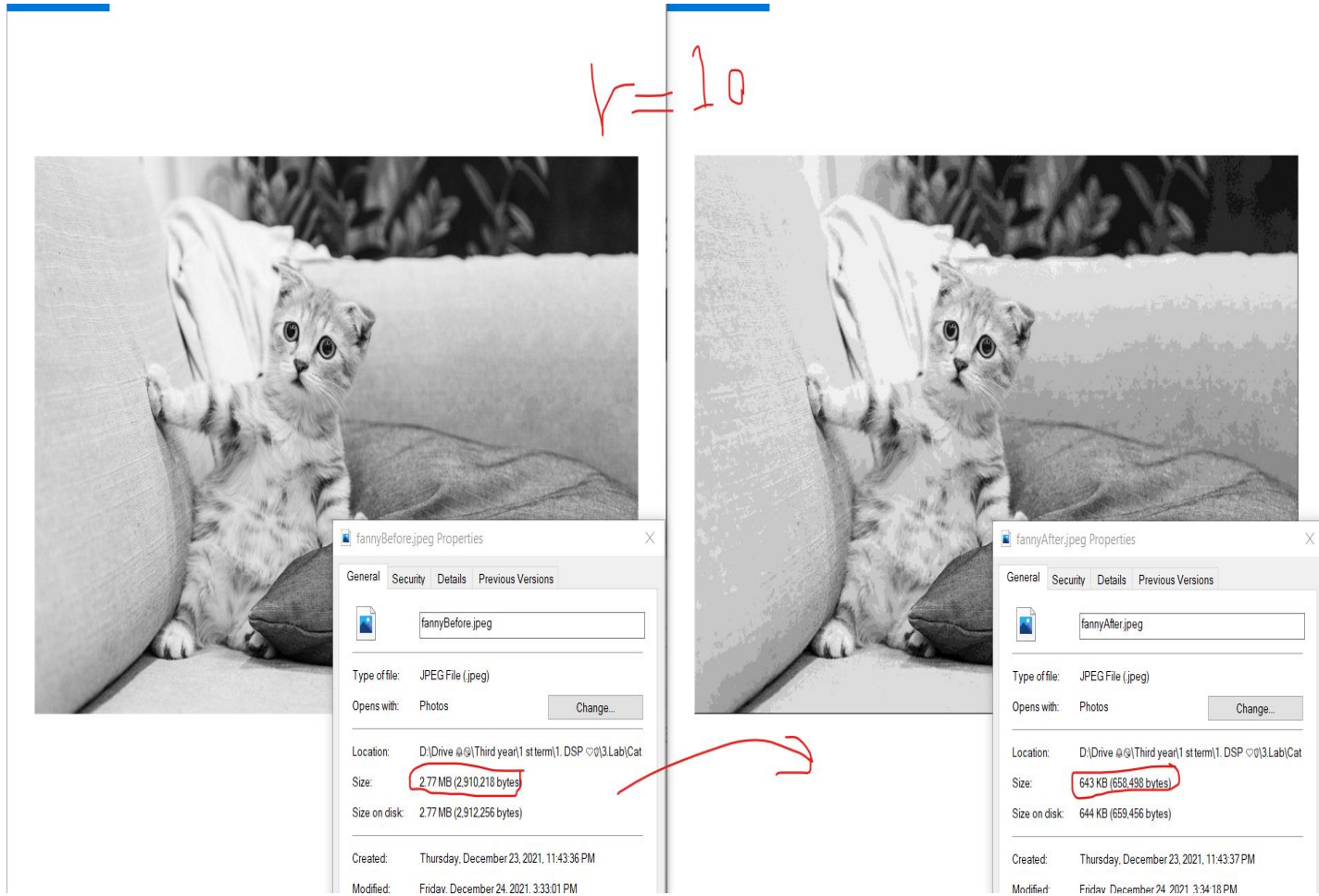


JPEG transform project



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

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 ( $10^{-15} \sim 0$ )
```

2. JPEG encoding

```
grayImage = rgb2gray(image);  
imwrite(grayImage,saveNameBefore );  
grayImage = padding(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

where the matrix C_N has elements

$$C_N(k, r) = u_k \cos \left(\frac{\pi}{N} k \left(r + \frac{1}{2} \right) \right) \quad (2)$$

Construct (C8) where :

$$u_0 = \sqrt{\frac{1}{N}} \quad \&\& \quad u_k = \sqrt{\frac{2}{N}} \text{ for } k > 0.$$

```
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
```

(2) DCT Block

Proposition 1. The two dimensional DCT of $m \times n$ matrix A is the product

$$\hat{A} = C_m A C_n^T \quad (1)$$

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
```

(3) Split Image

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

Result is a 4-D matrix $8 \times 8 \times \frac{row}{8} \times \frac{column}{8}$:

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

```

(4) Pad Zeros

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


```

function padGray = padding(grayIm)
[row column]=size(grayIm);
padrow=0;
padclo=0;
if(mod(row,8))
    num=floor(row/8)+1;
    padrow=num*8-row;
end
if(mod(column,8))
    num=floor(column/8)+1;
    padclo=num*8-column;
end
if((mod(column,8))&(mod(row,8)))
    padGray= padarray(grayIm,[padrow padclo],0,'post');
else
    padGray=grayIm;
end

```



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



Name	Value
ans	8x8 double
blocksDCT	4-D double
blocksIDCT	4-D double
C8	8x8 double
DCTQ	8x8 double
grayImage	424x632 uint8
image	417x625x3 uint8
inve	8x8 double

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

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

(5) Quantization JPEG

Here we will multiply DCTQ (standard matrix for jpeg) by r
($T = \text{scale} * \text{DCTQ}$):

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 :

```
val (:, :, 25, 1) =
```

[illegible]

(6) Rescaling

In this function we multiply quantized blocks by sampling 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(QuantBlock,scaling,DCTQ)
T=scaling*DCTQ;
[1 m row col]=size(QuantBlock);
for i=1:row
    for j=1:col
        subIm=double(QuantBlock(:, :, 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 10³)

```
blocksDCT(:, :, 1, 1) =  
1.0e+03 *  
1.2853    0.0052    0.0040   -0.0001   -0.0002    0.0002   -0.0079    0.0099  
0.0082   -0.0020    0.0020   -0.0059    0.0041    0.0002   -0.0001    0.0000  
-0.0059   -0.0002   -0.0036    0.0040   -0.0001   -0.0002   -0.0001    0.0003  
0.0018    0.0005    0.0039    0.0003   -0.0001   -0.0002   -0.0004   -0.0000  
0.0000   -0.0002    0.0000    0.0003    0.0000   -0.0001    0.0000   -0.0003  
-0.0002    0.0000   -0.0007    0.0002   -0.0001   -0.0001    0.0005   -0.0006  
0.0000   -0.0001    0.0001    0.0004   -0.0002   -0.0002    0.0003    0.0007  
0.0005   -0.0000    0.0005    0.0005    0.0001    0.0001    0.0002   -0.0002
```

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

[illegible]

(7) 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:

Live Editor - E:\academic\3 Third year\DSP\project\Final.mlx

Final.mlx x DCTBlock.m x findDCTMatrix.m x padding.m x QuantJPEG.m x recombinesBlocks.m x rescaling.m x SplitImage.m x +

Global variable

```
image = imread('car.jpg');
saveNameBefore = 'Before.jpg';
saveNameAfter = 'After.jpg';
scaling=10;
```

1.Find C8

```
C8=findDCTMatrix(); %call function
inv=inv(C8);
transpo=C8';
transpo[inv]; %check if inverse == transpose (38*15 ==0)
```

2. JPEG encoding

```
grayImage = rgb2gray(image);
imwrite(grayImage,saveNameBefore);
grayImage = padding(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);
```

Color image before compression



gray image before compression



compression image




IMAGE BEFORE



SCALING FACTOR =3



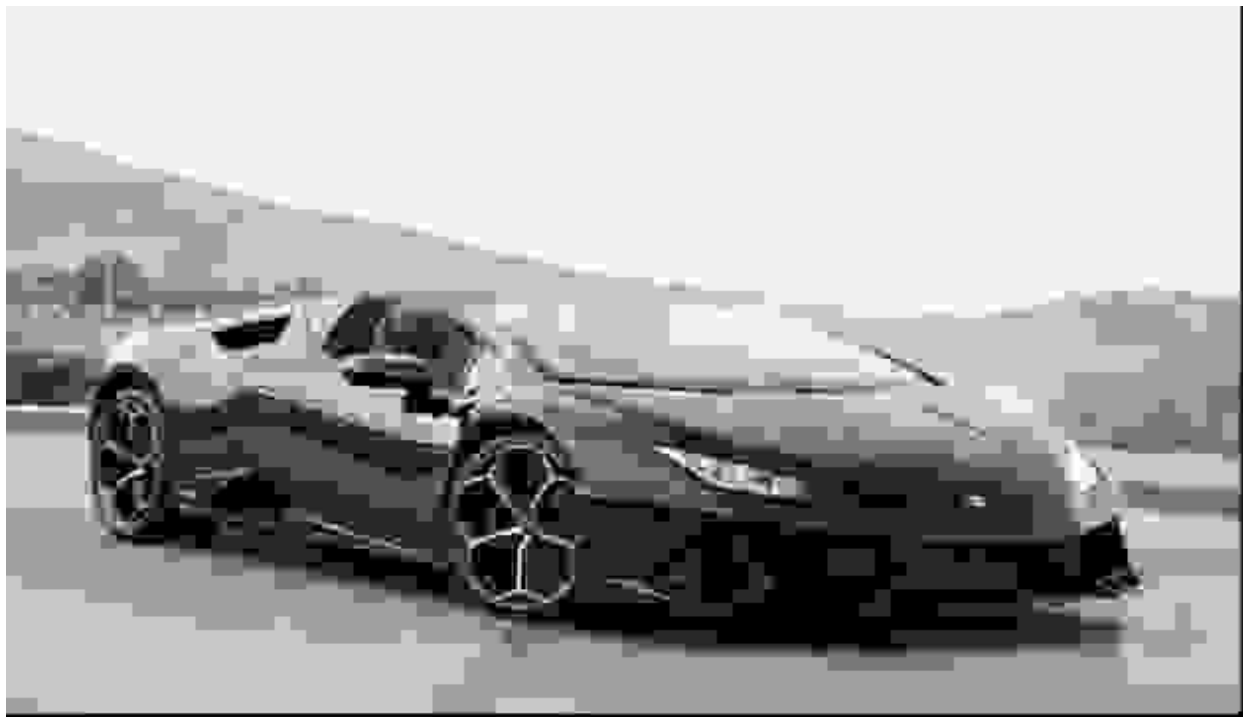
SCALING FACTOR =5



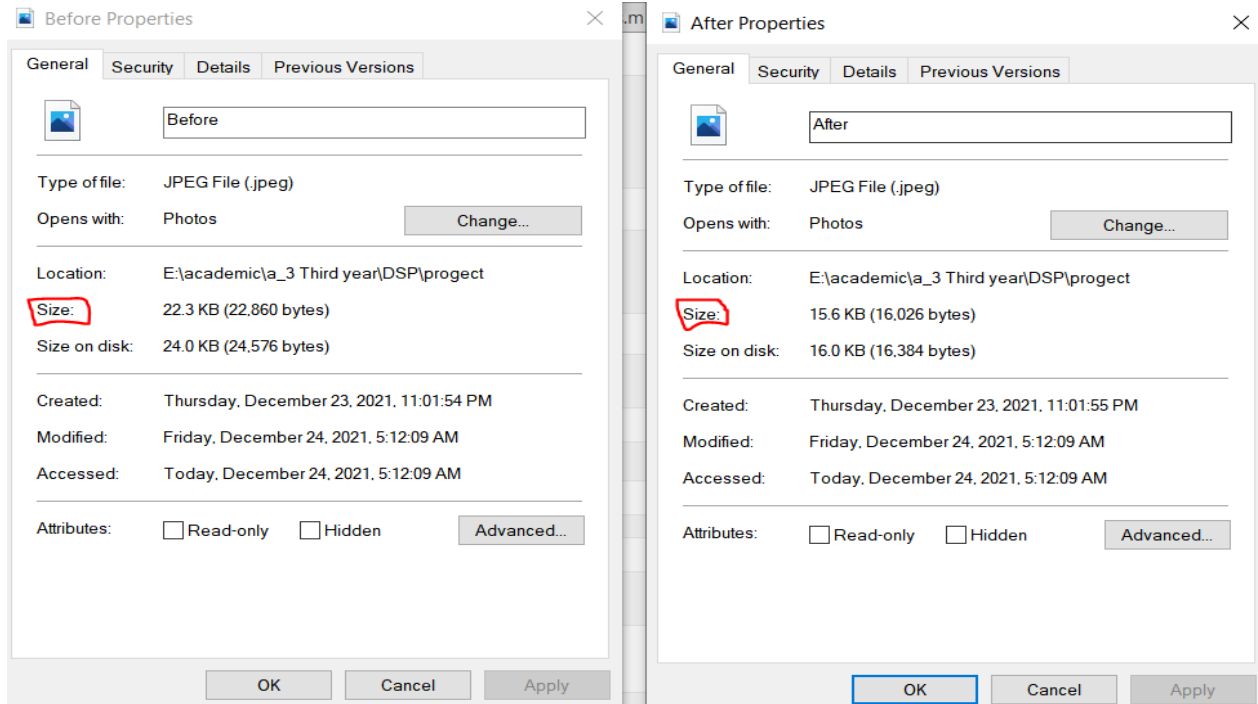
SCALING FACTOR = 10



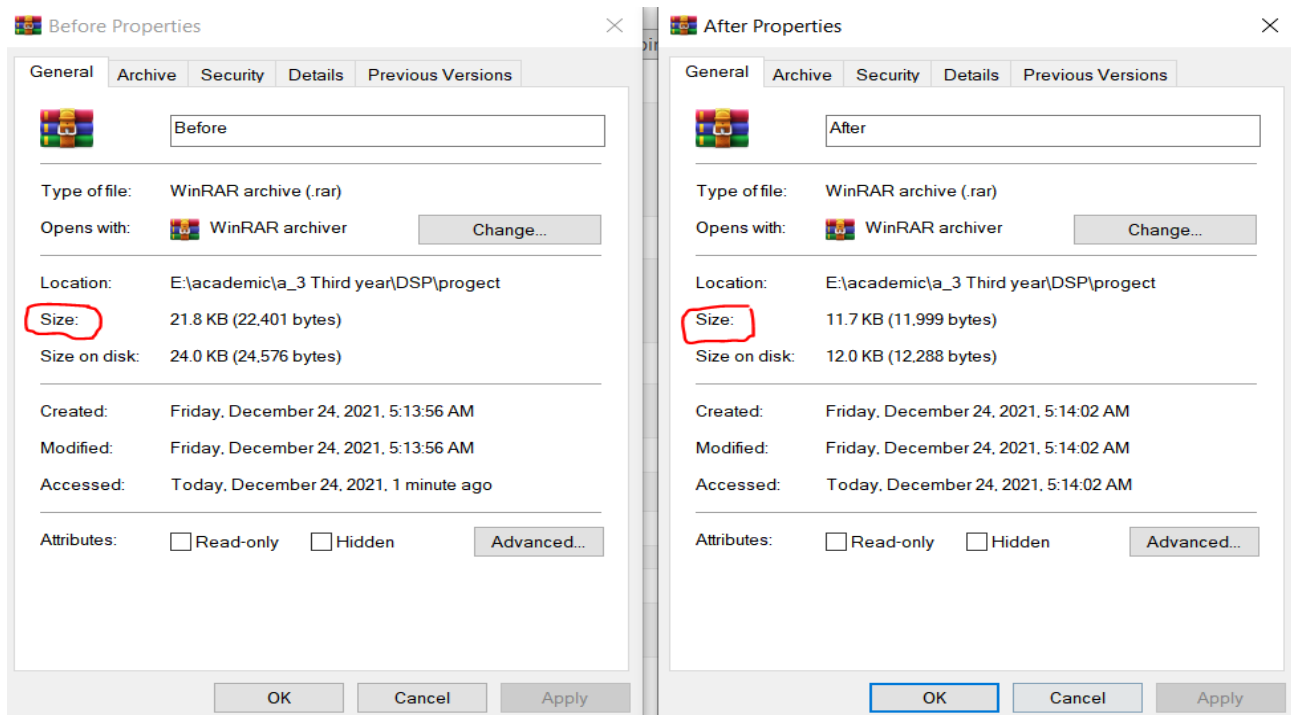
SCALING FACTOR = 20



AND HERE WE NOTICE THAT SIZE AFTER COMPRESSION IS LESS THAN BEFORE:



And this if we get (.rar file)



Change Scaling factor:

Final.mlx × DCTBlock.m × findDCTMatrix.m × padding.m × QuantJPEG.m × recombinesBlocks.m × rescaling.m × SplitImage.m × +

Global variable

```
% image = imread('cat.jpg');  
image =imread('meow.jpg');  
  
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 = padding(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

gray image before compression



compression image


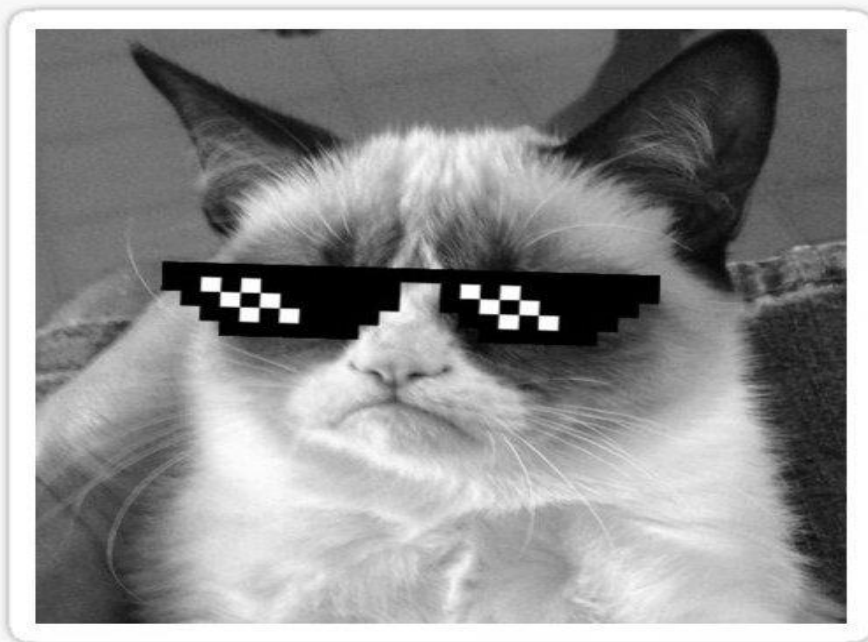


IMAGE BEFORE



SCALING FACTOR = 1



SCALING FACTOR = 3



SCALING FACTOR =5



SCALING FACTOR =10

