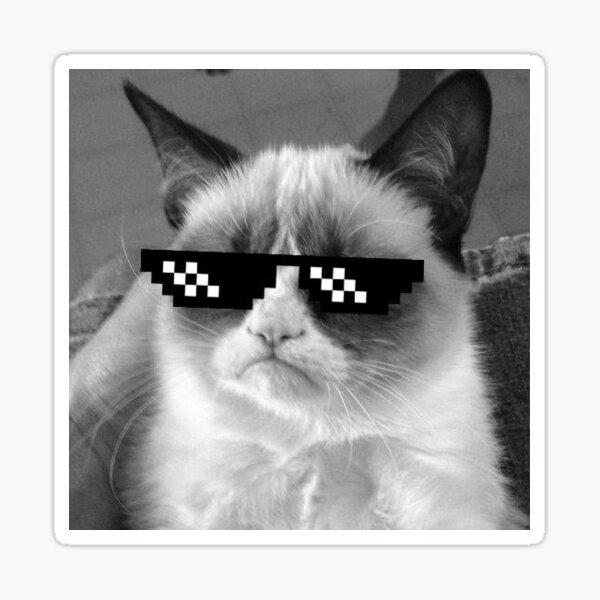


Image before



Scaling factor =1



Scaling factor =3



Scaling factor =5



Scaling factor =10

