JPEG Image Compression Implemented in Matlab

Sarthak Chaudhary (101412034)

Rubal Bansal (101412031)

**Abstract:**

In this project WE attempted to implement basic JPEG compression using only basic Matlab functions. This included going from a basic grayscale bitmap image all the way to a fully encoded file readable by standard image readers.

# Encoding

# Colour space transformation

First, the image should be converted from RGB into a different color space called YCbCr. It has three components Y, Cb and Cr: the Y component represents the brightness of a pixel, the Cb and Cr components represent the chrominance (split into blue and red components). The YCbCr color space conversion allows greater compression without a significant effect on perceptual image quality (or greater perceptual image quality for the same compression). The compression is more efficient as the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel, more closely representing the human visual system.

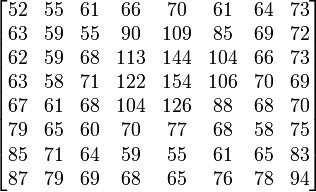
# Block splitting

After subsampling, each channel must be split into 8×8 blocks of pixels. If the data for a channel does not represent an integer number of blocks then the encoder must fill the remaining area of the incomplete blocks with some form of dummy data. Filling the edge pixels with a fixed color (typically black) creates ringing artifacts along the visible part of the border; repeating the edge pixels is a common technique that reduces the visible border, but it can still create artifacts.

# Discrete cosine transform

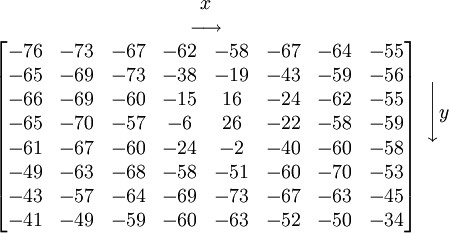
Next, each component (Y, Cb, Cr) of each 8×8 block is converted to a frequency-domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT).

Consider the following as an example of an 8x8 sub image:



Before computing the DCT of the sub image, its gray values are shifted from a positive range to one centered around zero. For an 8-bit image each pixel has 256 possible values: [0,255]. To center around zero it is necessary to subtract by half the number of possible values, or 128.

Subtracting 128 from each pixel value yields pixel values on [− 128,127].

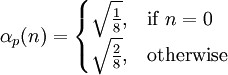


The next step is to take the two-dimensional DCT, which is given by:

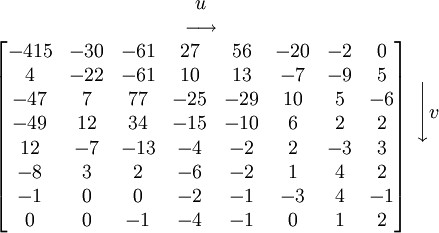
u is the horizontal spatial frequency, for the integers,

v is the vertical spatial frequency, for the integers,



* 
* gx,y is the pixel value at coordinates (x,y).
* Gu,v is the DCT coefficient at coordinates (u,v)

If this transformation is performed on the above matrix,

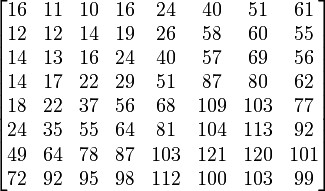


The advantage of the DCT is its tendency to aggregate most of the signal in one corner of the result, as may be seen above.

# Quantization

The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows one to greatly reduce the amount of information in the high frequency components. This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer. This is the main lossy operation in the whole process. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers, which take many fewer bits to store.

A typical Quantization matrix:

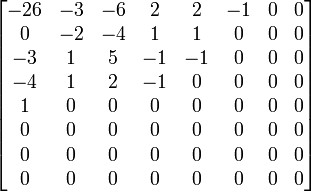


The quantized DCT coefficients are computed with:



Where *G* is the unquantized DCT coefficients; *Q* is the quantization matrix above; and *B* is the quantized DCT coefficients.

Using this quantization matrix with the DCT coefficient matrix from above results in:



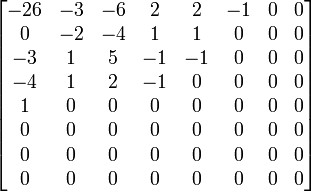
# Entropy coding

Entropy coding is a special form of lossless data compression. It involves arranging the image components in a "zigzag" order employing run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using Huffman coding on what is left.

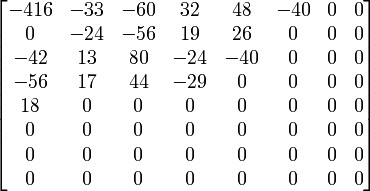
# Decoding

Decoding to display the image consists of doing all the above in reverse.

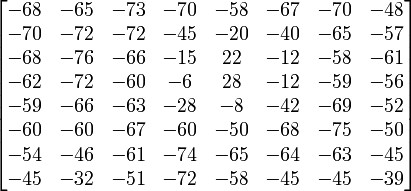
Taking the DCT coefficient matrix (after adding the difference of the DC coefficient back in)



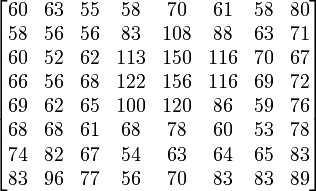
and taking the entry-for-entry product with the quantization matrix from above results in



which closely resembles the original DCT coefficient matrix for the top-left portion. Taking the inverse DCT (type-III DCT) results in an image with values (still shifted down by 128)



and adding 128 to each entry



This is the uncompressed sub image and can be compared to the original sub image

**Tools and Techniques Used**

MATLAB

Image Processing Toolbox

JPEG Compression Algorithm

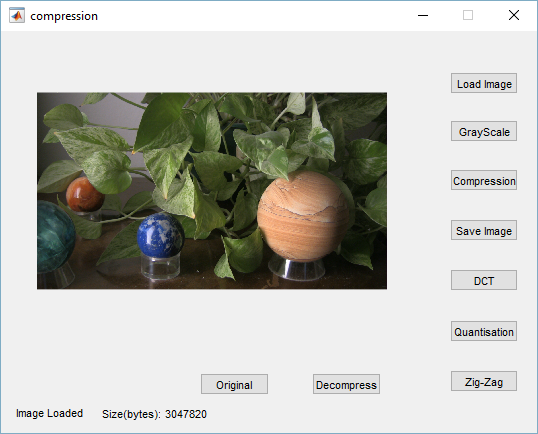
**Need/ Application of Project**

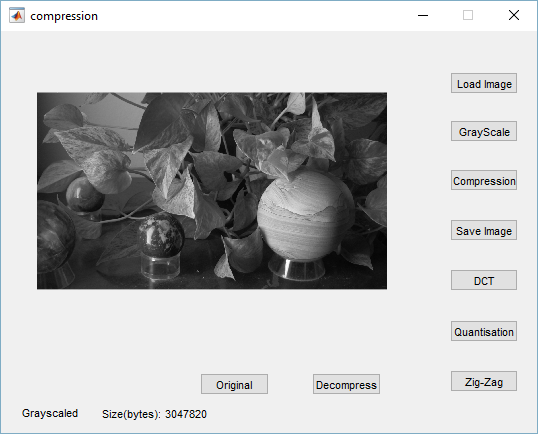
By combining several compression algorithms, JPEG achieves remarkable compression ratios. Even for prepress use, you can easily compress a file to one-fifth of its original size. For web publishing or e-mail exchange, even better ratios up to 20-to-1 can be achieved

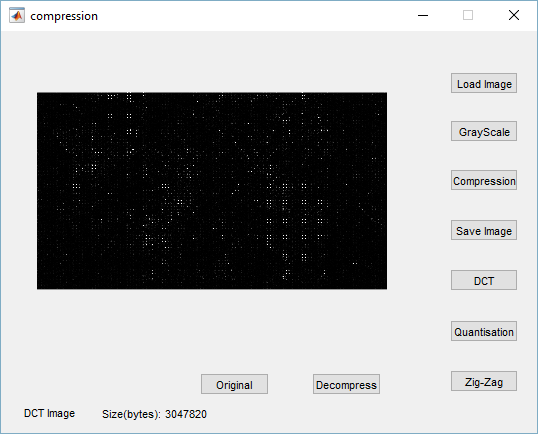
JPEG compression can be used in a variety of file formats that are commonly used in graphic arts:

* EPS files
* EPS DCS files
* JFIF files
* PDF files

**SCREENSHOTS**

****

****

****