**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem VI

**Course: Image Processing**

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.10**

**A.1 Aim:**

Write a program to compress the image (your own photograph) using energy compaction concept of discrete cosine transform and calculate RMSE, PSNR and compression ratio.

**A.2 Prerequisite:**

1 Matlab programming syntax (Refer the Matlab manual).

2. Knowledge of fundamentals of Image Compression & Properties of DCT.

2. Soft copy of input image.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Compress the given input image.
2. Calculate RMSE, PSNR and Compression ratio.
3. Identify other image compression techniques and its applications

**A.4 Theory:**

**A.4.1. Data Compression**

**1. Image Compression**

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from Web pages.

There are several different ways in which image files can be compressed. For Internet use, the two most common compressed graphic image formats are the [JPEG](http://searchsoa.techtarget.com/definition/JPEG) format and the[GIF](http://searchwindevelopment.techtarget.com/definition/GIF) format. The JPEG method is more often used for photographs, while the GIF method is commonly used for line art and other images in which geometric shapes are relatively simple. Mathematical Transformation functions like DCT can be used for image compression.

**1. DCT**

The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain.

**DCT Encoding :**

The general equation for a 1D (*N* data items) DCT is defined by the following equation:

 (1.1)

and the corresponding ***inverse*** 1D DCT transform is simple *F-1*(*u*), where

 (1.2)

The general equation for a 2D (*N* by *M* image) DCT is defined by the following equation:

 (1.3)

and the corresponding ***inverse*** 2D DCT transform is simple *F-1*(*u*,*v*), where

 (1.4)

The basic operation of the DCT is as follows:

* The input image is N by M;
* f(i,j) is the intensity of the pixel in row i and column j;
* F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.
* For most images, much of the signal energy lies at low frequencies; these appear in the

upper left corner of the DCT.

Compression is achieved since the lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion. This achieves the energy compaction.

**3. Performance Evaluation parameters:**

* 1. **Root mean squared error (RMSE):**

In order to evaluate the performance of the image compression coding, it is necessary to define a measurement that can estimate the difference between the original image and the decoded image. Two common used measurements are the **Mean Square Error (MSE)** and the **Peak Signal to Noise Ratio (PSNR)**, which are defined in (1.5) and (1.6), respectively. f(x,y) is the pixel value of the original image, and f’(x,y)is the pixel value of the decoded image. Most image compression systems are designed to minimize the MSE and maximize the PSNR.

 **(1.5)**

* 1. **Peak signal to noise ratio (PSNR):**

 **(1.6)**

* 1. **Compression Ratio (CR):**

CR = Compressed image size/ uncompressed image size **(1.7)**

The ratio should be between 0 and 1 (multiply with 100 to get ratio in percentages)

If CR>1 that means compressed image size is greater than actual uncompressed image size. If CR is just below 1 means bad compression.

**A.5 Procedure/Algorithm:**

**A.5.1:**

**TASK 1:**

1. Read 3 different input images as per following categories.
2. Your photograph
3. Flower image
4. Animal image

2. Apply DCT on the image as per following directions separately for each image.

a. Full DCT

b. 4x4 Block DCT

c. 8x8 Block DCT

3. Eliminate 50% spatial frequencies (take in a matrix of DCT coefficients and set everything beyond the upper left triangle to zero) for each method to compress the image.

4. Reconstruct the compressed image and fill up the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Subjective quality \* | RMSE | PSNR | CR |
| Full DCT |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 4x4 Block DCT |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 8x8 Block DCT |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

\* Comments should be (no difference/less difference/ maximum difference)

5. Observe/compare all outputs and complete PART B of lab manual.

6. Save and close the file and name it as **EX10\_Task1\_your Roll no.m**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PART B

|  |  |
| --- | --- |
| Roll No : B032 | Name: Naman Garg |
| Class : B.Tech CS-B | Batch : B2 |
| Date of Experiment: 23-9-2020 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

img=imread('anml.jpg');

img=rgb2gray(img);

img=imresize(img,[256,256],'nearest');

imwrite(img , 'originalimage.jpg')

dct=dctmtx(256);

dct\_t=dct.';

F=dct\*double(img)\*dct\_t;

dct\_4=dctmtx(4);

dct\_4t=dct\_4.';

F4=zeros(256);

for i=1:4:256

for j=1:4:256

F4(i:i+3,j:j+3)=dct\_4\*double(img(i:i+3,j:j+3))\*dct\_4t;

end

end

dct\_8=dctmtx(8);

dct\_8t=dct\_8.';

F8=zeros(256);

for i=1:8:256

for j=1:8:256

F8(i:i+7,j:j+7)=dct\_8\*double(img(i:i+7,j:j+7))\*dct\_8t;

end

end

F4\_30=compression\_30\_4x4(F4);

F4\_50=compression\_50\_4x4(F4);

F4\_70=compression\_70\_4x4(F4);

f4\_30=decompressed\_image(dct\_4,dct\_4t,F4\_30);

f4\_50=decompressed\_image(dct\_4,dct\_4t,F4\_50);

f4\_70=decompressed\_image(dct\_4,dct\_4t,F4\_70);

F8\_30=compression\_30\_8x8(F8);

F8\_50=compression\_50\_8x8(F8);

F8\_70=compression\_70\_8x8(F8);

f8\_30=decompressed\_image\_8x8(dct\_8,dct\_8t,F8\_30);

f8\_50=decompressed\_image\_8x8(dct\_8,dct\_8t,F8\_50);

f8\_70=decompressed\_image\_8x8(dct\_8,dct\_8t,F8\_70);

f1=compression\_50(dct,dct\_t,F);

f2=compression\_30(dct,dct\_t,F);

f3=compression\_70(dct,dct\_t,F);

subplot(3,3,1),imshow(uint8(f2)),title('70%');

subplot(3,3,2),imshow(uint8(f1)),title('50%');

subplot(3,3,3),imshow(uint8(f3)),title('30%');

subplot(3,3,4),imshow(uint8(f4\_30)),title('4x4');

subplot(3,3,5),imshow(uint8(f4\_50)),title('4x4');

subplot(3,3,6),imshow(uint8(f4\_70)),title('4x4');

subplot(3,3,7),imshow(uint8(f8\_30)),title('8x8');

subplot(3,3,8),imshow(uint8(f8\_50)),title('8x8');

subplot(3,3,9),imshow(uint8(f8\_70)),title('8x8');

imwrite(uint8(f1),'img\_50.jpg');

imwrite(uint8(f2),'img\_30.jpg');

imwrite(uint8(f3),'img\_70.jpg');

imwrite(uint8(f4\_30),'img\_30\_4.jpg');

imwrite(uint8(f4\_50),'img\_50\_4.jpg');

imwrite(uint8(f4\_70),'img\_70\_4.jpg');

imwrite(uint8(f8\_30),'img\_30\_8.jpg');

imwrite(uint8(f8\_50),'img\_50\_8.jpg');

imwrite(uint8(f8\_70),'img\_70\_8.jpg');

fprintf('RMSE Values:\n');

fprintf('30 percent compression %f\n',rmse\_value(img,f2));

fprintf('50 percent compression %f\n',rmse\_value(img,f1));

fprintf('70 percent compression %f\n',rmse\_value(img,f3));

fprintf('\n4x4 block\n');

fprintf('30 percent compression %f\n',rmse\_value(img,f4\_30));

fprintf('50 percent compression %f\n',rmse\_value(img,f4\_50));

fprintf('70 percent compression %f\n',rmse\_value(img,f4\_70));

fprintf('\n8x8 block\n');

fprintf('30 percent compression %f\n',rmse\_value(img,f8\_30));

fprintf('50 percent compression %f\n',rmse\_value(img,f8\_50));

fprintf('70 percent compression %f\n',rmse\_value(img,f8\_70));

fprintf('\nPSNR Values:\n');

fprintf('30 percent compression %f\n',20\*log10(255/rmse\_value(img,f2)));

fprintf('50 percent compression %f\n',20\*log10(255/rmse\_value(img,f1)));

fprintf('70 percent compression %f\n',20\*log10(255/rmse\_value(img,f3)));

fprintf('\n4x4 block\n');

fprintf('30 percent compression %f\n',20\*log10(255/rmse\_value(img,f4\_30)));

fprintf('50 percent compression %f\n',20\*log10(255/rmse\_value(img,f4\_50)));

fprintf('70 percent compression %f\n',20\*log10(255/rmse\_value(img,f4\_70)));

fprintf('\n8x8 block\n');

fprintf('30 percent compression %f\n',20\*log10(255/rmse\_value(img,f8\_30)));

fprintf('50 percent compression %f\n',20\*log10(255/rmse\_value(img,f8\_50)));

fprintf('70 percent compression %f\n',20\*log10(255/rmse\_value(img,f8\_70)));

fprintf('\nCR Values:\n');

fprintf('30 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_30.jpg').FileSize);

fprintf('50 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_50.jpg').FileSize);

fprintf('70 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_70.jpg').FileSize);

fprintf('\n4x4 block\n');

fprintf('30 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_30\_4.jpg').FileSize);

fprintf('50 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_50\_4.jpg').FileSize);

fprintf('70 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_70\_4.jpg').FileSize);

fprintf('\n8x8 block\n');

fprintf('30 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_30\_8.jpg').FileSize);

fprintf('50 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_50\_8.jpg').FileSize);

fprintf('70 percent compression %f\n',imfinfo('originalimage.jpg').FileSize/imfinfo('img\_70\_8.jpg').FileSize);

function f4=decompressed\_image(dct\_4,dct\_4t,F4)

f4=zeros(256);

for i=1:4:256

for j=1:4:256

f4(i:i+3,j:j+3)=dct\_4t\*double(F4(i:i+3,j:j+3))\*dct\_4;

end

end

end

function f4=decompressed\_image\_8x8(dct\_4,dct\_4t,F4)

f4=zeros(256);

for i=1:8:256

for j=1:8:256

f4(i:i+7,j:j+7)=dct\_4t\*double(F4(i:i+7,j:j+7))\*dct\_4;

end

end

end

function F2=compression\_30\_4x4(F)

F2=zeros(256,256);

for i=1:4:256

for j=1:4:256

for k=i:i+1

for l=j:j+1-k+i

F2(k,l)=F(k,l);

end

end

end

end

end

function F1=compression\_50\_4x4(F1)

for i=1:4:256

for j=1:4:256

for k=i:i+3

for l=j+3-k+i:j+3

F1(k,l)=0;

end

end

end

end

end

function F=compression\_70\_4x4(F)

for i=1:4:256

for j=1:4:256

for k=i+2:i+3

for l=j+5-k+i:j+3

F(k,l)=0;

end

end

end

end

end

function F2=compression\_30\_8x8(F)

F2=zeros(256,256);

for i=1:8:256

for j=1:8:256

for k=i:i+3

for l=j:j+3-k+i

F2(k,l)=F(k,l);

end

end

end

end

end

function F1=compression\_50\_8x8(F1)

for i=1:8:256

for j=1:8:256

for k=i:i+7

for l=j+7-k+i:j+7

F1(k,l)=0;

end

end

end

end

end

function F=compression\_70\_8x8(F)

for i=1:8:256

for j=1:8:256

for k=i+5:i+7

for l=j+12-k+i:j+7

F(k,l)=0;

end

end

end

end

end

function f2=compression\_30(dct,dct\_t,F)

F2=zeros(256,256);

for i=1:90

for j=1:91-i

F2(i,j)=F(i,j);

end

end

f2=dct\_t\*F2\*dct;

end

function f1=compression\_50(dct,dct\_t,F1)

for i=1:256

for j=256-i+1:256

F1(i,j)=0;

end

end

f1=dct\_t\*F1\*dct;

end

function f3=compression\_70(dct,dct\_t,F)

F3=F;

for i=180:256

for j=436-i:256

F3(i,j)=0;

end

end

f3=dct\_t\*F3\*dct;

end

function rmse=rmse\_value(img,f)

rmse=0.0;

for i=1:256

for j=1:256

rmse=rmse+ (double(img(i,j))-double(f(i,j)))\*(double(img(i,j))-double(f(i,j)));

end

end

rmse=rmse/(256\*256);

rmse=sqrt(rmse);

end

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

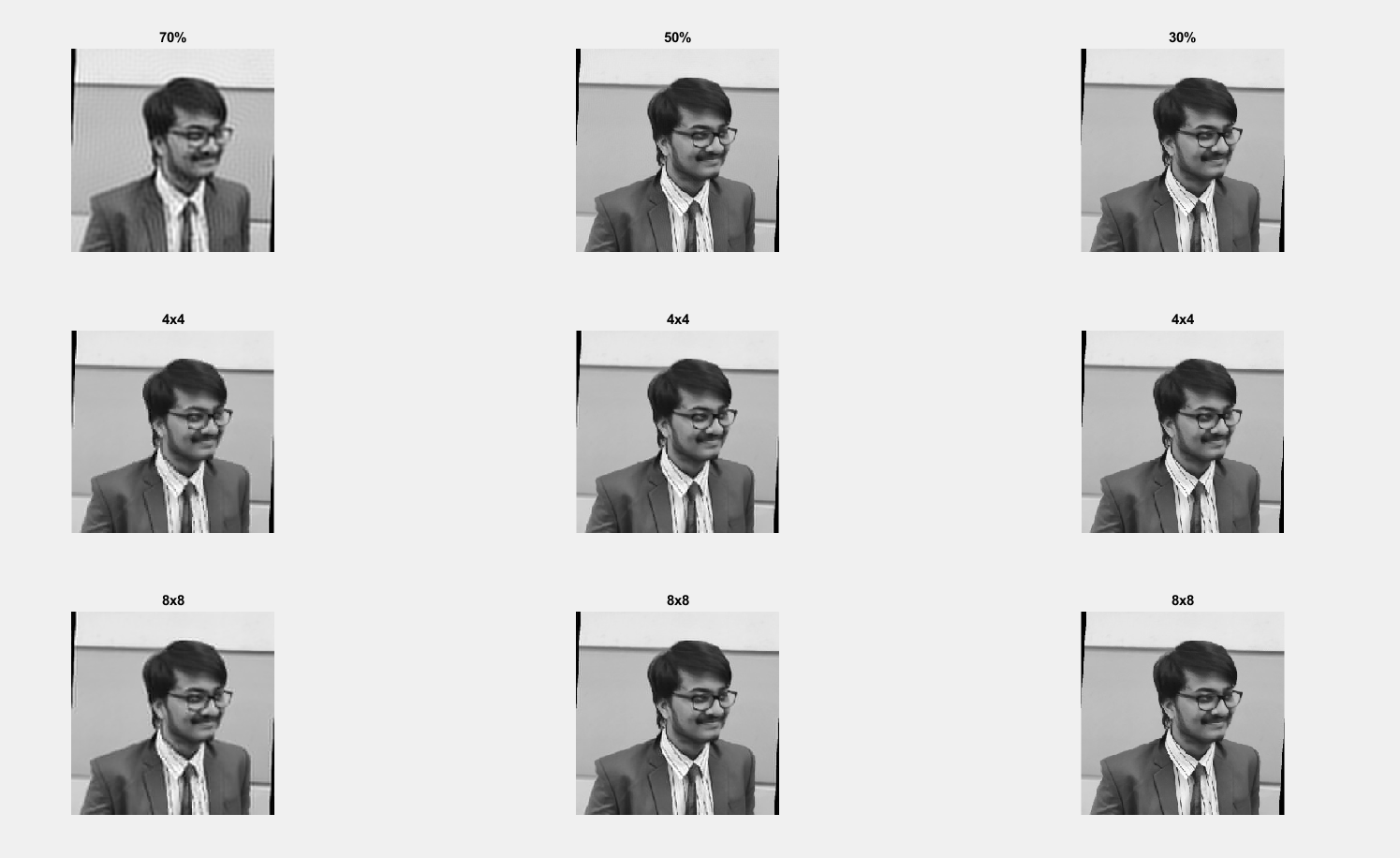
**Input Images:**

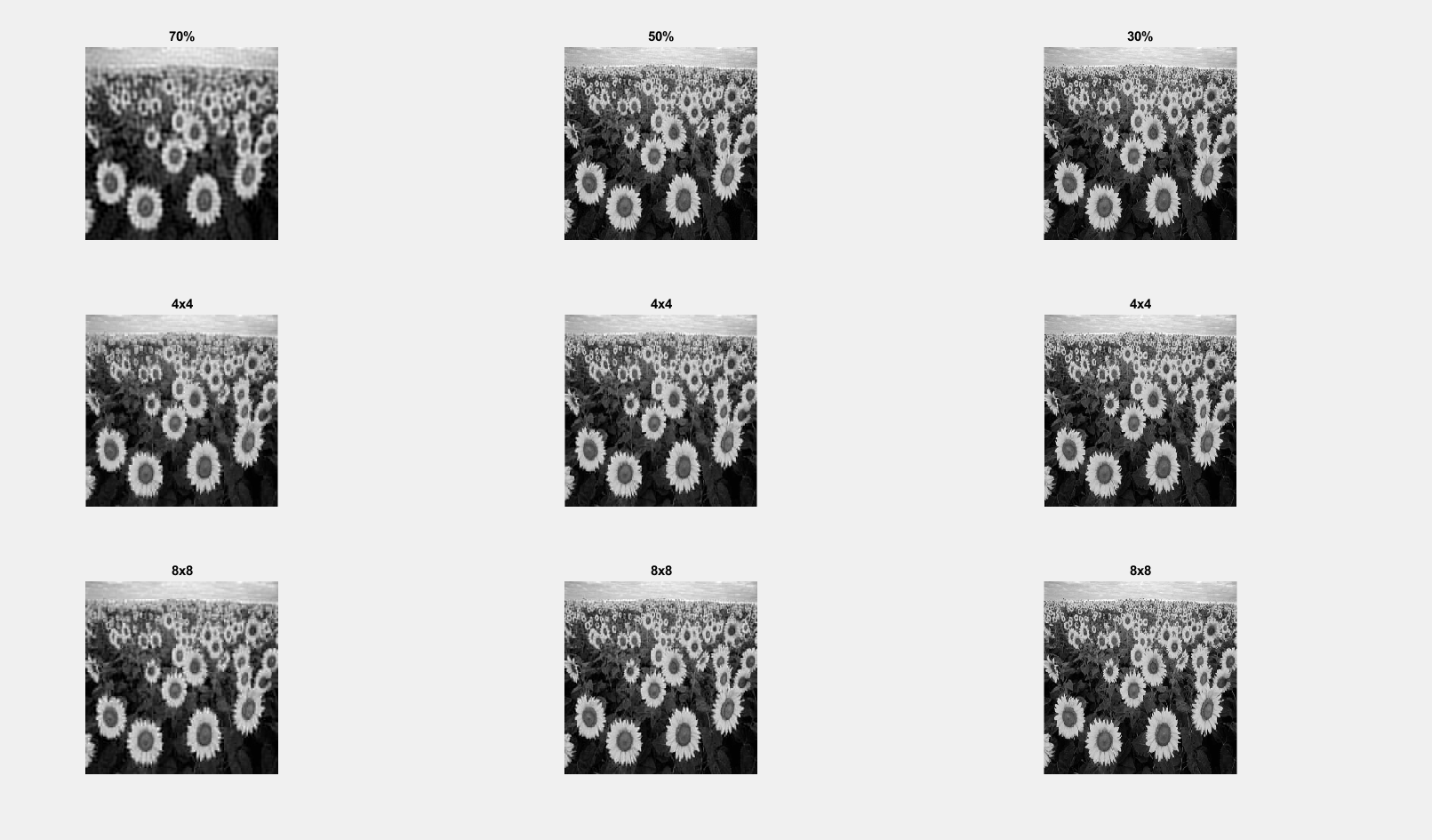
****

**Output Images:**

1. **For each edge detection operatoras per the procedure discussed in section A.5.**

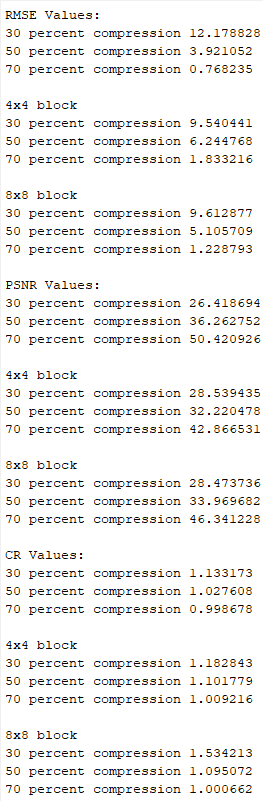
**NAMAN’s Image**

****

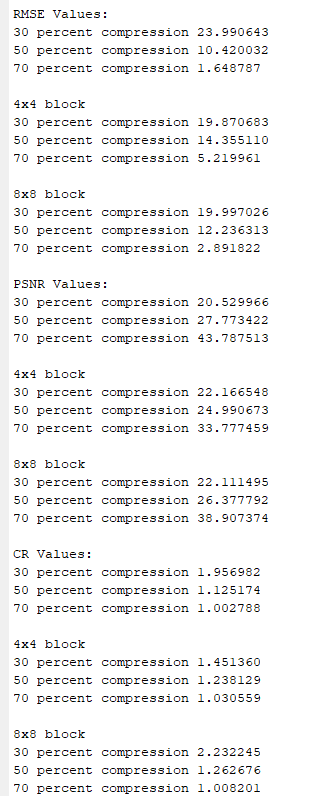
**Flower Image**

**Animal Image**

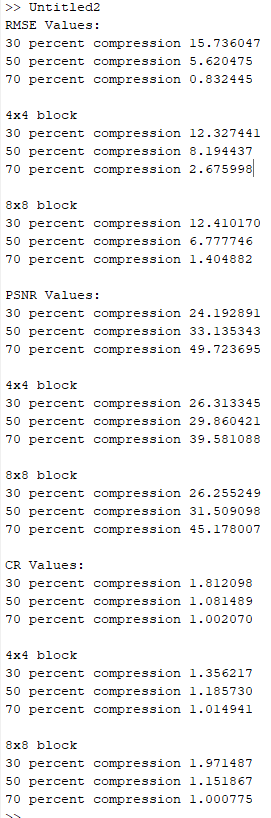
My image::



Flower::



Animal:



**B.3 Observations and learning:**

**My Image:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Subjective quality \* | RMSE | PSNR | CR |
| **Full DCT** – 30% Retention | The image is a bit blurred with black lines outside my image. The white spots on my shirt is very dull and blurred here. If we are not interested in the details then, we can opt for this. | 8.958972 | 29.085640 | 0.744874 |
| 50% Retention | The image looks quite identical to my image and if we prefer details , we can go for this. | 4.366190 | 35.328750 | 0.935067 |
| 70% Retention | The image is identical to mine and it might probably have more details than the 50% retention but the difference cannot be perceived by the eye. | 0.910785 | 48.942487 | 0.998068 |
| **4x4 Block DCT –**  30% Retention | Not much blurring is not observed here (equivalent to no blurring). The white spots are not blurred but are dull in terms of the brightness | 7.446125 | 30.692197 | 0.765973 |
| 50% Retention | The image looks identical to mine and the white spots on my shirt are brighter here but some dull portion can be seen. | 5.560114 | 33.229130 | 0.866568 |
| 70% Retention | The image looks the most identical to mine and the white spots are the brightest here and seem to retain the original quality here. | 2.283975 | 40.956977 | 0.985587 |
| **8x8 Block DCT-**  30%  Retention | The output is similar to 4x4 one with similar amount of details retained | 7.627069 | 30.483650 | 0.614859 |
| 50% Retention | The image has some more details here as the white spots are a bit brighter here. So we can observe some increase in level of details. | 4.970476 | 34.202845 | 0.878158 |
| 70% Retention | The image has the white spots more whiter here and we can see the most level of details here. | 1.523097 | 44.476250 | 0.994799 |

**Flower Image:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Subjective quality \* | RMSE | PSNR | CR |
| **Full DCT** – 30% Retention | The flower is a bit blurred .If we are not interested in the details then, we can opt for this. | 8.220108 | 29.833253 | 0.868273 |
| 50% Retention | The image looks quite identical to the original image and if we prefer details , we can go for this. | 2.733646 | 39.395959 | 0.959750 |
| 70% Retention | The image is identical to original and it might probably have more details than the 50% retention but the difference cannot be perceived by the eye. | 0.655288 | 51.802156 | 0.998933 |
| **4x4 Block DCT –**  30% Retention | Not much blurring is observed here (equivalent to no blurring). | 6.462687 | 31.922542 | 0.857600 |
| 50% Retention | The image looks identical to the original . | 3.980246 | 36.132606 | 0.910047 |
| 70% Retention | The image looks identical to mine seem to retain the original quality here. | 1.495389 | 44.635720 | 0.991310 |
| **8x8 Block DCT-**  30%  Retention | The output is similar to 4x4 one with similar amount of details retained | 6.348929 | 32.076794 | 0.698125 |
| 50% Retention | We can observe some increase in level of details. | 3.234230 | 37.935386 | 0.931849 |
| 70% Retention | The image seems to retain most of the level of details here. | 0.933871 | 48.725068 | 0.997561 |

**Animal Image:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Subjective quality \* | RMSE | PSNR | CR |
| **Full DCT** – 30% Retention | The image is blurred and the stripes on the tiger are blurred a lot. If we want to go for details, we should decrease the compression level or increase the retention of information | 22.241996 | 21.187328 | 0.525765 |
| 50% Retention | The image looks quite identical to the original image and if we prefer details , we can go for this. | 10.230533 | 27.932838 | 0.887283 |
| 70% Retention | The image is identical to the original and it might probably have more details than the 50% retention but the difference cannot be perceived by the eye. | 2.390306 | 40.561732 | 0.993246 |
| **4x4 Block DCT –**  30% Retention | Not much blurring is observed here (equivalent to no blurring). The stripes are not blurred but are a bit dull in terms of the brightness | 18.373095 | 22.847157 | 0.675761 |
| 50% Retention | The image looks identical to the original and the black stripes on the shirt are brighter here. | 13.691138 | 25.402013 | 0.811238 |
| 70% Retention | The image looks the most identical to the original and the black stripes are the brightest here and seem to retain the original quality here. | 5.653888 | 33.083859 | 0.968123 |
| **8x8 Block DCT-**  30%  Retention | The output is similar to 4x4 one with similar amount of details retained | 18.755930 | 22.668032 | 0.455190 |
| 50% Retention | The image has some more details here as the black stripes are a bit brighter here. So we can observe some increase in level of details. | 11.778149 | 26.709262 | 0.796785 |
| 70% Retention | The image has the black stripes more distinct here and we can see the most level of details here. | 3.376468 | 37.561551 | 0.988114 |

Since the dct transformation applied on an image will give energy levels on the top left corner of the output represented by the white pixels there; most of the information about the original image will be stored on the top left corner. Very less and relatively less significant information is scattered in the rest of the image. This makes the discrete cosine transform very useful for data compression. We can pick a diagonal line across the transformed matrix and make the elements below the diagonal line zero. The word ‘below’ is important as the data above will be towards the top left corner where the more significant data will be stored. The diagonal can be chosen such that it creates 30-70 or 50-50 or 70-30 split of data and zeros. The higher the information we retain, the more accurate will be the compressed image with respect to the original image but it will be very less compressed and vice versa. The accuracy metrics wrt information retention are rmse which will measure the mean squared error between the pixel values of the compressed image and the original image ; psnr will be towards the accuracy side as the rmse part in the formula is in the denominator denoting an inverse relation between rmse and psnr ; CR which will be the measure of compression as the ratio of compressed image size and original image size. When the dct compression is applied in blocks of 4x4 or 8x8, the 30% data dropped will be in different areas of the matrix rather than all information being dropped below the diagonal. This will give result to different values of the above metrics. Data compression should be performed in such a way that we get to reduce the size of the image as much as possible without loosing the necessary information. We can see that when we retain only 30% information, even then we don’t loose any significant information and very less blur is observed. We can choose to use that compressed image if we don’t want that much details. However for the animal image, the compressed image with only 30% information retained faces severe blurring. This can be seen in the rmse value which is more than double than that for my image and the flower image which indicates that the information dropped had a lot of significant data of the image. Also since data lost is more here, CR is the least here which means high level of compression is achieved for the tiger photo.

So overall, when we go for DCT transform, the full DCT with only 30% info retained shows some blurring and can suffer in the minute details part example : white spots on my shirt, edges of the petals of the flower, stripes on the tiger. Full DCT with 50% and 70% information retained are similar with the only difference being the level of compression which also doesn’t have a significant difference. The 4x4 and 8x8 blocks have similar outputs but 8x8 blocks have lesser CR value (better compression) for 30% information retention. And when it comes to best method for compression, we can choose to go for 8x8 block with 30% information retention as it is more detailed than a full DCT with 30% retention and has the highest amount of compression indicated by the least CR value.

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

I learnt and successfully implemented image compression using the discrete cosine transformation in MATLAB for my image, flower image and an animal image. I dropped information below the diagonal lines and the diagonal line themselves were shifted both towards the top left corner and the bottom right corner. This created a variation in the amount of information dropped and the amount of compression achieved. More the information is dropped, less accurate is the compressed image wrt the original image and more is the compression in size. I dropped information in blocks as well which made the dropping of information uniform across the transformed matrix and not just below the main diagonal in the entire matrix. This will lead to different values of the metrics. The metrics implemented were rmse, psnr and CR. I understood the applications of the data compression techniques.

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Do you find the steps applied for image compression in this experiment needs further improvement ? Justify your answer.

Yes, some improvement can be done. The compression done such that only 30% information was retained led to the compressed image being 0.5x- 0.6x the original image in size but not a lot of the significant information is lost. We can opt for more compression as I think the diagonal line can be tried to be shifted towards the top left corner more. This can be done until we reach a point where we start to loose valuable information. Until then, we can opt for more compression. This can be necessary when the images are of huge sizes so any transformation or processing we wish to apply will take lots of time. In areas such as machine learning where we would want to preprocess entire datasets in 1 pipeline, the entire step can take lots of time if image size is high and so is the number of images. If we can make do with less detailed images, we can apply image compression on these images to reduce time. In terms of algorithm, if there exists a greedy approach to lessen the rmse while picking the least amount of information of the F matrix instead of just dropping the information below the diagonal without considering its effect on rmse, we can optimize the algorithm to output a very accurate image with highest amount of compression.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*