DIGITAL IMAGE COMPRESSION

IIVP PROJECT

By:

Suresh Kumar Dhayal (IIT2014060)

Anupam Jaiswal (IIT2014038)

Shubham Gupta (IIT2014061)

Sarthak Panda (IIT2014063)

UNDER THE SUPERVISION OF

Prof. Anupam Agarwal

Professor

IIIT ALLAHABAD

CANDIDATE’S DECLARATION

We hereby declare that the work presented in this project report entitled “**Digital Image Compression**”, submitted report of 6th Semester IIVP project report of B.Tech. (IT) at Indian Institute of Information Technology, Allahabad, is an authenticated record of our original work carried out in April 2017 under the guidance of **Prof. Anupam Agrawal**. Due acknowledgements has been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Place: Allahabad	Anupam Jaiswal	(IIT2014038)
Date: 15 April 2017	Suresh K. Dhayal	(IIT2014060)
	Shubham Gupta	(IIT2014061)
	Sarthak Panda	(IIT2014063)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 15 April 2017
Place: Allahabad

Dr. Anupam Agrawal
Professor
IIIT-Allahabad

<u>**Table of Contents**</u>

Introduction

A **digital image** is a numeric representation of (normally binary) a two-dimensional image. Depending on whether the image resolution is fixed, it may be of vector. In today's world we use digital images a lot and as these images are simply a digital data, so they need to be stored. These digital images are stored in form of binary data which represents the information in the image.

Need of Image compression :

To efficiently use these images and store them effectively so they consume less space we try to reduce its size and losing information as minimum as possible.

Data Compression and Data Redundancy:

Data compression is defined as the process of encoding data using a representation that reduces the overall size of data. This reduction is possible when the original dataset contains some type of redundancy. Digital image compression is a field that studies methods for reducing the total number of bits required to represent an image. This can be achieved by eliminating various types of redundancy that exist in the pixel values. In general, three basic redundancies exist in digital images that follow.

Coding Redundancy:

The uncompressed image usually is coded with each pixel by a fixed length. For example, an image with 256 gray scales is represented by an array of 8 bit integers. Using some variable length code schemes such as Huffman coding and arithmetic coding may produce compression.

Inter - pixel Redundancy:

It is a redundancy corresponding to statistical dependencies among pixels, especially between neighboring pixels.

Psycho visual Redundancy:

It is a redundancy corresponding to different sensitivities to all image signals by human eyes. Therefore, eliminating some less relative important information in our visual processing may acceptable Because its omission results in a loss of quantitative information (which is not essential), its removal is commonly referred to as quantization.It means mapping of a broad range input values to a limited number of output values. Because information is lost, quantization is an irreversible operation.

Compression ratio:

If we represent b and b' denote the number of bits in two representations of the same information, then we define data compression ratio and relative data redundancy as:

Compression ratio $C = b/b'$

relative data redundancy $R = 1 - 1/C$

There are different methods to deal with different kinds of aforementioned redundancies. As a result, an image compressor often uses a multi step algorithm to reduce these redundancies.

Problem Definition and objectives

Our aim is to study various image compression techniques such as standard JPEG image compression technique. We aim to find out which technique is more suitable for specific type of image.

Literature Survey

1- Digital Image compression techniques by Gonzalez
2- Survey of different image compression techniques by Shum
3- Image compression using Wavelet transform by kirandeep kaur

Hardware and Software requirements

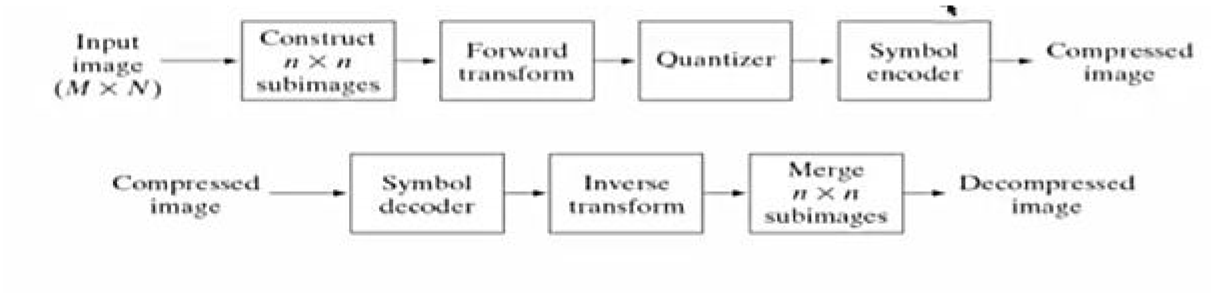
1. Matlab r2011a or higher version
2. Image viewer
3. Operating system compatible with installed matlab

Proposed Approach

Compression Methods:

During the past two decades, various compression methods have been developed to address major challenges faced by digital imaging. These compression methods can be classified broadly into lossy or lossless compression. Lossy compression can achieve a high compression ratio since it allows some acceptable degradation. Yet it cannot completely recover the original data. On the other hand, lossless compression can completely recover the original data but this reduces the compression ratio.
In medical applications, lossless compression has been a requirement because it facilitates accurate diagnosis due to no degradation on the original image.

Lossy Compression Methods:



Generally most lossy compressors are three step algorithms, each of which is in accordance with three kinds of redundancy mentioned above. The first stage is a transform to eliminate the inter-pixel redundancy to pack information efficiently. Then a quantizer is applied to remove psycho-visual redundancy to represent the packed information with as few bits as possible. The quantized bits are then efficiently encoded to get more compression from the coding redundancy.

1- Transform Coding:

Transform coding is a general scheme for lossy image compression. It uses a reversible and linear transform to decorrelate the original image into a set of coefficients in transform domain. The coefficients are then quantized and coded sequentially in transform domain. Numerous transforms are used in a variety of applications. The discrete KLT (Karhunen-Loeve transform), which is based on the Hotelling transform, is optimal with its information packing properties, but usually not practical since it is difficult to compute.
The DFT (discrete Fourier transform) and DCT (discrete cosine transform) approximate the energy packing efficiency of the KLT, and have more efficient implementation. In practice, DCT is used by most practical transform systems since the DFT coefficients require twice the storage space of the DCT coefficients.

Block Transform Coding:

In order to simplify the computations, block transform coding exploits correlation of the pixels within a number of small blocks that divide the original image. As a result, each block is transformed, quantized and coded separately. This technique, using square 8*8 pixel blocks and the DCT followed by Huffman or arithmetic coding, is utilized in the ISO JPEG (joint photographic expert group) draft international standard for image compression.

Full-Frame Transform Coding:

To avoid the artifacts generated by block transforms, full-frame methods, in which the transform is applied to the whole image as a single block, have been investigated in medical imaging research. The tradeoff is the increased computational requirement.

Discrete Cosine Transform:

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:

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

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

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

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

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

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

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

The basic operation of the DCT is as follows:

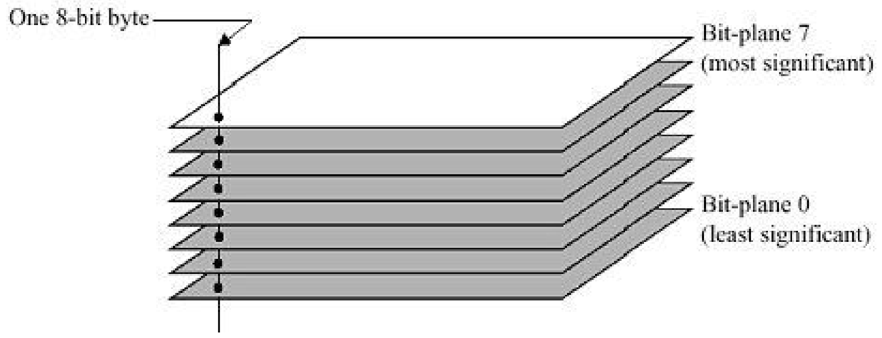
- The input image is N by M;
- f(i,j) is the intensity of the pixel in row i and column j;
- F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.
- For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
- Compression is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

Bit Plane Slicing and encoding:

Bitplanes

Grayscale digital images can be thought of as a matrix of pixels with intensities (values). In an 8-bit image, these values range from 0 to 255. The bit planes of an 8-bit grayscale image can be thought of as the stack of 8 binary images, containing only white or black pixels. Each pixel value is determined based upon the binary representation of its intensity. The first bit plane contains a set of pixels whose *least* significant bits (LSB) are on, and the 8th bit plane contains the set of pixels whose *most* significant bits (MSB) are on.

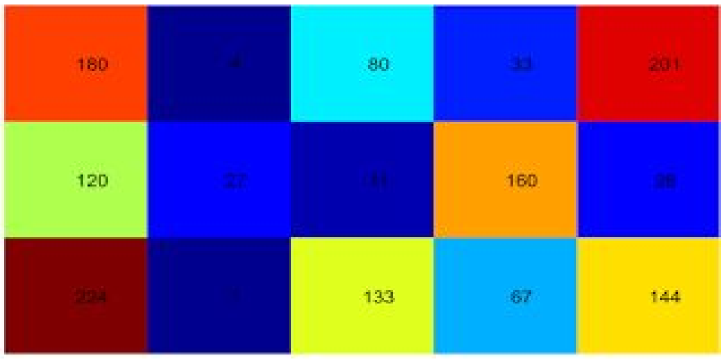
In bit plane slicing , for slicing one bit plane we set all other bit to 0 except that particular bit for which we are performing slicing. Each bit plane contains different amount of data. , so by removing those bit planes which don't have much data would help in compressing image.So from above we can understand that Bitplane slicing is a lossy image compression technique.



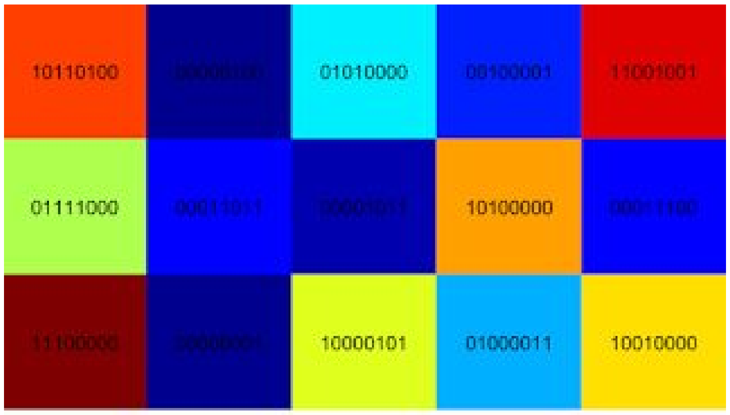
Let’s explain with a simple example how encoding and decoding is carried out in Bit plane compression. In this example, the *Most* Significant Bit(MSB) alone is considered and encoded. For better quality image retrieval, combination of various bit planes such as [8 7] ,[[8 7 6], [8 7 6 5] ..etc can be encoded and decoded. The numbers 8,7,6 , 5 ,..etc represent bit positions.

Compression:

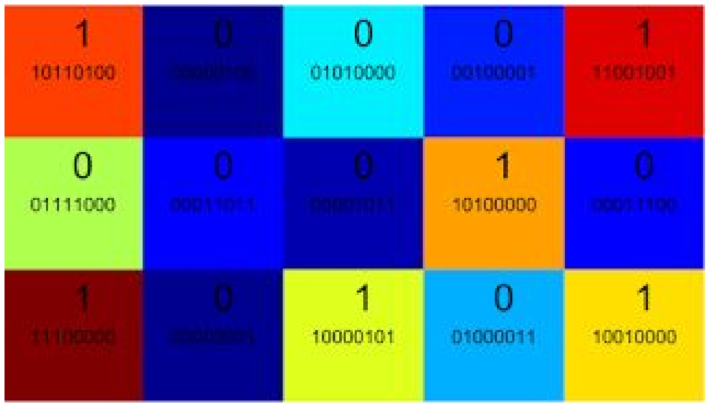
Step 1: Consider a matrix A of size 3 x 5



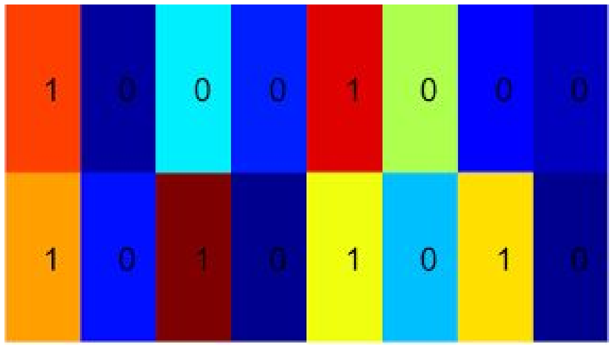
Step 2: Obtain the binary equivalent of the values in the matrix A. The MATLAB function ‘dec2bin’ can be used for conversion.



Step 3: Extract the *Most* Significant Bit(MSB) for each value in the matrix from the binary representation. The MATLAB function 'bitget' can be used for the same.

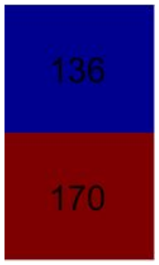


Step 4: Rearrange the above MSB values such that each row contains 8 columns or 8 bits. In the above example, we have 3x5=15 values but we need 8 columns in each row. It can be achieved by padding the matrix with zeros in the end in order to form a matrix which has 8 columns in each row.



1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0

Step 5: Convert the binary representation in each row to a decimal number and store it.Use ‘bin2dec’ MATLAB function to convert binary values to decimal values.In our example, decimal equivalent of [1 0 0 0 1 0 0 0] = 136 and [1 0 1 0 1 0 1 0] = 170

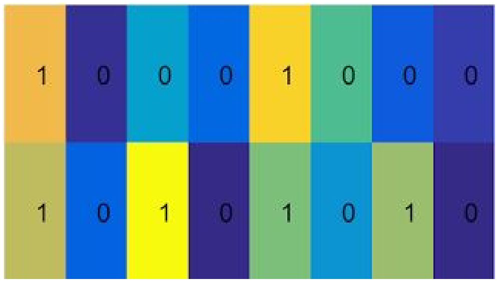


136
170

Decompression:

For Decoding an image/matrix, the compressed data has to be provided as the input along with size of the original image/matrix and the vector containing the position of the bits used for encoding in order.

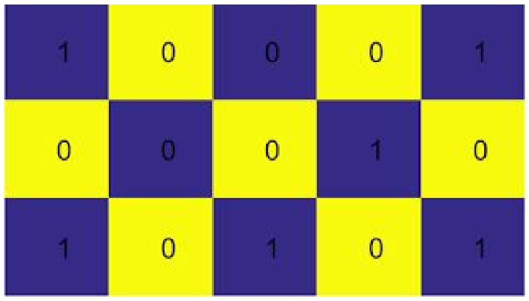
Step 1: Convert the decimal value of the compressed data into binary format.



1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0

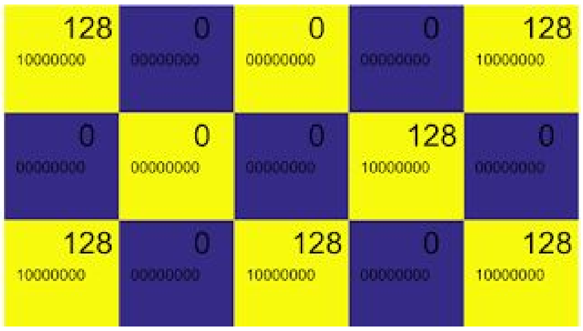
Step 2: Remove the extra zeros appended to the matrix, if needed.

Step 3: Reshape the matrix to size of the original matrix A using the MATLAB function 'reshape'.



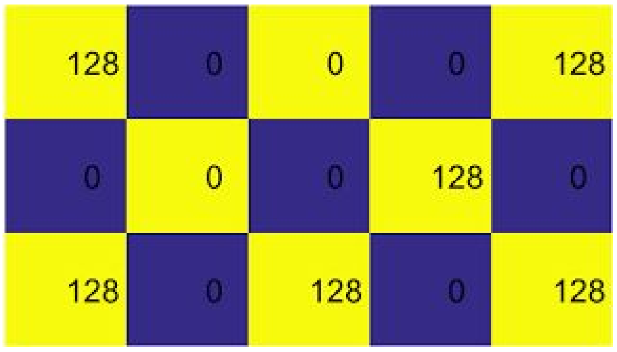
1	0	0	0	1
0	0	0	1	0
1	0	1	0	1

Step 4: Preallocate a matrix of same size of original matrix A and replace the MSB(Most Significant Bit) of each value in the matrix with the bit we decompressed in the previous step.Use the MATLAB function 'bitset'.



128 10000000	0 00000000	0 00000000	0 00000000	128 10000000
0 00000000	0 00000000	0 00000000	128 10000000	0 00000000
128 10000000	0 00000000	128 10000000	0 00000000	128 10000000

Step 5: Display the final data.



128	0	0	0	128
0	0	0	128	0
128	0	128	0	128

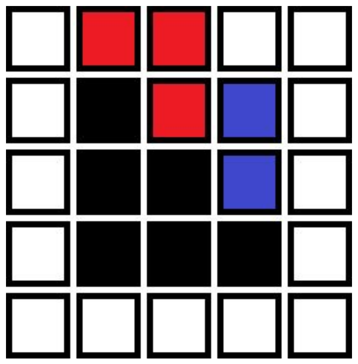
Lossless Image Compression Method:

<u>Huffman Compression:</u>

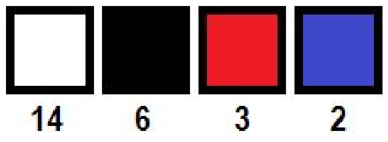
In 1952 David Huffman, a graduate student at the famous Massachusetts Institute of Technology developed an elegant algorithm for lossless compression as part of his schoolwork. The algorithm is now known as Huffman coding.

Huffman coding can be used to compress all sorts of data. It is an entropy-based algorithm that relies on an analysis of the frequency of symbols in an array.

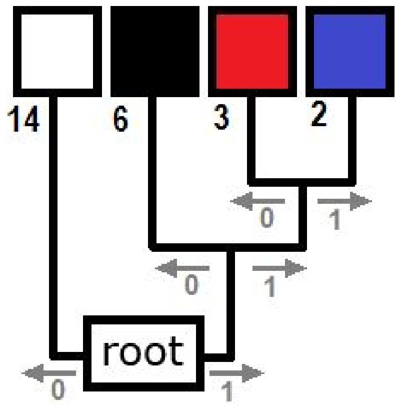
Huffman coding can be demonstrated most vividly by compressing a raster image. Suppose we have a 5×5 raster image with 8-bit color, i.e. 256 different colors. The uncompressed image will take 5 x 5 x 8 = 200 bits of storage.



First, we count up how many times each color occurs in the image. Then we sort the colors in order of decreasing frequency. We end up with a row that looks like this:



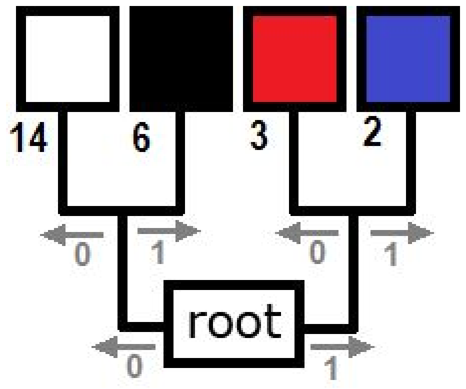
Now we put the colors together by building a tree such that the colors farthest from the root are the least frequent. The colors are joined in pairs, with a node forming the connection. A node can connect either to another node or to a color. In our example, the tree might look like this:



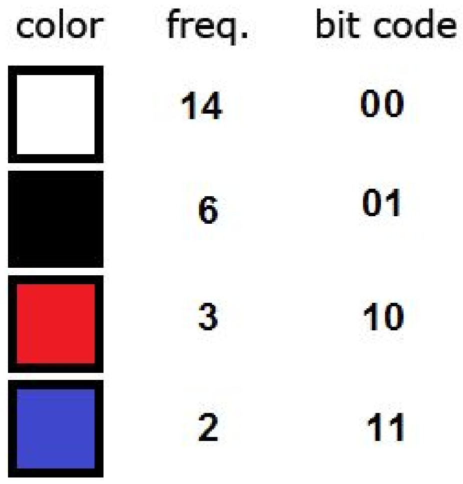
Our result is known as a Huffman tree. It can be used for encoding and decoding. Each color is encoded as follows. We create codes by moving from the root of the tree to each color. If we turn right at a node, we write a 1, and if we turn left – 0. This process yields a Huffman code table in which each symbol is assigned a bit code such that the most frequently occurring symbol has the shortest code, while the least common symbol is given the longest code.

color	freq.	bit code
⬜	14	0
⬛	6	10
🟥	3	110
🟦	2	111

The Huffman tree and code table we created are not the only ones possible. An alternative Huffman tree that looks like this could be created for our image:



The corresponding code table would then be:



color	freq.	bit code
⬜	14	00
⬛	6	01
🟥	3	10
🟦	2	11

Using the variant is preferable in our example. This is because it provides better compression for our specific image.

Because each color has a unique bit code that is not a prefix of any other, the colors can be replaced by their bit codes in the image file. The most frequently occurring color, white, will be represented with just a single bit rather than 8 bits. Black will take two bits. Red and blue will take three. After these replacements are made, the 200-bit image will be compressed to 14 x 1 + 6 x 2 + 3 x 3 + 2 x 3 = 41 bits, which is about 5 bytes compared to 25 bytes in the original image.

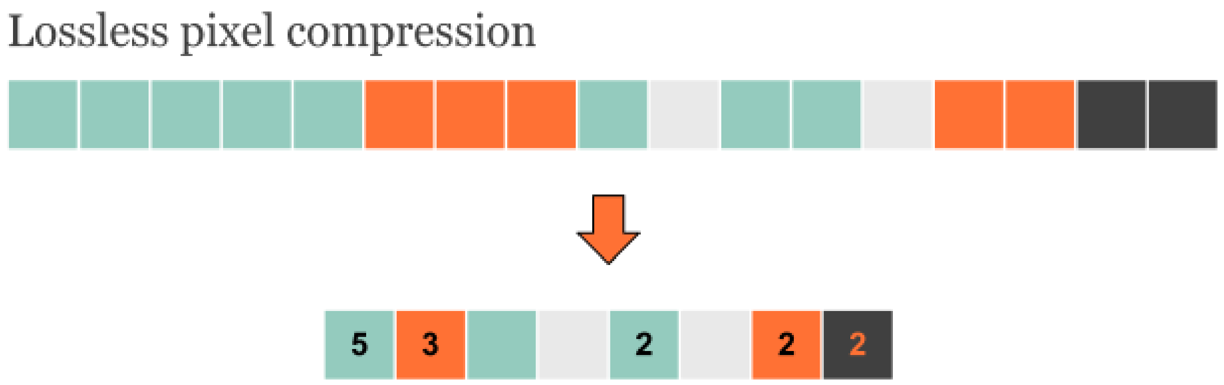
We know that it’s a prefix free encoding so to decode we can start from beginning of the encoded string and traverse the tree left or right as we encounter zeros and ones. If we end up at a leaf node, we print the character and reset our pointer to the root of the tree.

Of course, to decode the image the compressed file must include the code table, which takes up some space. Each bit code derived from the Huffman tree unambiguously identifies a color, so the compression loses no information.

Run-Length Encoding:

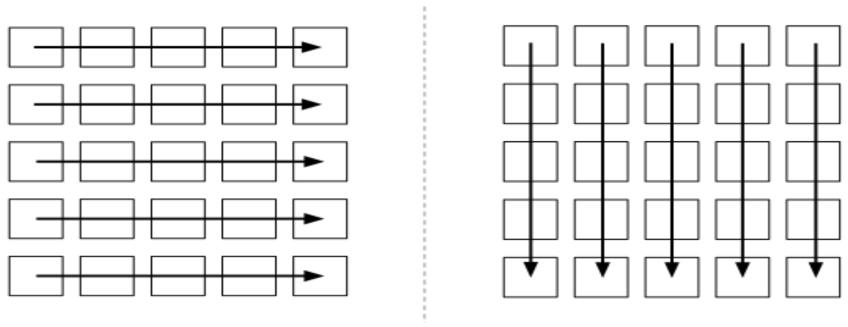
Run-length encoding is a data compression algorithm that helps us encode large runs of repeating items by only sending one item from the run and a counter showing how many times this item is repeated. Unfortunately this technique is useless when trying to compress natural language texts, because they don’t have long runs of repeating elements. In the other hand RLE is useful when it comes to image compression, because images happen to have long runs pixels with identical color.

As you can see on the following picture we can compress consecutive pixels by only replacing each run with one pixel from it and a counter showing how many items it contains.



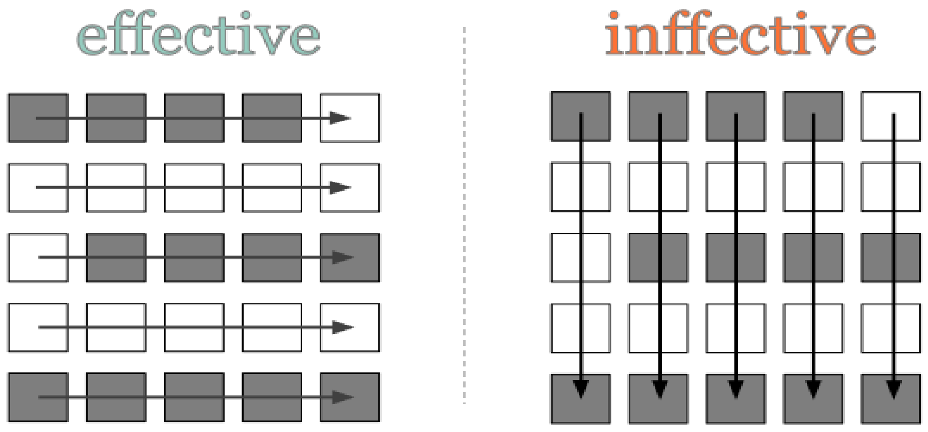
Although lossless RLE can be quite effective for image compression, it is still not the best approach!
In this case we can save only counters for pixels that are repeated more than once. Such the input stream “aaaabbbb” will be compressed as “[4]a[2]ba[1]b[1]a[1]”.

Actually there are several ways run-length encoding can be used for image compression. A possible way of compressing a picture can be either row by row or column by column, as it is shown on the picture below.



Row by row or column by column compression.

The problem in practice is that sometimes compressing row by row may be effective, while in other cases the same approach is very ineffective. This is illustrated by the image below.

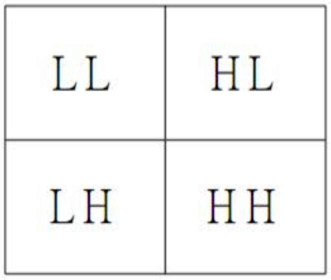


Sometimes image compression may be done only after some preprocessing that can help us understand the best compression approach.

Obviously run-length encoding is a very good approach when compressing images, however when we talk about big images with millions of pixels it's somehow natural to come with some lossy compression.

Discrete Wavelet Transform:

The discrete wavelet transform is a very useful tool for signal analysis and image processing, especially in multi-resolution representation. It can decompose signal into different components in the frequency domain. One-dimensional discrete wavelet transform (1-D DWT) decomposes an input sequence into two components (the average component and the detail component) by calculations with a low-pass filter and a high-pass filter. Two-dimensional discrete wavelet transform (2-D DWT) decomposes an input image into four sub-bands, one average component (LL) and three detail components (LH, HL, HH) as shown in Figure.



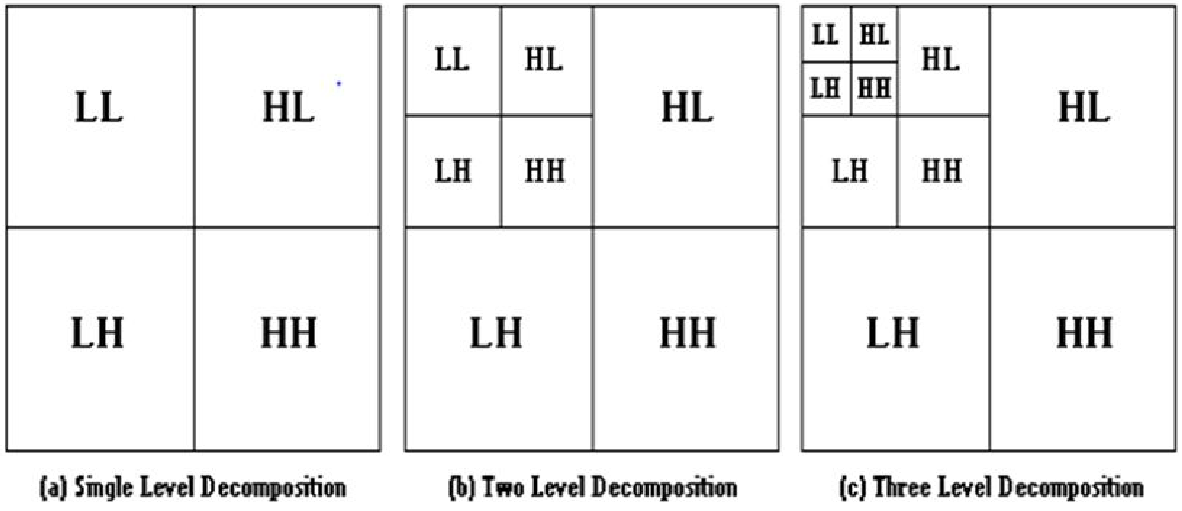
Here in this DWT we are using **Haar wavelet transform** because The operation for Haar DWT is simpler than that of any other wavelets. It has been applied to image processing especially in multi-resolution representation. Haar DWT has the following important features.

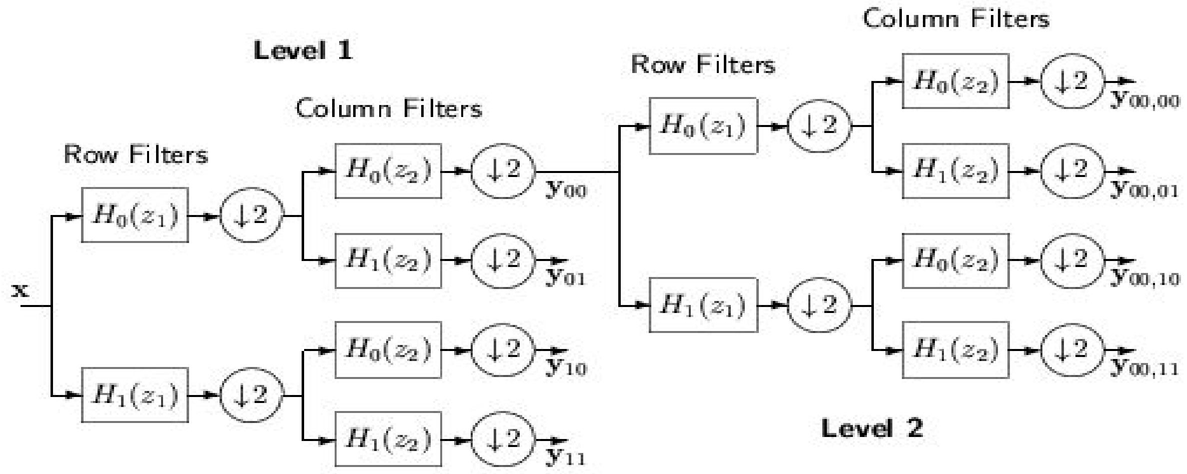
Haar wavelets are real, orthogonal, and symmetric.

The high-pass filter and the low-pass filter coefficient is simple.

The procedure goes like this. A low pass filter and a high pass filter are chosen, such that they exactly halve the frequency range between themselves. This filter pair is called the Analysis Filter pair. First, the low pass filter is applied for each row of data, thereby getting the low frequency components of the row. But since the lpf (low pass filter) is a half band filter, the output data contains frequencies only in the first half of the original frequency range. They can be subsampled by two, so that the output data now contains only half the original number of samples. Now, the high pass filter is applied for the same row of data, and similarly the high pass components are separated, and placed by the side of the low pass components. This procedure is done for all rows..

Next, the filtering is done for each column of the intermediate data. The resulting two-dimensional array of coefficients contains four bands of data, each labelled as LL (low-low), HL (high-low), LH (low-high) and HH (high-high). The LL band can be decomposed once again in the same manner, thereby producing even more subbands. This can be done upto any level, thereby resulting in a pyramidal decomposition as shown below.


LL
HL
LH
HH
(a) Single Level Decomposition
(b) Two Level Decomposition
(c) Three Level Decomposition



Two levels of a 2-D filter tree, formed from 1-D lowpass (H0H0) and highpass (H1H1) filters.

1)Matrix (a) shows detail coefficients of original image

2)Matrix (b) show detail coefficients after row operation

3)Matrix (c) show detail coefficients after column operation

$$\begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix}$$

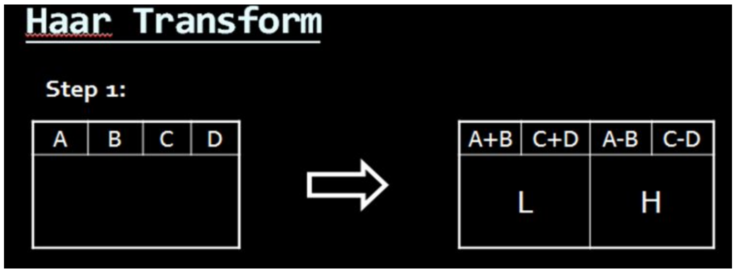
(a)

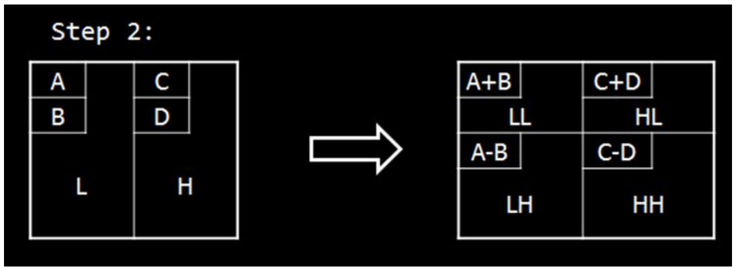
$$\begin{bmatrix} (A+B) & (C+D) & (A-B) & (C-D) \\ (E+F) & (G+H) & (E-F) & (G-H) \\ (I+J) & (K+L) & (I-J) & (K-L) \\ (M+N) & (O+P) & (M-N) & (O-P) \end{bmatrix}$$

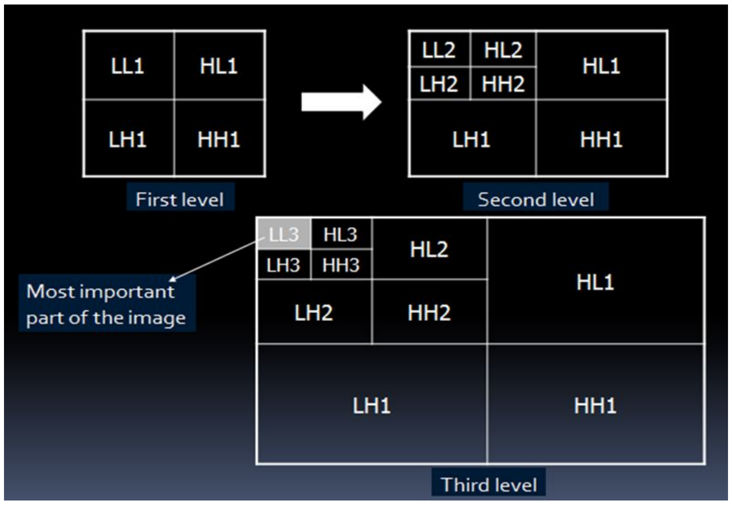
(b)

$$\begin{bmatrix} (A+B)+(E+F) & (C+D)+(G+H) & (A-B)+(E-F) & (C-D)+(G-H) \\ (I+J)+(M+N) & (K+L)+(O+P) & (I-J)+(M-N) & (K-L)+(O-P) \\ (A+B)-(E+F) & (C+D)-(G+H) & (A-B)-(E-F) & (C-D)-(G-H) \\ (I+J)-(M+N) & (K+L)-(O+P) & (I-J)-(M-N) & (K-L)-(O-P) \end{bmatrix}$$

(c)







Inverse Discrete wavelet transform:

Just as a forward transform to used to separate the image data into various classes of importance, a reverse transform is used to reassemble the various classes of data into a reconstructed image. A pair of high pass and low pass filters are used here also. This filter pair is called the Synthesis Filter pair. The filtering procedure is just the opposite - we start from the topmost level, apply the filters column wise first and then row wise, and proceed to the next level, till we reach the first level.

Conclusion

We studied and implemented above mentioned techniques and we concluded following points for each technique.

1- Standard JPEG compression using DCT , zigzag and run length encoding:

- Use of quantizer (Discrete cosine Transform and normalization in our case) in this technique resulted in lossy image compression. This technique shouldn’t be used for compressing images for medical purposes as compression will be lossy and the resulting image will miss some information which may be useful for medical examiners.
- Basic structure was shown in figure above. We may use Huffman encoding for image encoding instead of zig zag traversal followed by run length encoding.

2- Bit plane encoding Compression:

- Shouldn’t be used for compression of images for medical purposes for reasons mentioned above.
- Not much information is lost as we always include the most significant bit plane.
- It is possible to see that the first bit plane gives the roughest but the most critical approximation of values of a medium, and the higher the number of the bit plane, the less is its contribution to the final stage. Thus, adding a bit plane gives a better approximation.

3- Huffman encoding:

- Although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods. Specifically, Huffman coding is optimal only if the probabilities of symbols are natural powers of 1/2. This is usually not the case. As an example, a symbol of probability 0.99 carries only $\log_2 (1 / 0.99) \approx 0.014$ bits of information, but Huffman coding encodes each symbol separately and therefore the minimum length for each symbol is 1 bit.

4- Run length encoding:

- If intensity levels are not repetitive then this techniques doesn't result in good compression. That's why in run length we implemented we converted original image to binary image so that we can show the prominent effect of image compression.

5- Discrete Wavelet Transform encoding:

- Wavelet compression is not good for all kinds of data: transient signal characteristics mean good wavelet compression, while smooth, periodic signals are better compressed by other methods, particularly traditional harmonic compression (frequency domain, as by Fourier transforms and related).

References

1- Digital Image compression techniques by Gonzalez
2- Survey of different image compression techniques by Shum
3- Image compression using Wavelet transform by kirandeep kaur
4- https://en.wikipedia.org/wiki/Wavelet_transform
5- mathworks.com