西北工業大學

Experiment Report of Multimedia

Selected Question: Question No.1

Name: Dikshya Kafle

Student ID: 2018380039

Deadline: 22nd Nov 2020

Submitted on: 22 Nov 2020

Histogram Equalization:

<u>Introduction:</u>

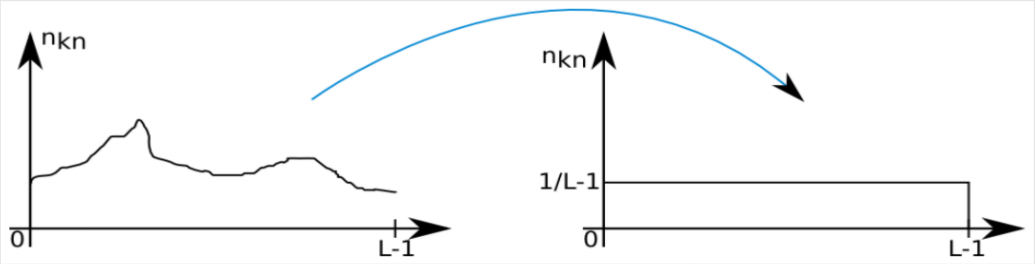
Histogram equalization is a technique for processing images by changing the histogram's intensity distribution in order to change the contrast of an image. The objective of this technique is to give a linear trend to the cumulative probability function associated to the image.

Histogram equalization processing is based on the use of the cumulative probability function (cdf). The cdf is a cumulative sum of all the odds in its domain and is defined

$$cdf(x) = \sum_{k=-\infty}^{x} P(k)$$

by:

The principle of this method is to give a linear cumulative distribution function to the resulting image. In fact, the uniform histogram that we want the resulting image to have is connected to a linear cdf.



Technique to perform histogram equalization

So, to get the latest pdf, we are going to apply the following formula:

$$S_k = (L-1)cdf(x)$$

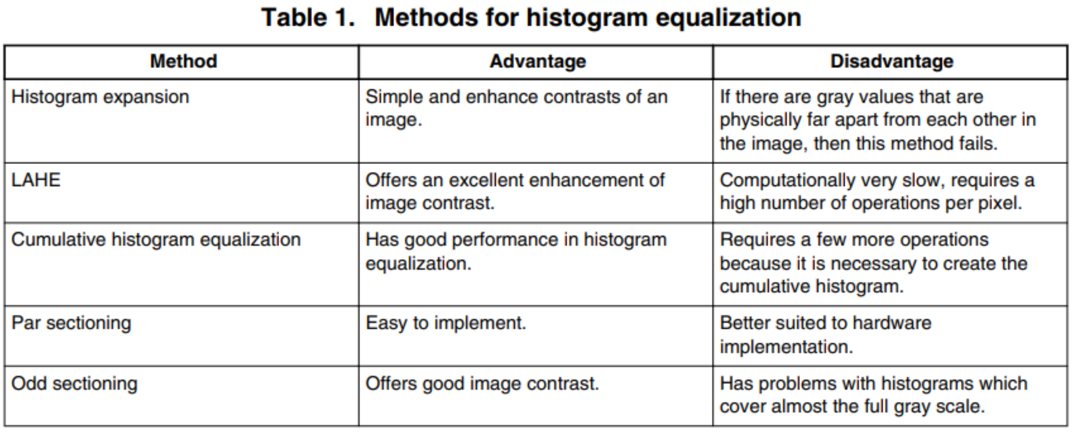


Table 1. Methods for histogram equalization

Method	Advantage	Disadvantage
Histogram expansion	Simple and enhance contrasts of an image.	If there are gray values that are physically far apart from each other in the image, then this method fails.
LAHE	Offers an excellent enhancement of image contrast.	Computationally very slow, requires a high number of operations per pixel.
Cumulative histogram equalization	Has good performance in histogram equalization.	Requires a few more operations because it is necessary to create the cumulative histogram.
Par sectioning	Easy to implement.	Better suited to hardware implementation.
Odd sectioning	Offers good image contrast.	Has problems with histograms which cover almost the full gray scale.

Experimental Purpose:

Histogram Equalization (test images: fig1.jpg, fig2.jpg)

(a)To write a computer program to obtain the histogram of an image.

(b)To implement the algorithm of histogram equalization to achieve image enhancement. (c)To make a comparison between your implementation and the intrinsic functions of MATLAB or OpenCV.

Histogram of an Image:

The histogram of a digital image indicates the distribution of the strength of its pixels. Let X be a (achromatic) image with a gray scale. A digital image's histogram displays the distribution of the power of its pixels. Let X be the (achromatic) gray-scale image.

Generally speaking, Xs is believed to take on the discrete values of 0,.. L-1 where L= 256 is usually used. Then the histogram of picture X is given by

$$h(i) = \sum_{s \in S} \delta(X_s - i) \ .$$

Thus, h(i) measures the amount of pixels that obtain the value i.
In Figure 1, a standard histogram of an 8 bit image is shown.

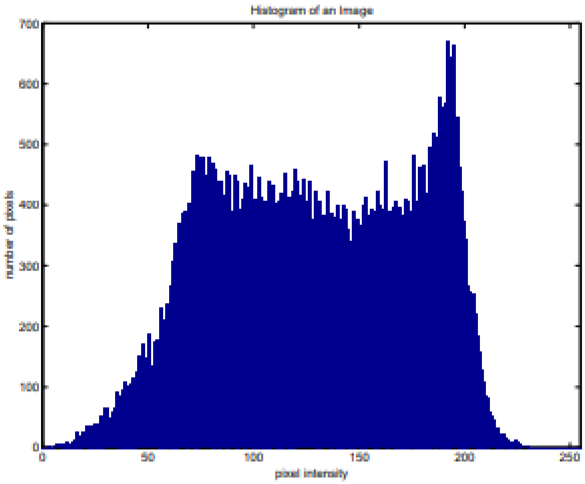
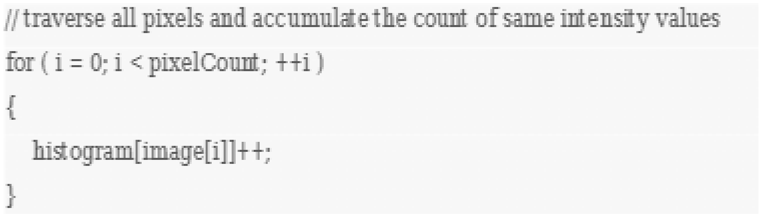


Figure 1: Histogram of an 8 bit image

Experimental Procedure:

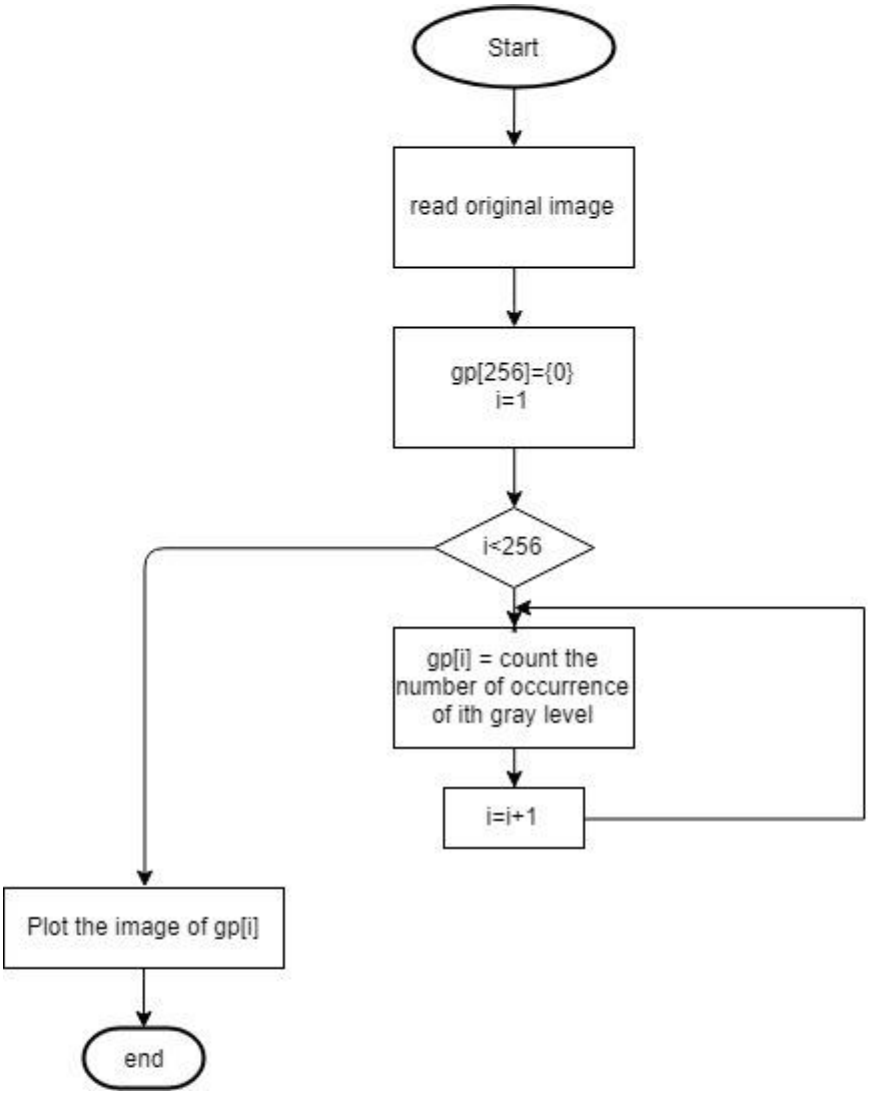
The histogram is a valuable method to evaluate an image's brightness and contrast. This illustrates how an image's intensity values are assigned and the spectrum of brightness from dark to light. By remapping the intensity values using the histogram, an image can be enhanced. A histogram is also use segmentation of an image by thresholding into different regions. For example, if the strength of the image in the histogram is divided into 2 classes, it is possible to pick the threshold value at the center of the 2 peaks of the histogram.
It is very easy to create a histogram of an image. Traverse and count the intensity values for all pixels.[1]



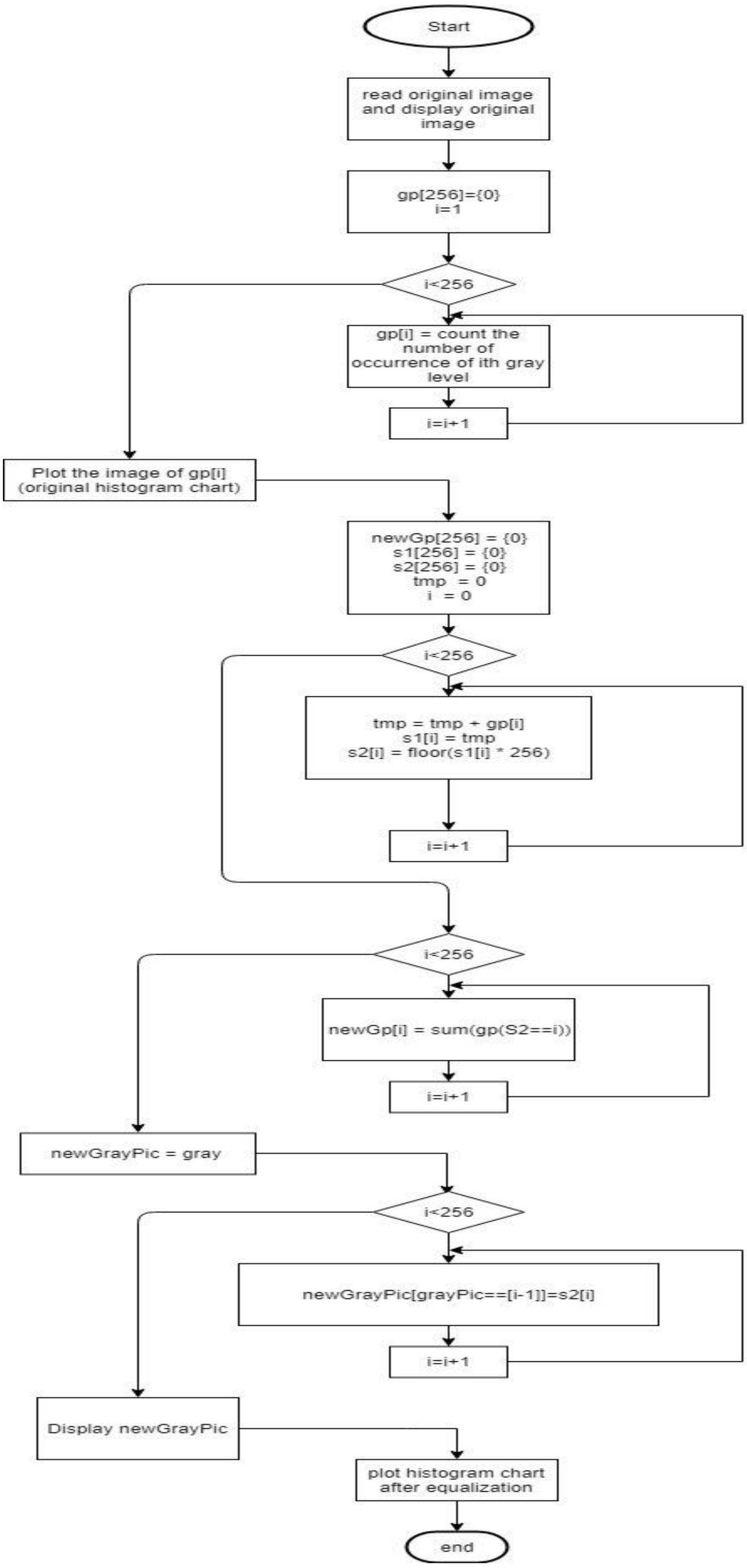
```
//traverse all pixels and accumulate the count of same intensity values
for ( i = 0; i < pixelCount; ++i )
{
   histogram[image[i]]++;
}
```

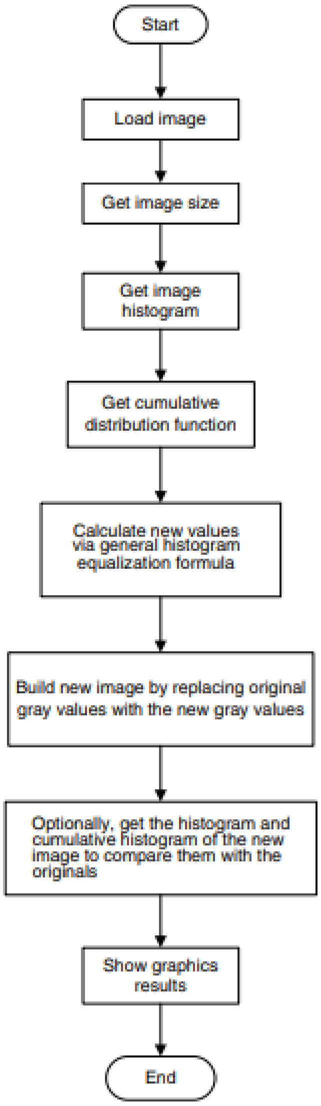
[1] https://www.researchgate.net/publication/283727396_Image_enhancement_by_Histogram_equalization

Therefore, in order to break down the process and also make it simple, it briefly illustrates the method of creating a histogram and histogram equalization implementation using flowcharts.



Histogram Equalization:


Start
read original image and display original image
gp[256]={0}
i=1
i<256
gp[i] = count the number of occurrence of ith gray level
i=i+1
Plot the image of gp[i] (original histogram chart)
newGp[256] = {0}
s1[256] = {0}
s2[256] = {0}
tmp = 0
i = 0
i<256
tmp = tmp + gp[i]
s1[i] = tmp
s2[i] = floor(s1[i] * 256)
i=i+1
i<256
newGp[i] = sum(gp(S2==i))
i=i+1
newGrayPic = gray
i<256
newGrayPic[grayPic==[i-1]]=s2[i]
i=i+1
Display newGrayPic
plot histogram chart after equalization
end


Start
Load image
Get image size
Get image histogram
Get cumulative distribution function
Calculate new values via general histogram equalization formula
Build new image by replacing original gray values with the new gray values
Optionally, get the histogram and cumulative histogram of the new image to compare them with the originals
Show graphics results
End

Block diagram of algorithm implementation [2]

Experimental Result and Analysis

Write a computer program to obtain the histogram of an image.

A discrete function is the histogram of a digital image with gray levels within the range [0, L-1].

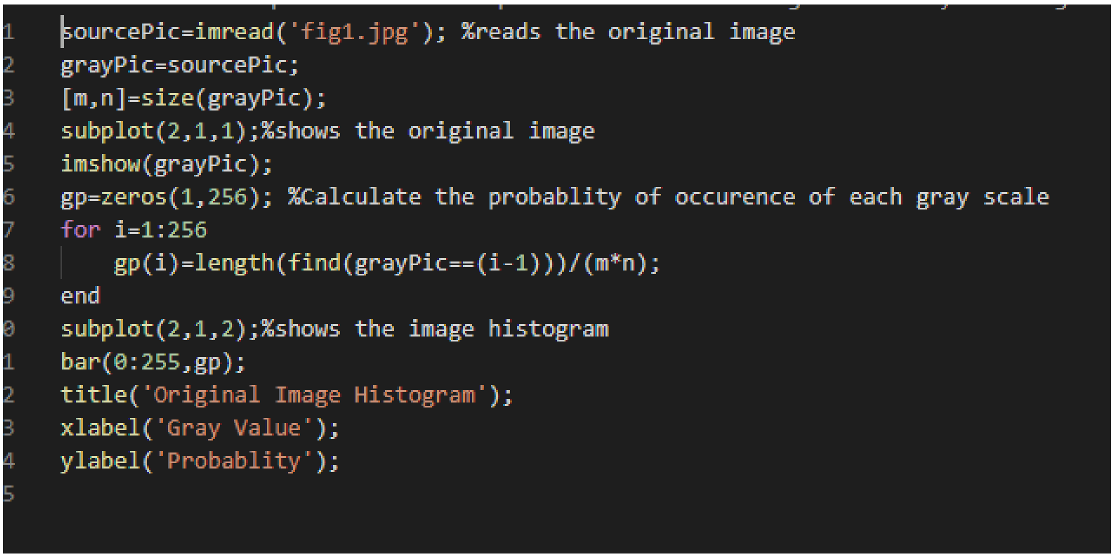
Histogram Function:

$$H(r_k) = n_k$$

[2] https://www.nxp.com/docs/en/application-note/AN4318.pdf

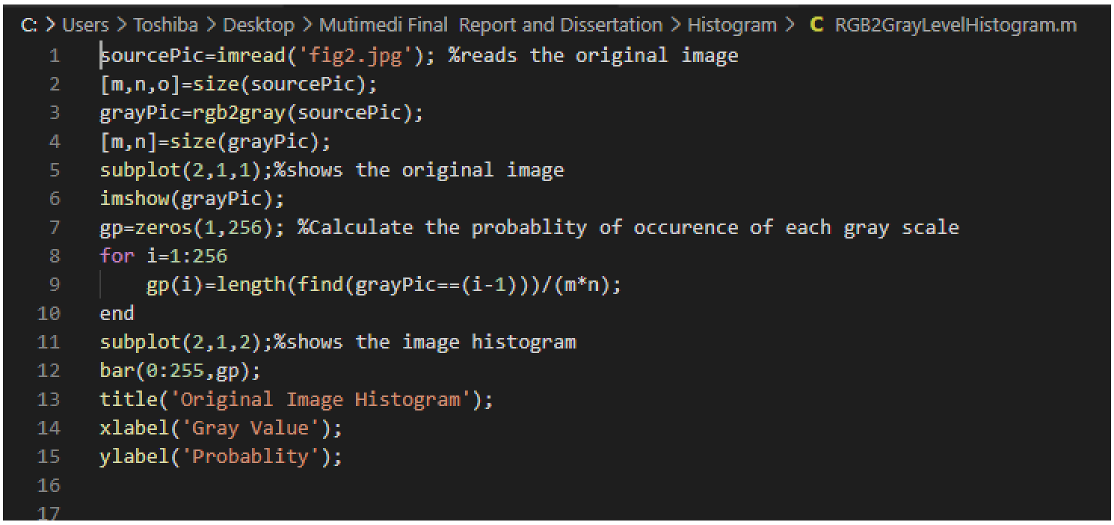
To view the histogram, we have written 2 separate source codes for the color image and the gray level image, and 2 distinct source codes for the color image and the gray level image to display the equalization of the histogram.

<u>Gray Level Image Histogram Source Code:</u>



```
sourcePic=imread('fig1.jpg'); %reads the original image
grayPic=sourcePic;
[m,n]=size(grayPic);
subplot(2,1,1);%shows the original image
imshow(grayPic);
gp=zeros(1,256); %Calculate the probablity of occurence of each gray scale
for i=1:256
    gp(i)=length(find(grayPic==(i-1)))/(m*n);
end
subplot(2,1,2);%shows the image histogram
bar(0:255,gp);
title('Original Image Histogram');
xlabel('Gray Value');
ylabel('Probablity');
```

<u>Color Image Histogram Source Code:</u>



C: > Users > Toshiba > Desktop > Mutimedi Final Report and Dissertation > Histogram > C RGB2GrayLevelHistogram.m

```
sourcePic=imread('fig2.jpg'); %reads the original image
[m,n,o]=size(sourcePic);
grayPic=rgb2gray(sourcePic);
[m,n]=size(grayPic);
subplot(2,1,1);%shows the original image
imshow(grayPic);
gp=zeros(1,256); %Calculate the probablity of occurence of each gray scale
for i=1:256
    gp(i)=length(find(grayPic==(i-1)))/(m*n);
end
subplot(2,1,2);%shows the image histogram
bar(0:255,gp);
title('Original Image Histogram');
xlabel('Gray Value');
ylabel('Probablity');
```

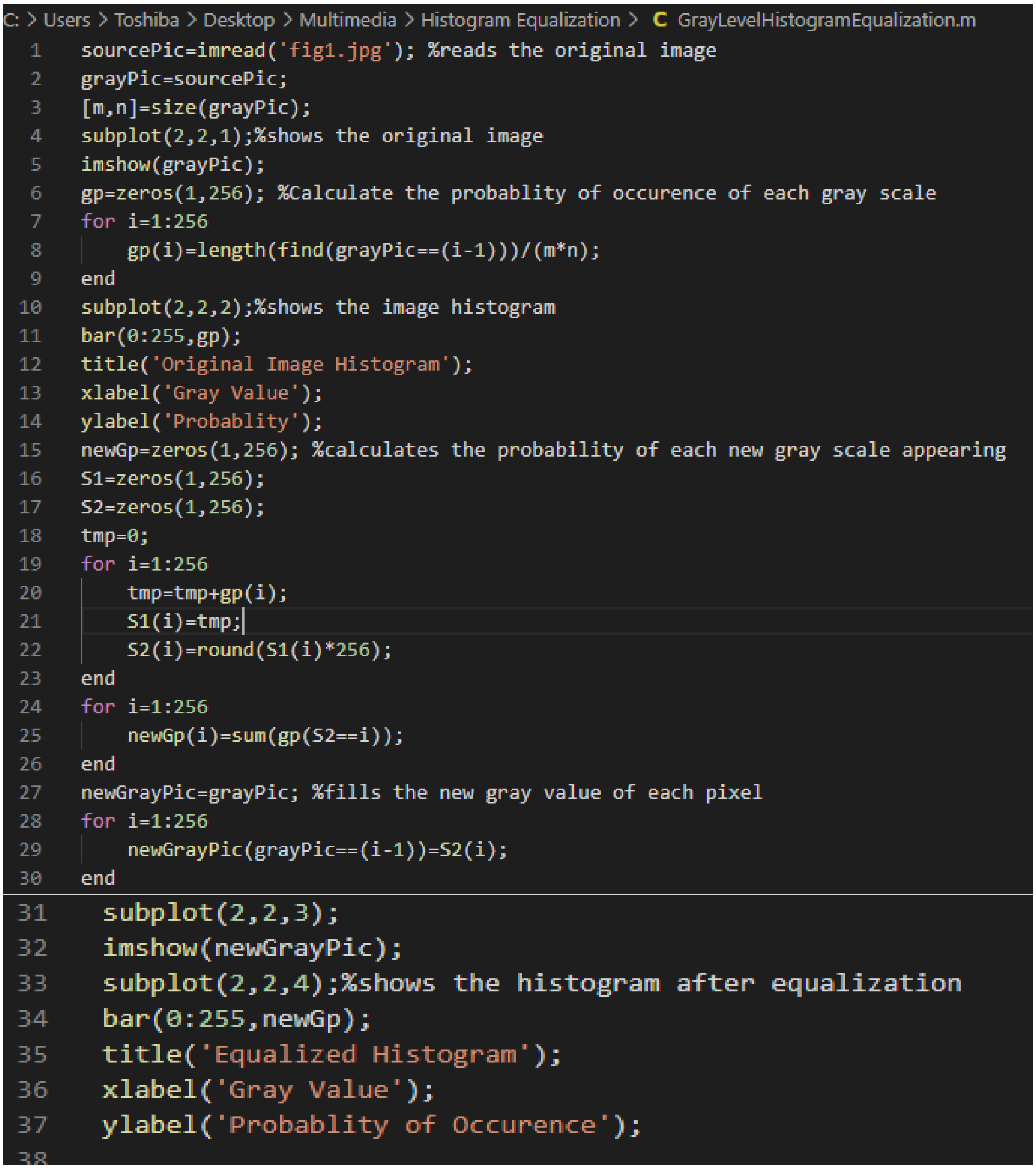
<u>Note:</u>

The need for a distinction in the color and gray level histogram and histogram equalization for the source code is that we need to convert RGB to gray level as we measure and adjust the gray level of the image.

<u>Implement the algorithm of histogram equalization to achieve image enhancement.</u>

For the histogram equalization, the images of the source code are given in the following section; both the color and the gray level images are provided.

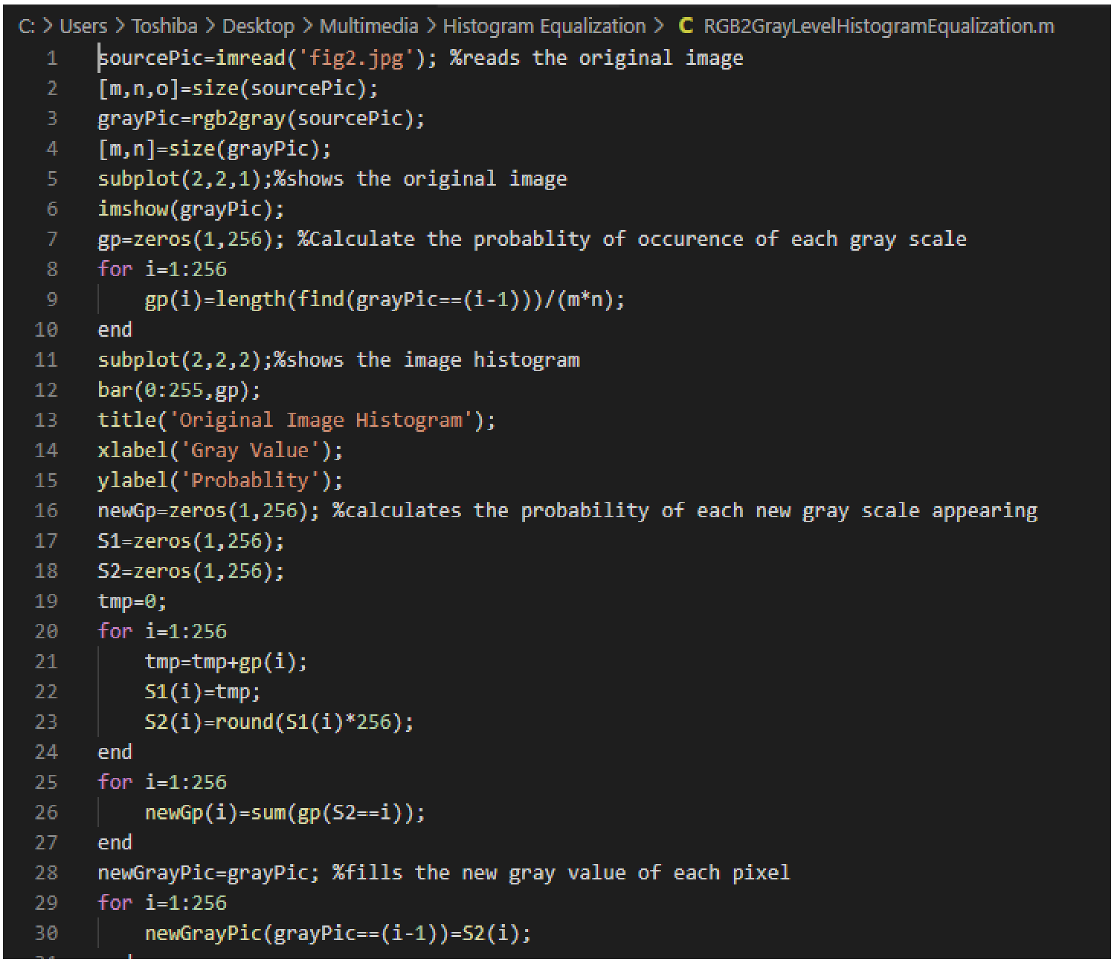
Gray Level Image Histogram Equalization Source Code:



C: > Users > Toshiba > Desktop > Multimedia > Histogram Equalization > GrayLevelHistogramEqualization.m

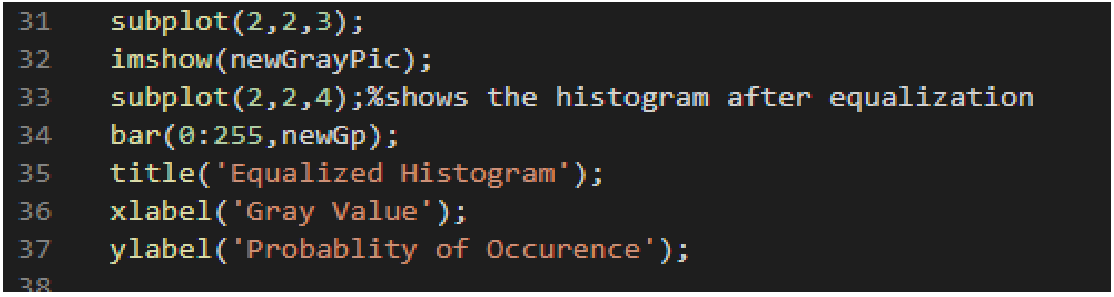
```
sourcePic=imread('fig1.jpg'); %reads the original image
grayPic=sourcePic;
[m,n]=size(grayPic);
subplot(2,2,1);%shows the original image
imshow(grayPic);
gp=zeros(1,256); %Calculate the probablity of occurence of each gray scale
for i=1:256
    gp(i)=length(find(grayPic==(i-1)))/(m*n);
end
subplot(2,2,2);%shows the image histogram
bar(0:255,gp);
title('Original Image Histogram');
xlabel('Gray Value');
ylabel('Probablity');
newGp=zeros(1,256); %calculates the probability of each new gray scale appearing
S1=zeros(1,256);
S2=zeros(1,256);
tmp=0;
for i=1:256
    tmp=tmp+gp(i);
    S1(i)=tmp;
    S2(i)=round(S1(i)*256);
end
for i=1:256
    newGp(i)=sum(gp(S2==i));
end
newGrayPic=grayPic; %fills the new gray value of each pixel
for i=1:256
    newGrayPic(grayPic==(i-1))=S2(i);
end
subplot(2,2,3);
imshow(newGrayPic);
subplot(2,2,4);%shows the histogram after equalization
bar(0:255,newGp);
title('Equalized Histogram');
xlabel('Gray Value');
ylabel('Probablity of Occurence');
```

Color Image Histogram Equalization Source Code:



C: > Users > Toshiba > Desktop > Multimedia > Histogram Equalization > C RGB2GrayLevelHistogramEqualization.m

```
sourcePic=imread('fig2.jpg'); %reads the original image
[m,n,o]=size(sourcePic);
grayPic=rgb2gray(sourcePic);
[m,n]=size(grayPic);
subplot(2,2,1);%shows the original image
imshow(grayPic);
gp=zeros(1,256); %Calculate the probablity of occurence of each gray scale
for i=1:256
    gp(i)=length(find(grayPic==(i-1)))/(m*n);
end
subplot(2,2,2);%shows the image histogram
bar(0:255,gp);
title('Original Image Histogram');
xlabel('Gray Value');
ylabel('Probablity');
newGp=zeros(1,256); %calculates the probability of each new gray scale appearing
S1=zeros(1,256);
S2=zeros(1,256);
tmp=0;
for i=1:256
    tmp=tmp+gp(i);
    S1(i)=tmp;
    S2(i)=round(S1(i)*256);
end
for i=1:256
    newGp(i)=sum(gp(S2==i));
end
newGrayPic=grayPic; %fills the new gray value of each pixel
for i=1:256
    newGrayPic(grayPic==(i-1))=S2(i);
```



```
subplot(2,2,3);
imshow(newGrayPic);
subplot(2,2,4);%shows the histogram after equalization
bar(0:255,newGp);
title('Equalized Histogram');
xlabel('Gray Value');
ylabel('Probablity of Occurence');
```

Results:

Product of histogram for gray level and color image respectively:

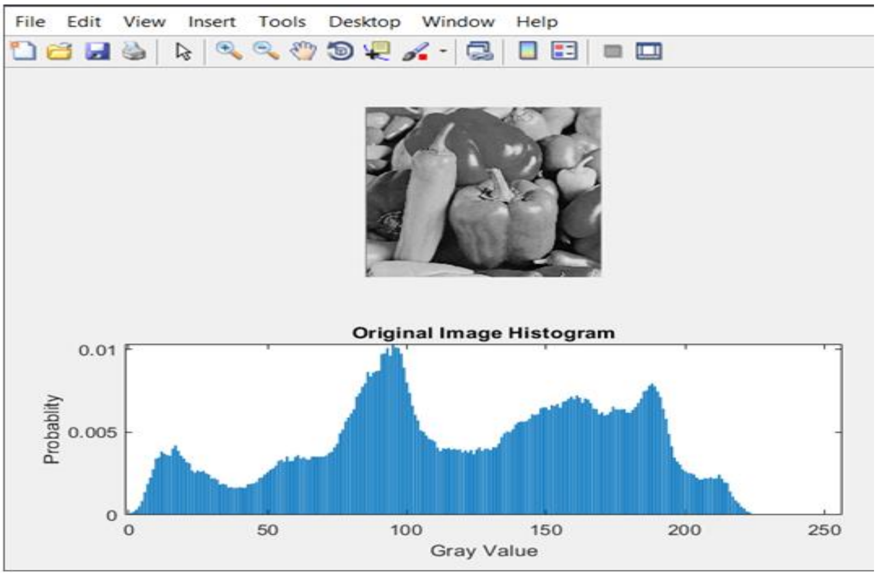


Figure 1.

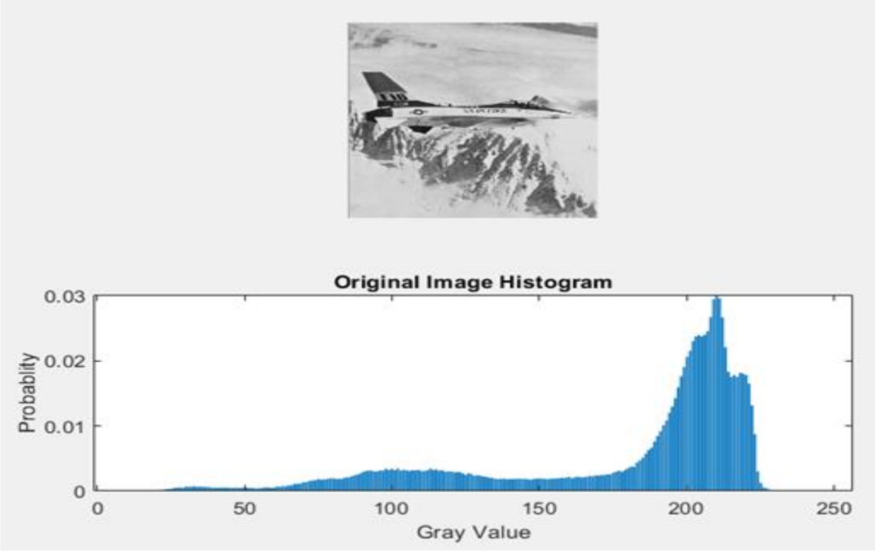


Figure 2.

Histogram Equalization for gray level and color image respectively:

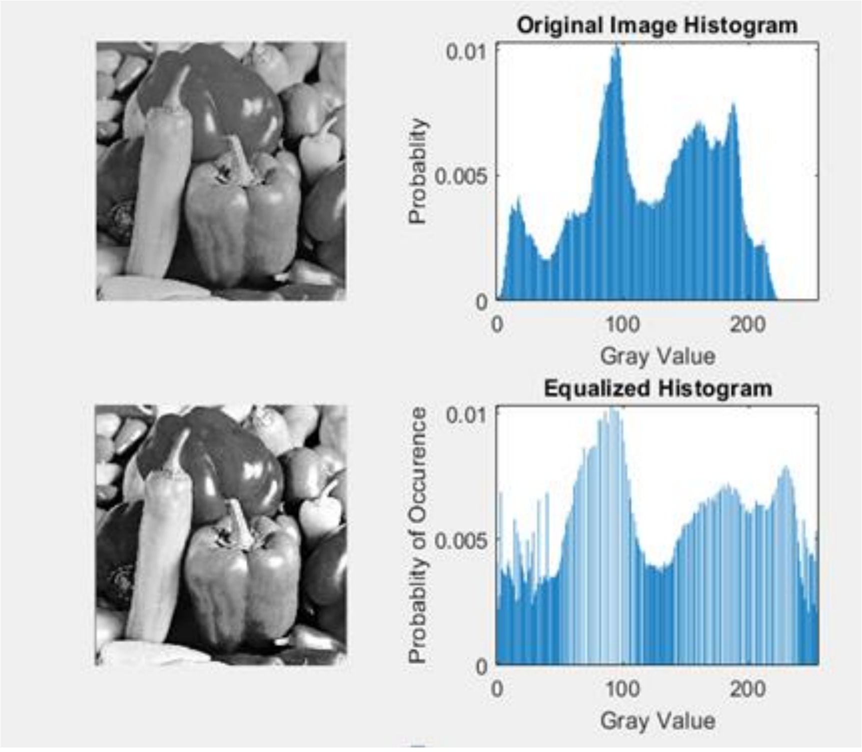


Figure 1.

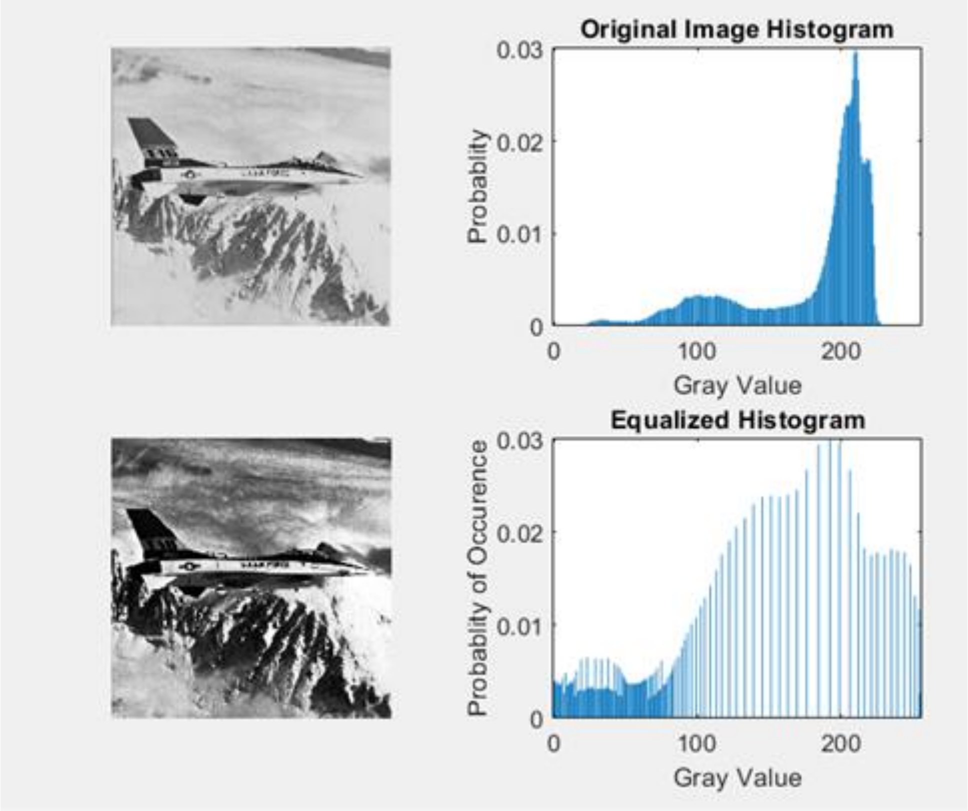


Figure 2:

Analysis:

The gray level is scattered around the picture for the image histogram in figure1, which makes the image dull.

The gray level in the region is roughly between 175 and 225 for the image histogram in Figure 2, making the image appear brighter.

We will note from the results shown for histogram equalization that the initial histogram has been extended.

Gray values in the region [50-100] and [150-225] were stretched in Figure 1 and created a clear image.

Gray values were extended in the region [200-250] and [175-225] in figure 2, which made the image appear darker, but more details remained.

(c)Make a comparison between your implementation and the intrinsic functions of MATLAB or OpenCV

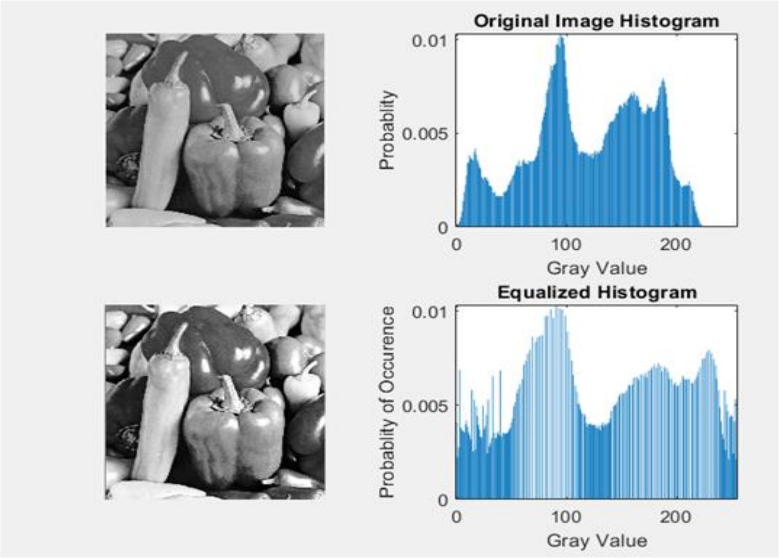


Figure 1.

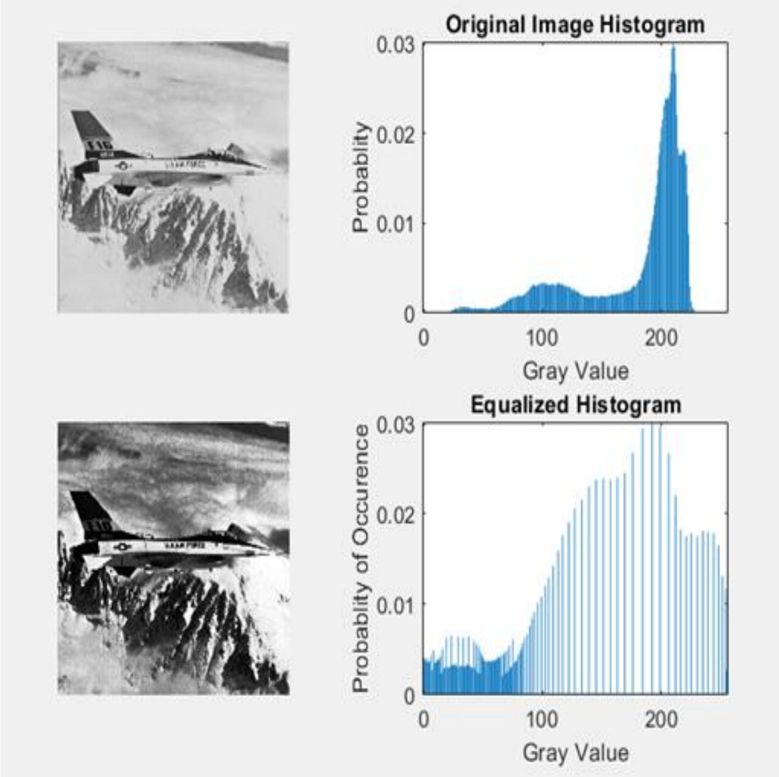
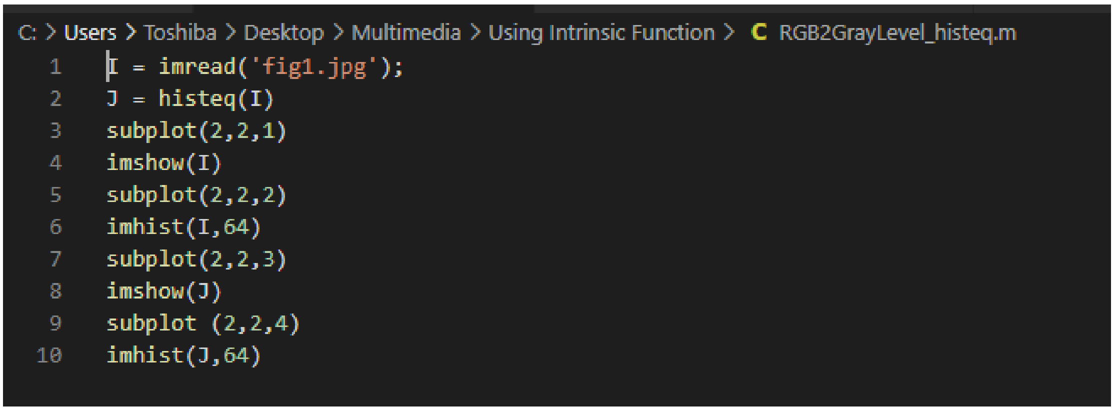


Figure 2.

Code of equalization of the histogram and histogram using the intrinsic function:


C: > Users > Toshiba > Desktop > Multimedia > Using Intrinsic Function > C RGB2GrayLevel_histeq.m

```
I = imread('fig1.jpg');
J = histeq(I)
subplot(2,2,1)
imshow(I)
subplot(2,2,2)
imhist(I,64)
subplot(2,2,3)
imshow(J)
subplot (2,2,4)
imhist(J,64)
```

Overview:

When contrasting the implementation with the intrinsic function’s implementation, there are some points we should consider. In this source code, much of the calculation, such as calculating the likelihood of each gray level and the histogram equalization function, is performed in the foreground process.

In order to process more lines of code, this implementation often takes more space and time. In manipulations we can add in my graph such as the range of bins and automatic assignment of comparison of multiple sets of plots, the execution is also constrained.

The intrinsic function has a separate algorithm to process the equalization of the histogram. The number and range of the bins to be suitable for the data is also automatically set. It also elegantly contrasts several data sets on one or more plots, with legends or names. It graphs the mean and standard deviations as well. It can plot the median and mode as well.

The histogram and the equalization are pre-calculated by the intrinsic function, so displaying the result takes less time. For the equalization of the histogram, both implementations generated distinct results, but a very similar picture.

The rationale behind this is that, as seen in the flow chart, my execution used minimal procedures and the intrinsic function used a more complicated function capable of computing large floating-point numbers.

Conclusion:

From the first question, we learned how the likelihood of occurrence of each gray level can be determined in the range of [0-255]. We also added a feature to extend the histogram from question (b) to give the picture greater contrast and clarity. And we used the built-in function in question (c) to do the histogram and the equalization of the histogram and compared them to our prior implementation.

This approach is very useful for images of backgrounds and foregrounds that are too light or too dark. In X-ray images, especially for better display of the bone structure and better detail in overexposed or underexposed pictures. A significant benefit of this strategy is that it is a reasonably straightforward process and is reversible. If the function for equalization is known, we may restore the original histogram and the computational quantity is not high. One downside of this method is that it does not pick the processed data; it can increase the background noise contrast and decrease the useful signal contrast.

In order to determine the relationship between the original image and the gray value of the new image, histogram equalization primarily uses the concept of statistics and eventually equalizes the gray level of the image.
The effect of the second image is stronger, from the final result, and the first image's contrast is improved. However, at the same time, it leads to a lot of salt and pepper noise.

References:

https://www.nxp.com/docs/en/application-note/AN4318.pdf
https://www.researchgate.net/publication/283727396_Image_enhancement_by_Histogram_equalization

https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/graphs/histogram/interpret-the-results/key-results/
https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm
https://veprit.com/photography-guide/using-histogram/what-is-image-histogram

**Experir
Multimedia**

Question No.2

Discrete Cosine Transform

Introduction:

The discrete cosine transform (DCT) helps to divide the image into sections of varying significance (with respect to the optical consistency of the image) (or spectral sub-bands).The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain.(Fig 1.1)

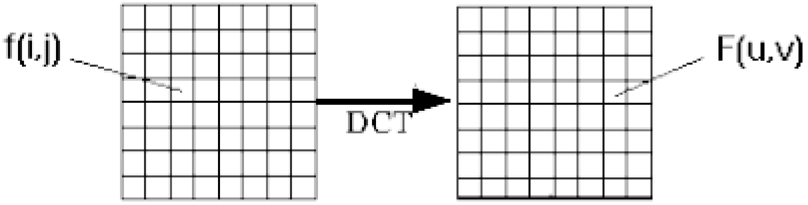


Fig 2.1

If our use of and dependency on computers continues to expand, so does our need for effective ways to store vast volumes of knowledge. Someone with a web page or online catalog, for example, who uses dozens or maybe hundreds of images, would probably have to use some sort of image compression to store those images. This is because broad cost terms will prohibit the amount of space needed to store unadulterated pictures. Fortunately, many image compression techniques are available today. Both fell into two general categories: today's usable lossless and lossy file compression. This fall into two general categories: compression of the lossless and lossy image. A commonly used method of lossy image compression is the JPEG technique.

The JPEG approach is a commonly used type of compression of lossy images that centers on the Discrete Cosine Transform. By splitting images into sections of varying wavelengths, the DCT runs. The less significant frequencies are discarded during a process called quantization, where part of compression actually happens, therefore the usage of the "lossy" term. In the decompression process, only the most significant frequencies that remain are then used to recover the file.

As a result, restored images involve some distortion, but during the compression process, these degrees of distortion can be modified. For both color and black-and-white images, the JPEG approach is used but the focus of this article would be on the compression of the latter.

Process:

A general description of the method of JPEG is as follows. Later, in order to achieve a comprehensive understanding of the technique, we will take the reader on a thorough tour of the JPEG system.
The image is divided into 8×8 blocks of pixels.
2. Working from left to right, top to bottom, each block is added to the DCT.
3. By quantization, each block is compressed.
In a dramatically reduced amount of space, the sequence of compact blocks that form the image is processed.
The image is recovered by decompression as desired, a procedure that utilizes the Inverse Discrete Cosine Transform (IDCT).

The DCT Equation:

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N} \right] \cos\left[\frac{(2y+1)j\pi}{2N} \right]$$

$$C(u) = \left\{ \begin{array}{ll} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{array} \right\}$$

DCT Encoding:

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

$$F(\mathrm{u}) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} A(i) \cos\left[\frac{\pi \cdot u}{2 \cdot N} (2i+1) \right] f(i)$$

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

$$A(i) = \begin{cases} \frac{1}{\sqrt{2}} & for\xi = 0 \\ 1 & otherwise \end{cases}$$

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

$$F(\mathrm{u,v}) = (\frac{2}{N})^{\frac{1}{2}} (\frac{2}{M})^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} A(i) \cdot A(j) \cdot \cos\left[\frac{\pi \cdot u}{2 \cdot N}(2i+1)\right] \cos\left[\frac{\pi \cdot u}{2 \cdot M}(2j+1)\right] f(i,j)$$

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

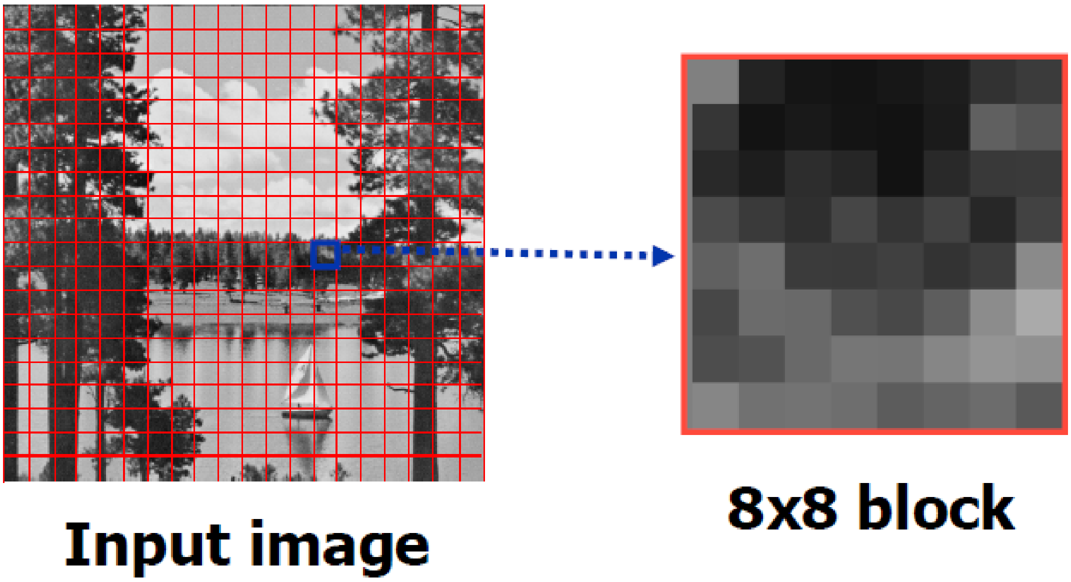
$$A(i) = \begin{cases} \frac{1}{\sqrt{2}} & for\xi = 0 \\ 1 & otherwise \end{cases}$$

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.
-The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
-8 bit pixels have levels from 0 to 255.
-Therefore an 8 point DCT would be:

<u>JPEG compression is a lossy compression format for images. Here is a simple explanation of how it works.</u>

Step 1:

The image is divided into non-overlapping 8 by 8 blocks. If the image width or size does not divide evenly into 8, the image may be cropped or pixels may be added to make the image divisible by 8. Each 8 by 8 block will contain 64 values. For a grayscale image, each pixel may be anywhere from 0 to 255, where 0 is pure black and 255 is pure white.



$$\begin{bmatrix}
52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\
63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\
62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\
63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\
67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\
79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\
85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\
87 & 79 & 69 & 68 & 65 & 76 & 78 & 94
\end{bmatrix}$$

A sample 8 by 8 block for a grayscale image

Step 2:

The 2-D Discrete Cosine Transform (DCT) is applied to each 8 by 8 block. For the purposes of applying the DCT, the values in each block are first shifted by -128 to center around zero.

$$\overset{\xrightarrow{\;x\;}}{\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}} \;\Big\downarrow y$$

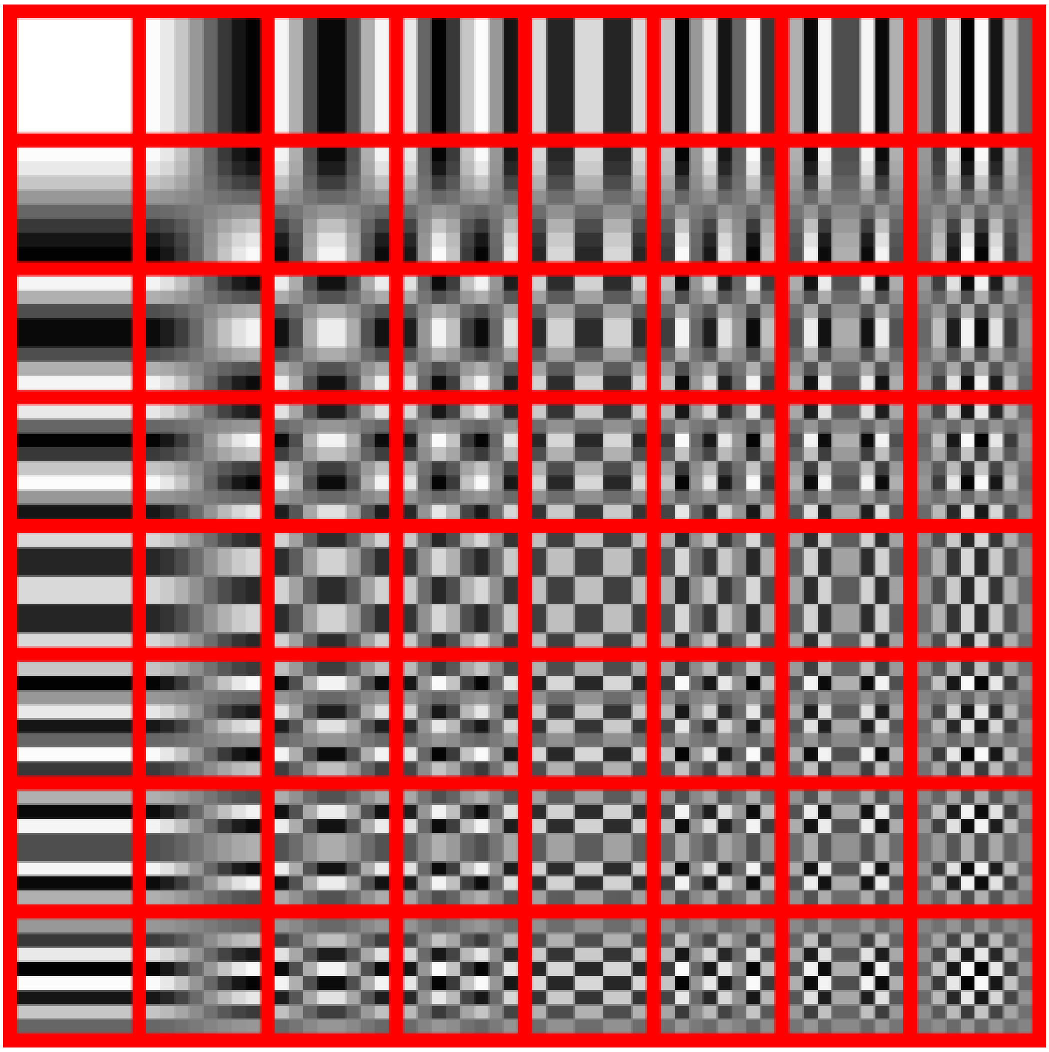
Shift the image by -128 to center pixel values around 0

$$\overset{\xrightarrow{\;u\;}}{\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}} \;\Big\downarrow v$$

2-D DCT of the zero-centered 8x8 block

2-D DCT of the zero-centered 8x8 block

The DCT is selected over transforms such as the FFT, since only real numbers represent the frequency coefficients. Two 8 by 8 matrices will be needed by the FFT, one to store the true part of the frequency coefficient, another to store the imaginary part. The outcome of the DCT is also an 8 by 8 block, where the DC or average pixel value for that particular 8 by 8 block is expressed by the top left corner. An AC coefficient is the lower left corner that represents the fastest vertical transition. An AC coefficient is the top-right corner that represents the fastest horizontal transition. The picture is represented as a weighted sum of the 64 configurations that are possible.



Grayscale representation of the AC components of the 2-D DCT

Step 3:

Each block is quantized using a quantization matrix until the DCT of the 8 by 8 block is obtained. This is where the JPEG format's lossiness is implemented. Quantization is basically an operation of divide and floor. The consistency factor of a JPEG is directly connected to the matrix of quantization.

An 8 by 8 matrix is the quantization matrix. Based on the Q-factor, values inside this matrix are chosen. The q-matrix can be set to all 1s for a high-quality image. This basically implies that no quantization happens. The way the values are determined in the q-matrix is based on a picture's popular patterns. Pictures usually involve a lot of continuous tones. This makes the image's DC component and the low AC components particularly dominant in the image representation. The majority of the image intensity is found in the top left corner of the 2-D DCT in a typical image. The q-matrix values for the top-left corner are smaller in order to conserve these portions of the image. In the lower right corner of the q-matrix, the values are especially large. This is because, with only the top left portion of the 2-D DCT, the picture is already well characterized. Unless it is especially dominant and necessary for accurate representation, we do not want to use a high frequency coefficient. The values in the lower right part of the q-matrix are therefore large. Unless they have high signal energy (aka, they are important in the reconstruction and representation of the image), you can think of this as attenuating high-frequency portions of the image.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

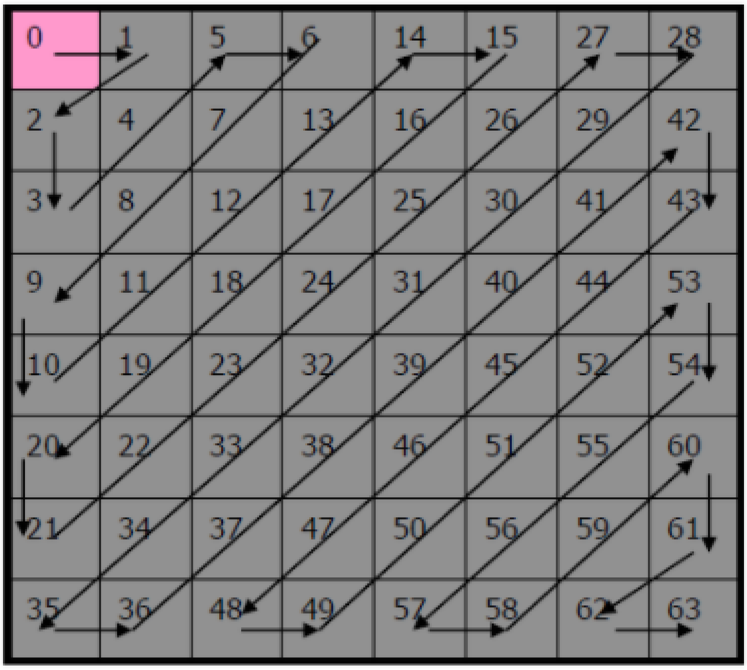
Sample quantization matrix:

Normally, you will then multiply the quantized value by the same q value until you have quantized it by a certain value. So imagine you had 125 and you were trying to measure by 16. 16*floor (125/16) will be the quantized value. This process is saved for later, however (I mean decompression of the image by later). Since multiplying by the q-matrix we want to encode what we have now would inevitably cause all the numbers to get bigger. Smaller values are often better when encoding content. Below is the quantized DCT without the q-matrix multiplication of element-by-element.

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 4:

The last step to perform is entropy encoding. The quantized DCT coefficients are arranged in the following manner:



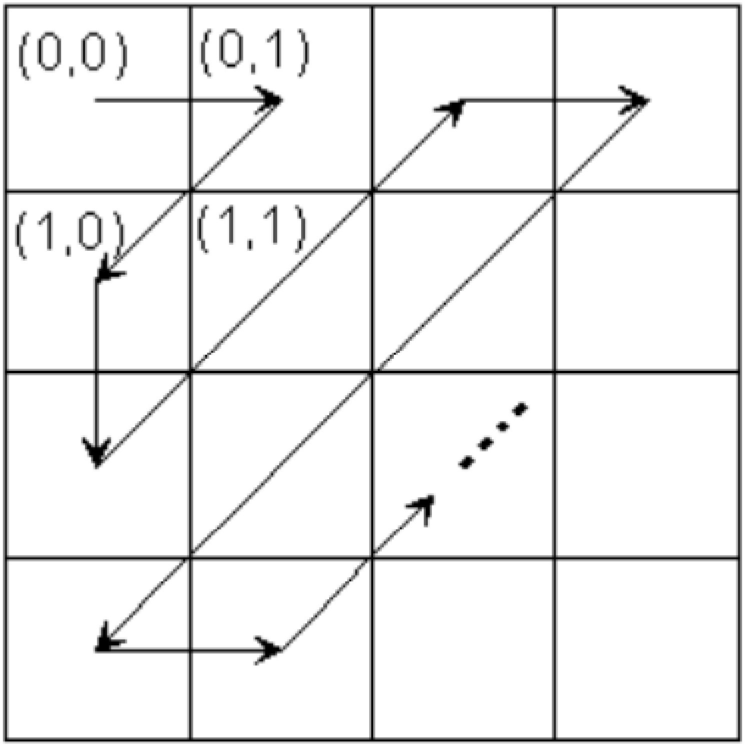
0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Until encoding, how DCT coefficients are ordered:

They are ordered this way since the quantized DCT's top left is where most non-zero values lie. There will also be mostly zeroes in the bottom right. Different encoding algorithms (including Huffman encoding that uses common patterns in the DC values to predict the DC of neighboring blocks) are implemented once the numbers in the matrix are re-arranged to minimize the amount of space each block requires.

Coding:

The quantized matrix C is now ready for the final stage of compression. Until storage, all coefficients of C are translated by an encoder to a stream of binary data (01101011….). In-depth coverage of the method of coding is beyond the scope of this paper. We should however, point out one crucial element that the reader is likely to appreciate. It is very normal for most of the coefficients to equal zero after quantization. By encoding quantized coefficients in the Zigzag sequence shown in the figure below, JPEG takes advantage of this. The profit lies in consolidating relatively large zero runs that compress very well. For the whole 8 to 8 block, the sequence is Figure 1(4 to 4).



Decompression:

The file is decoded first when opening a JPEG image. The quantized DCT is the outcome of the decoded paper. Multiplying the quantization matrix used to quantize the image is the first step. Then the Inverse Discrete Transform of Cosine (IDCT) is used. Finally, for all 64 pixels, the outcome of the IDCT is changed by 128. The result is the image, with the loss of information depending on the matrix of quantization.

$$R_{i,j} = Q_{i,j} \times C_{i,j}$$

$$R = \begin{bmatrix} 160 & 44 & 20 & 80 & 24 & 0 & 0 & 0 \\ 36 & 108 & 14 & 38 & 26 & 0 & 0 & 0 \\ -98 & -65 & 16 & -48 & -40 & 0 & 0 & 0 \\ -42 & -85 & 0 & -29 & 0 & 0 & 0 & 0 \\ -36 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Next the IDCT is added to matrix R, rounded to the nearest integer. Finally, 128 is added to each of the elements of that result, giving us JPEG version N of our original 8 to 8 picture block M decompressed. $N = round(T' \; R \; T) + 128$

Discrete Cosine Transform(test images: fig1.jpg)

(a) Perform the DCT on whole image and each 8*8 block of the image respectively.
(b) Change the number of preserved harmonics(variable f), make the value the AC components to zero and assign a part of DC components to zero.
(c) Apply Inverse-DCT to restore the image. Observe how well an image can be approximated with different number of DCT coefficients.
(d) Think about the difference when applying DCT over the whole image and the image in 8*8 block.

Comparison Of Matrices:

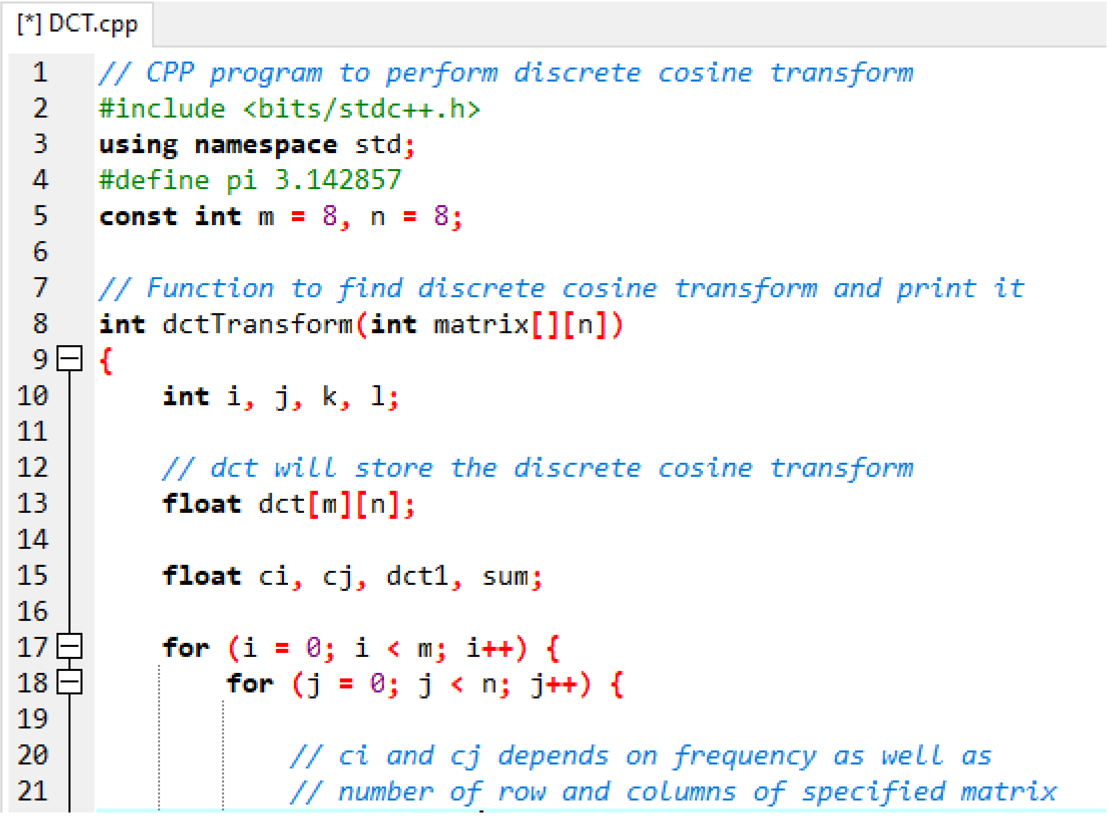
Let us see how our original pixel block JPEG version compares:

$$Original = \begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 136 & 154 & 136 \end{bmatrix}$$

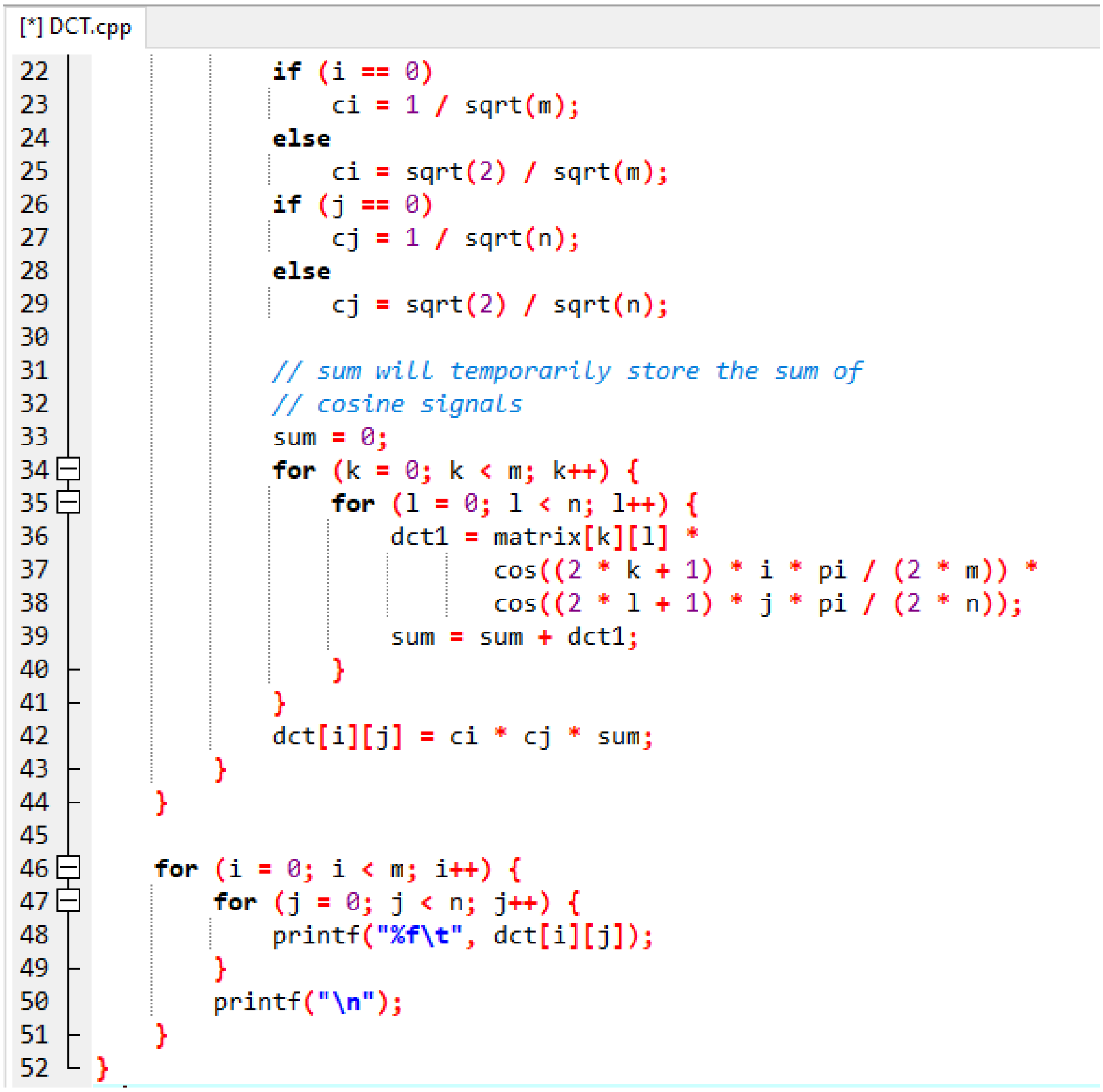
$$Decompressed = \begin{bmatrix} 149 & 134 & 119 & 116 & 121 & 126 & 127 & 128 \\ 204 & 168 & 140 & 144 & 155 & 150 & 135 & 125 \\ 253 & 195 & 155 & 166 & 183 & 165 & 131 & 111 \\ 245 & 185 & 148 & 166 & 184 & 160 & 124 & 107 \\ 188 & 149 & 132 & 155 & 172 & 159 & 141 & 136 \\ 132 & 123 & 125 & 143 & 160 & 166 & 168 & 171 \\ 109 & 119 & 126 & 128 & 139 & 158 & 168 & 166 \\ 111 & 127 & 127 & 114 & 118 & 141 & 147 & 135 \end{bmatrix}$$

This is a remarkable finding, given that almost 70 percent of the DCT coefficients were discarded before decompression of the picture block. Provided that the remaining blocks that make up the whole image will yield identical effects, it should not be shocking that the JPEG image will not be distinguishable from the original one. Know, in a black-and-white photo, there are 256 possible shades of gray, and a variation of, say, 10 is scarcely visible to the human eye.

Codes:

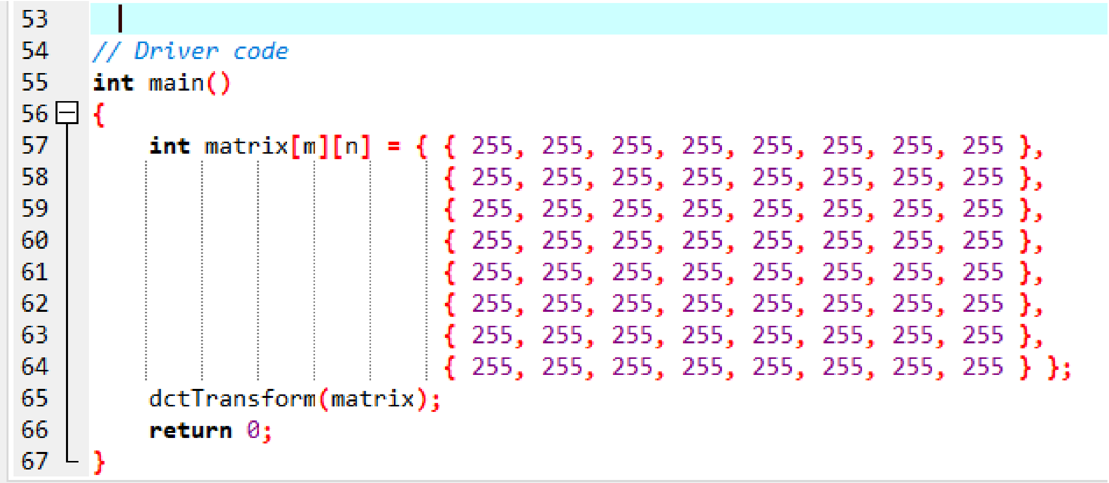


```cpp
// CPP program to perform discrete cosine transform
#include <bits/stdc++.h>
using namespace std;
#define pi 3.142857
const int m = 8, n = 8;

// Function to find discrete cosine transform and print it
int dctTransform(int matrix[][n])
{
    int i, j, k, l;

    // dct will store the discrete cosine transform
    float dct[m][n];

    float ci, cj, dct1, sum;

    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {

            // ci and cj depends on frequency as well as
            // number of row and columns of specified matrix
```

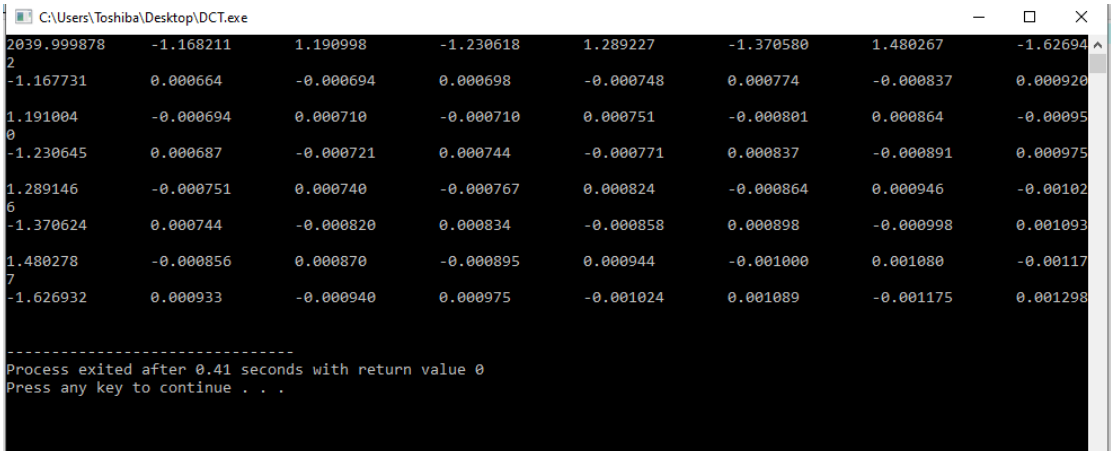


[*] DCT.cpp

```
            if (i == 0)
                ci = 1 / sqrt(m);
            else
                ci = sqrt(2) / sqrt(m);
            if (j == 0)
                cj = 1 / sqrt(n);
            else
                cj = sqrt(2) / sqrt(n);

            // sum will temporarily store the sum of
            // cosine signals
            sum = 0;
            for (k = 0; k < m; k++) {
                for (l = 0; l < n; l++) {
                    dct1 = matrix[k][l] *
                           cos((2 * k + 1) * i * pi / (2 * m)) *
                           cos((2 * l + 1) * j * pi / (2 * n));
                    sum = sum + dct1;
                }
            }
            dct[i][j] = ci * cj * sum;
        }
    }

    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            printf("%f\t", dct[i][j]);
        }
        printf("\n");
    }
}
```



```

// Driver code
int main()
{
    int matrix[m][n] = { { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 },
                         { 255, 255, 255, 255, 255, 255, 255, 255 } };
    dctTransform(matrix);
    return 0;
}
```



C:\Users\Toshiba\Desktop\DCT.exe

```
2039.999878     -1.168211       1.190998        -1.230618       1.289227        -1.370580       1.480267        -1.62694
2
-1.167731       0.000664        -0.000694       0.000698        -0.000748       0.000774        -0.000837       0.000920

1.191004        -0.000694       0.000710        -0.000710       0.000751        -0.000801       0.000864        -0.00095
0
-1.230645       0.000687        -0.000721       0.000744        -0.000771       0.000837        -0.000891       0.000975

1.289146        -0.000751       0.000740        -0.000767       0.000824        -0.000864       0.000946        -0.00102
6
-1.370624       0.000744        -0.000820       0.000834        -0.000858       0.000898        -0.000998       0.001093

1.480278        -0.000856       0.000870        -0.000895       0.000944        -0.001000       0.001080        -0.00117
7
-1.626932       0.000933        -0.000940       0.000975        -0.001024       0.001089        -0.001175       0.001298


--------------------------------
Process exited after 0.41 seconds with return value 0
Press any key to continue . . .
```

Results and Analysis:

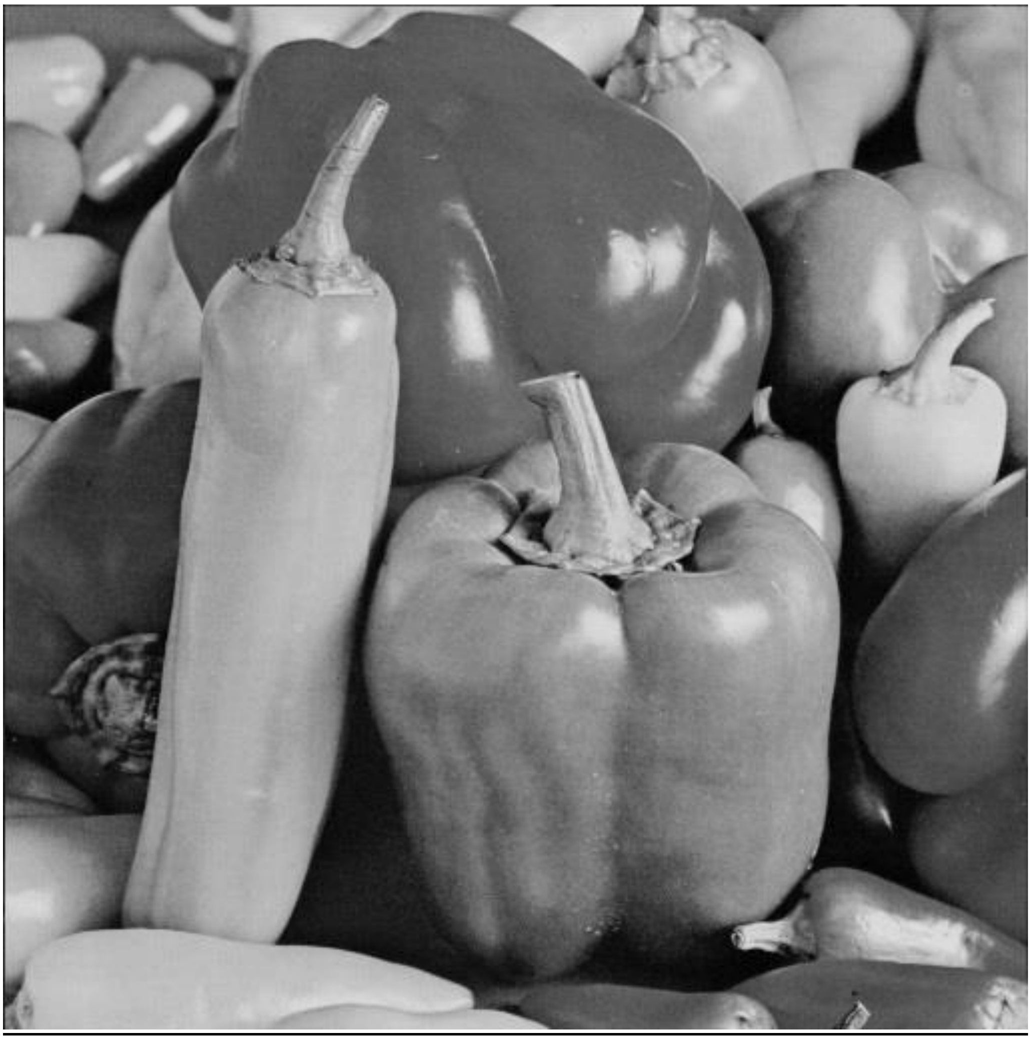


Fig.1.

The DCT is struck every eight by eight blocks, resulting in the picture shown below:

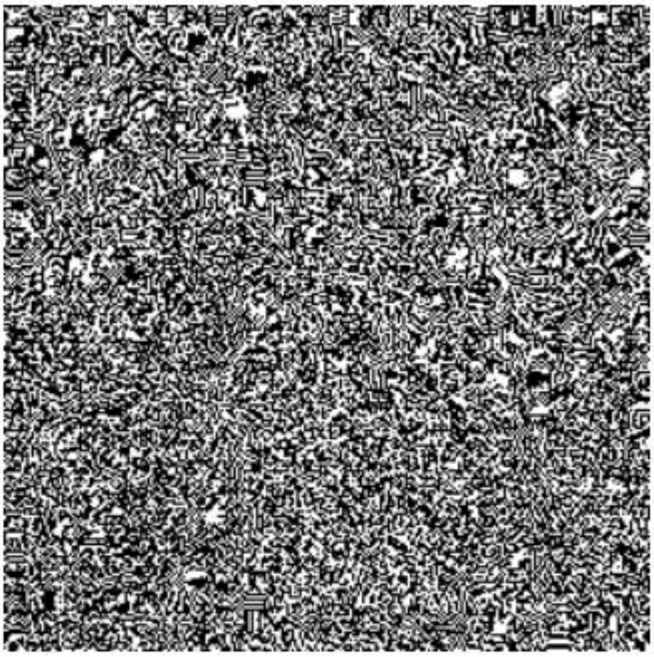


Fig: DCT of Fig.1.

Each entity is then quantized using a consistency level 50 quantization matrix in each block of the image. Many of the elements are zeroed out at this point, and the picture takes up much less storage space.

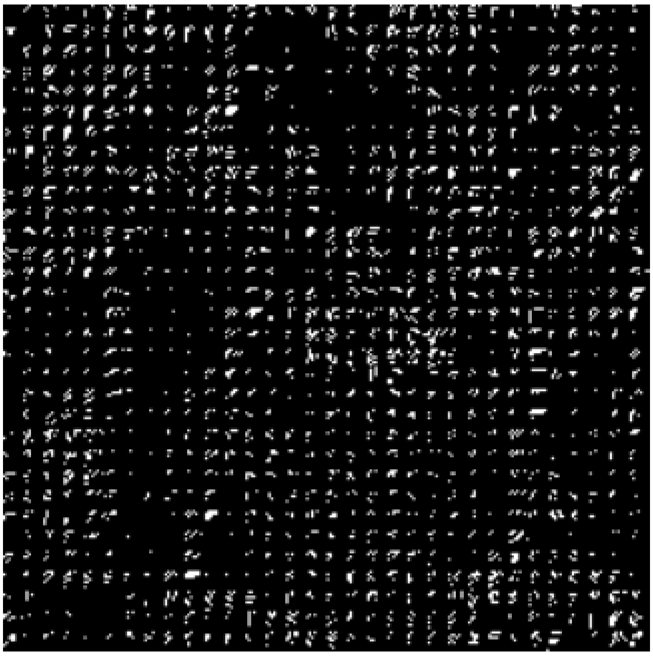
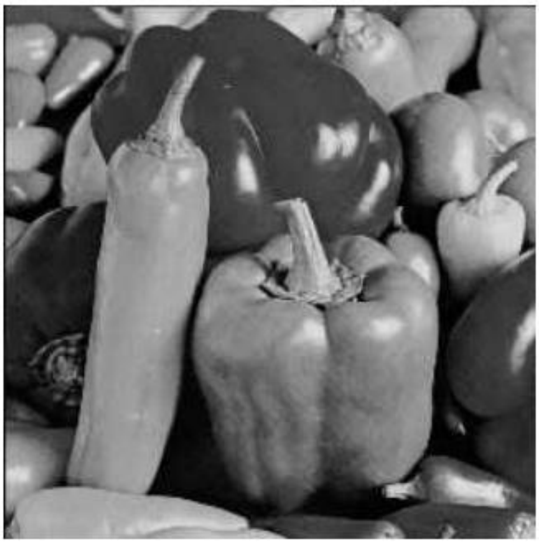


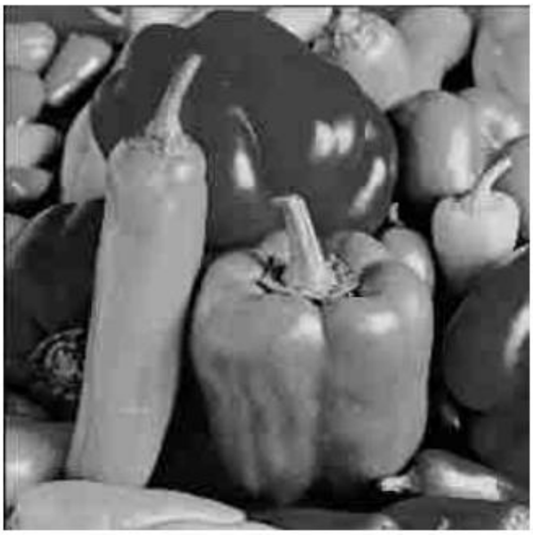
Fig: Quantized DCT of Fig.1.

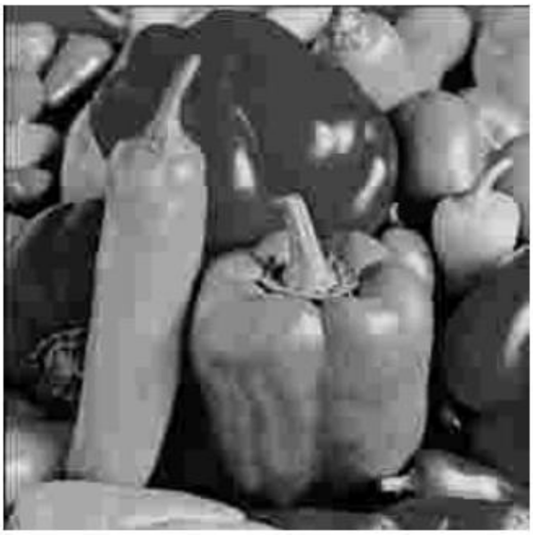
Using the inverse discrete cosine transform, the image can now be decompressed. There is almost no apparent loss in this picture at quality level 50, but heavy compression is present. The consistency goes down by a lot at lower quality values, but the compression does not improve too much.



Original Fig.1.

Quality 50-84% Zeros.





Quality 20-90% Zeros.

Quality 10-94% Zeros

Overview:

Discrete Cosine Transform is used in lossy image compression because it has very powerful energy compaction, i.e., its large amount of information is stored in very low frequency portion of a signal and rest other frequency having very limited data which can be stored by using very less number of bits (usually, at most 2 or 3 bit).

We first need to get image file information (pixel value in terms of integer with range 0-255) to perform DCT transformation on an image, which we break into 8 X 8 matrix block, and then we apply discrete cosine transformation on that data block.

After applying discrete cosine transformation, we can see that the lower frequency portion will be more than 90 percent details. We took a matrix of size 8 X 8 with all value as 255 for simplicity (considering the image to be entirely white) and we will transform 2-D discrete cosine on that to observe the output.

References:

https://www.projectrhea.org/rhea/index.php/Homework3ECE438JPEG
http://www.iosrjournals.org/iosr-jece/papers/Vol5-Issue4/H0545156.pdf?id=4310
https://arxiv.org/ftp/arxiv/papers/1405/1405.6147.pdf
https://ieeexplore.ieee.org/document/125072
https://www.geeksforgeeks.org/discrete-cosine-transform-algorithm-program/
https://www.researchgate.net/publication/220362455_Fast_2-D_88_discrete_cosine_transform_algorithm_for_image_coding