A Project Report On

Image Compression Using Block Truncation Coding

Submitted By

Varun Kumar Ojha
Computer Science and Engineering

2007-2008

Contents

Chapter 1

Introduction

Compressing an image is significantly different than compressing raw binary data. Of course, general purpose compression programs can be used to compress images, but the result is less than optimal. This is because images have certain statistical properties which can be exploited by encoders specifically designed for them. Also, some of the finer details in the image can be sacrificed for the sake of saving a little more bandwidth or storage space. This also means that lossy compression techniques can be used in this area.

Lossless compression involves with compressing data which, when decompressed, will be an exact replica of the original data. This is the case when binary data such as executables, documents etc. are compressed. They need to be exactly reproduced when decompressed. On the other hand, images (and music too) need not be reproduced 'exactly'. An approximation of the original image is enough for most purposes, as long as the error between the original and the compressed image is tolerable.

1.1 Image

An image is essentially a 2-D signal processed by the human visual system. The signals representing images are usually in analog form. However, for processing, storage and transmission by computer applications, they are converted from analog to digital form. A digital image is basically a 2-Dimensional array of pixels.

Images form the significant part of data, particularly in remote sensing, biomedical and video conferencing applications. The use of and dependence on information and computers continue to grow, so too does our need for efficient ways of storing and transmitting large amounts of data. In the figure1.1 [9] we show the concept of image formation for digital system. The Digital system store image as 2 Dimensional array of pixels.

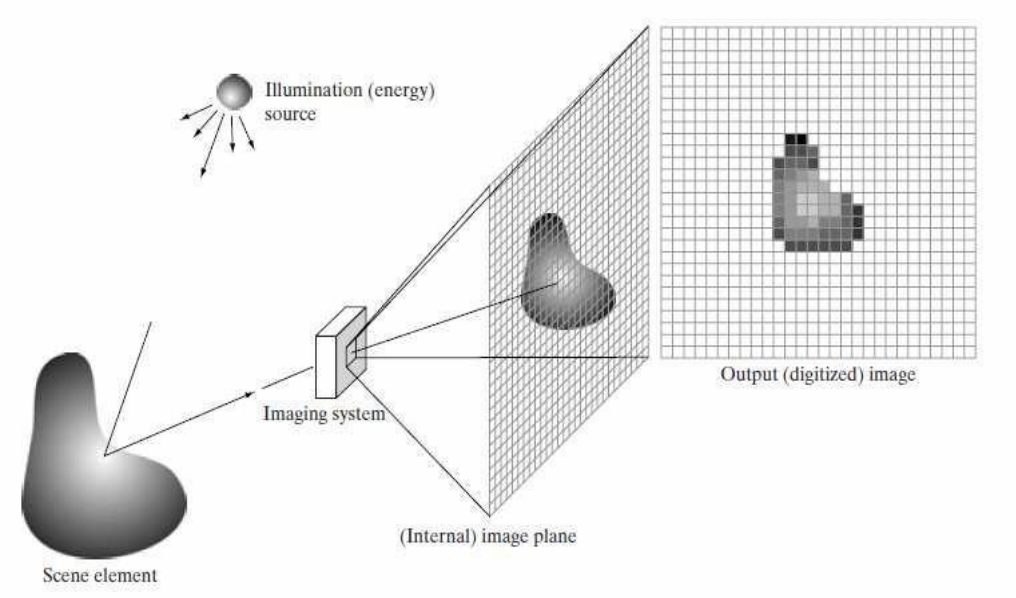


FIGURE 1.1: An example of the digital image acquisition process. (a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

1.2 What is 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

Image compression addresses the problem of reducing the amount of data required to represent a digital image. It is a process intended to yield a compact representation of an image, thereby reducing the image storage/transmission requirements. Compression is achieved by the removal of one or more of the three basic data redundancies:

1. Coding Redundancy
2. Interpixel Redundancy
3. Psycho visual Redundancy

Coding redundancy is present when less than optimal code words are used. Interpixel redundancy results from correlations between the pixels of an image. Psycho visual redundancy is due to data that is ignored by the human visual system (i.e. visually non essential information). Image compression techniques reduce the number of bits required to represent an image by taking advantage of these redundancies. An inverse process called decompression (decoding) is applied to the compressed data to get the reconstructed image. The objective of compression is to reduce the number of bits as much as possible, while keeping the resolution and the visual quality of the reconstructed image as close to the

original image as possible. Image compression systems are composed of two distinct structural blocks: an encoder and a decoder.

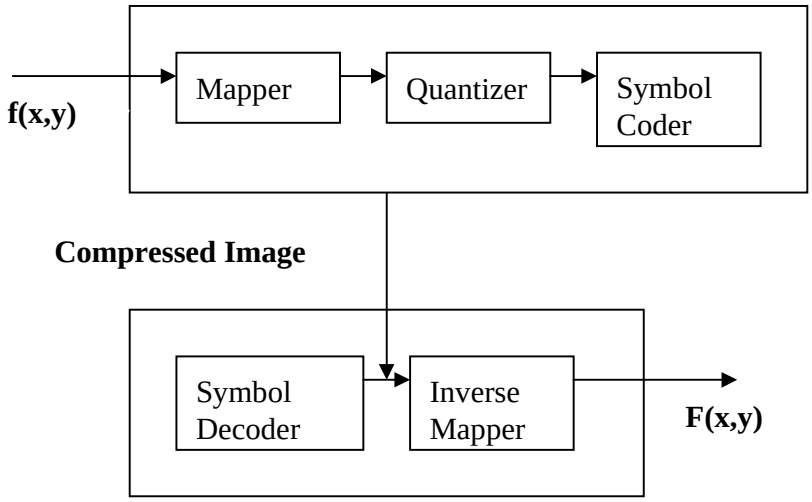


Figure 1.2: Encoder Decoder

Image f(x,y) is fed into the encoder, which creates a set of symbols form the input data and uses them to represent the image. If we let n1 and n2 denote the number of information carrying units(usually bits) in the original and encoded images respectively, the compression that is achieved can be quantified numerically via the compression ratio, CR = n1 /n2

As shown in the figure, the encoder is responsible for reducing the coding, interpixel and psycho visual redundancies of input image. In first stage, the mapper transforms the input image into a format designed to reduce interpixel redundancies. The second stage, qunatizer block reduces the accuracy of mapper's output in accordance with a predefined criterion. In third and final stage, a symbol decoder creates a code for quantizer output and maps the output in accordance with the code. These blocks perform, in reverse order, the inverse operations of the encoder's symbol coder and mapper block. As quantization is irreversible, an inverse quantization is not included.

1.3 Significance of the Image Compression

1. It provides a potential cost savings associated with sending less data over switched telephone network where cost of call is really usually based upon its duration.
2. It not only reduces storage requirements but also overall execution time.
3. It also reduces the probability of transmission errors since fewer bits are transferred.
4. It also provides a level of security against illicit monitoring.

1.4 Image Compression Techniques

The image compression techniques are broadly classified into two categories depending whether or not an exact replica of the original image could be reconstructed using the compressed image. These are:

1. Lossless technique
2. Lossy technique

1.4.1 Lossless compression technique

In lossless compression techniques, the original image can be perfectly recovered form the compressed (encoded) image. These are also called noiseless since they do not add noise to the signal (image).It is also known as entropy coding since it use statistics/decomposition techniques to eliminate/minimize redundancy. Lossless compression is used only for a few applications with stringent requirements such as medical imaging.
Following techniques are included in lossless compression:

1. Run length encoding
2. Huffman encoding
3. LZW coding
4. Area coding

1.4.2 Lossy compression technique

Lossy schemes provide much higher compression ratios than lossless schemes. Lossy schemes are widely used since the quality of the reconstructed images is adequate for most applications .By this scheme, the decompressed image is not identical to the original image, but reasonably close to it.

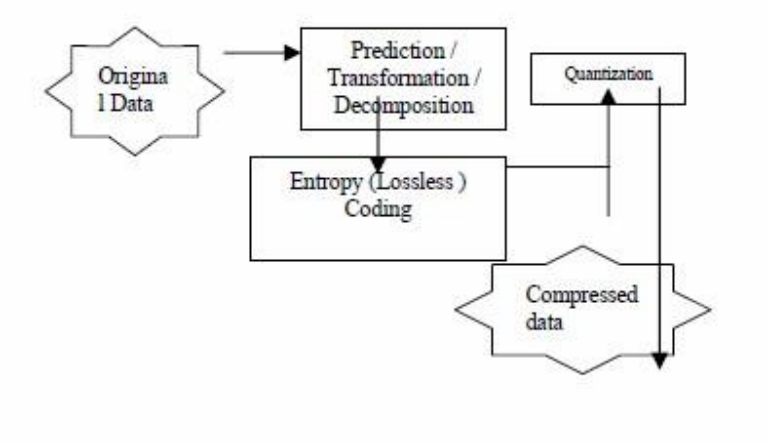


Figure1.3: Outline of Lossy image Compression

As shown above the outline of lossy compression techniques .In this prediction – transformation – decomposition process is completely reversible .The quantization process results in loss of information. The entropy coding after the quantization step, however, is lossless. The decoding is a reverse process. Firstly, entropy decoding is applied to compressed data to get the quantized data. Secondly, dequantization is applied to it & finally the inverse transformation to get the reconstructed image. Major performance considerations of a lossy compression scheme include:

1. Compression ratio
2. Signal - to – noise ratio
3. Speed of encoding & decoding.

Lossy compression techniques includes following schemes:

1. Transformation coding
2. Vector quantization
3. Fractal coding
4. Block Truncation Coding(BTC)
5. Subband coding

1.5 Lossless Compression Techniques

1.5.1 Run Length Encoding

This is a very simple compression method used for sequential data. It is very useful in case of repetitive data. This technique replaces sequences of identical symbols (pixels), called runs by shorter symbols. The run length code for a gray scale image is represented by a sequence {Vi, Ri} where Vi is the intensity of pixel and Ri refers to the number of consecutive pixels with the intensity Vi as shown in the figure. If both Vi and Ri are represented by one byte, this span of 12 pixels is coded using eight bytes yielding a compression ration of 1: 5

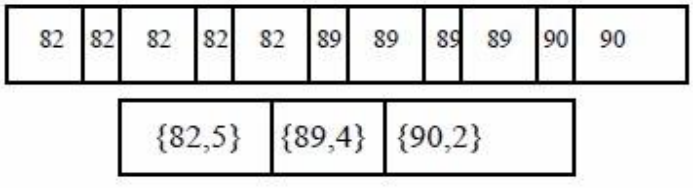


Figure1.4: Run Length Encoding

1.5.2 Huffman Encoding

This is a general technique for coding symbols based on their statistical occurrence frequencies (probabilities). The pixels in the image are treated as symbols. The symbols that occur more frequently are assigned a smaller number of bits, while the symbols that occur less frequently are assigned a relatively larger number of bits. Huffman code is a prefix code. This means that the (binary) code of any symbol is not the prefix of the code of any other symbol. Most image coding standards use lossy techniques in the earlier stages of compression and use Huffman coding as the final step.

1.5.3 LZW Coding

LZW (Lempel- Ziv – Welch) is a dictionary based coding. Dictionary based coding can be static or dynamic. In static dictionary coding, dictionary is fixed during the encoding and decoding processes. In dynamic dictionary coding, the dictionary is updated on fly. LZW is widely used in computer industry and is implemented as compress command on UNIX.

1.5.4 Area Coding

Area coding is an enhanced form of run length coding, reflecting the two dimensional character of images. This is a significant advance over the other lossless methods. For coding an image it does not make too much sense to interpret it as a sequential stream, as it is in fact an array of sequences, building up a two dimensional object. The algorithms for area coding try to find rectangular regions with the same characteristics. These regions are coded in a descriptive form as an element with two points and a certain structure. This type of coding can be highly effective but it bears the problem of a nonlinear method, which cannot be implemented in hardware. Therefore, the performance in terms of compression time is not competitive.

1.6 Lossy Compression Techniques

1.6.1. Transformation Coding

In this coding scheme, transforms such as DFT (Discrete Fourier Transform) and DCT (Discrete Cosine Transform) are used to change the pixels in the original image into frequency domain coefficients (called transform coefficients).These coefficients have several desirable properties. One is the energy compaction property that results in most of the energy of the original data being concentrated in only a few of the significant transform coefficients. This is the basis of achieving the compression. Only those few significant coefficients are selected and the remaining is discarded. The selected coefficients are considered for further quantization and entropy encoding. DCT coding has been the most common approach to transform coding. It is also adopted in the JPEG image compression standard.

1.6.2 Vector Quantization

The basic idea in this technique is to develop a dictionary of fixed-size vectors, called code vectors. A vector is usually a block of pixel values. A given image is then partitioned into non-overlapping blocks (vectors) called image vectors. Then for each image vector in the dictionary is determined and its index in the dictionary is used as the encoding of the original image vector. Thus, each image is represented by a sequence of indices that can be further entropy coded.

1.6.3 Fractal Coding

The essential idea here is to decompose the image into segments by using standard image processing techniques such as color separation, edge detection, and spectrum and texture analysis. Then each segment is looked up in a library of fractals. The library actually contains codes called iterated function system (IFS) codes, which are compact sets of numbers. Using a systematic procedure, a set of codes for a given image are determined, such that when the IFS codes are applied to a suitable set of image blocks yield an image that is a very close approximation of the original. This scheme is highly effective for compressing images that have good regularity and self-similarity.

1.6.4 Block truncation coding

In this scheme, the image is divided into non overlapping blocks of pixels. For each block, threshold and reconstruction values are determined. The threshold is usually the mean of the pixel values in the block. Then a bitmap of the block is derived by replacing all pixels whose values are greater than or equal (less than) to the threshold by a 1 (0). Then for each segment (group of 1s and 0s) in the bitmap, the reconstruction value is determined. This is the average of the values of the corresponding pixels in the original block.

1.6.5 Sub band coding

In this scheme, the image is analyzed to produce the components containing frequencies in well-defined bands, the sub bands. Subsequently, quantization and coding is applied to each of the bands. The advantage of this scheme is that the quantization and coding well suited for each of the sub bands can be designed separately.

1.7. PCA (PRINCIPAL COMPONENT ANALYSIS)

1.7.1 Concept

In statistics, PCA is a technique for simplifying a dataset by reducing multidimensional datasets to lower dimensions for analysis. PCA is a standard technique commonly used for data reduction in statistical pattern recognition and signal processing. PCA has been called one of the most valuable results from applied linear algebra. It is used abundantly in all forms of analysis from neuroscience to computer graphics, because it is a simple non- parametric method of extracting relevant information from confusing datasets. PCA is also called the KARHUNEN-LOEVE Transform (KLT, named after Kari Karhunen & Michel Loeve) or the HOTELLING Transform. Its general objectives are:

1. Data reduction
2. Interpretation

There are basically two approaches for performing PCA. They are classical statistical method and artificial neural network method.

1.7.2 PCA Classical Statistical Method

It involves finding eigen values and corresponding eigen vectors of the data set using covariance matrix. The corresponding eigen values of the matrix gives an indication of amount of information the respective principal components represent. The methodology for calculating principal component is given by the following algorithm. Let X1,X2,……..Xm are the sub images of dimension N.

The corresponding algorithm is described as follows:

1. Computation of the global mean (X) from sub images. $\mathbf{X\sim = 1 / M \Sigma Xi}$
2. Subtraction of the mean from each sub image to generate the mean removed image.
   $$\mathbf{Øi = Xi – X\sim}$$
3. Formation of the matrix using mean removed sub image of (M X N) dimension
   $$\mathbf{A = [Ø 1 Ø2 … ……ØM]}$$
4. Computation of the sample covariance matrix (C) of dimension (N X N)
5. Computation of the Eigen values of the covariance matrix. Computation of Eigen values is performed by jaccobian iteration method. $\mathbf{C: \lambda 1 > \lambda 2 > …… > \lambda N}$
6. Computation of the eigen vectors for the eigen values
   $$\mathbf{C: u1 , u2, ……..,uN}$$
7. Dimensionality reduction step.

Keep only the Eigen vectors corresponding to K largest eigen values. These Eigen values are called as “principal components”.

The above said steps are needed to generate the principal components of the image. Corresponding eigen vectors are uncorrelated and have the greater variance. In order to avoid the components that have an undue influence on the analysis, the components are usually coded with mean as zero and variance as one. This standardization of the measurement ensures that they all have equal weight in the analysis.

1.7.3 PCA Neural Network

Artificial neural network are model that attempt to achieve performance via dense inter connection of simple computational elements. The most important property of a neural network is the ability to learn from its environment. PCA is a powerful linear block transform coding in which, an image is subdivided into non-overlapping blocks of N×N pixels which can be considered as N-Dimensional vectors with N = n × n. A linear Transformation, which can be written as an M ×N – dimensional matrix W with $M \leq N$, is performed on each block with the M rows of W, wi being the basis vectors of the transformation. An adaptive principal component extraction (APEX) is used to decorrelate the principal components. The main difference between this APEX architecture and the existing PCA networks lies in the additional lateral connections at the outputs of the network.

1.7.4 Applications of PCA in Computer Vision

1.7.4.1. Representation

When using these sorts of matrix techniques in computer vision, representation of images should be considered. A square, N by N image can be expressed as an N2 – dimensional vector. X = (x1 x2 x3 ……xN2) Where the rows of pixels in the image are placed one after the other to form a one dimensional image. E.g. the first N elements x1 – xN will be the first row of the image, the next N elements are the next row, and so on. The values in the vector are the intensity values of the image, possibly a single grayscale value.

1.7.4.2. PCA to find patterns

Say we have 20 images Each image is N pixels high by N pixels wide. For each image we can create an image vector as described in the representation section. These all images can be put together in one big image-matrix like this :

$$\text{Image Matrix} = \text{ImageVec}_2 \begin{vmatrix} \text{ImageVec}_1 \\ \\ : \\ \text{ImageVec}_{20} \end{vmatrix}$$

This gives a starting point for our PCA analysis. It turns out that these axes works such better for recognizing faces, because the PCA analysis has given the original images in terms

of the differences and similarities between them. The PCA analysis has identified the statistical patterns in the data.

1. 7.4.3. PCA for Image Compression

If we have 20 images each with N2 vectors and 20 dimensions. Now, PCA can be implemented on this set of data. 20 Eigen vectors will be obtained because each vector is 20 –dimensional. To compress the data, choose the data using only 15 Eigen vectors. This gives a final data set with only 15 dimensions, which has saved ¼ of the space. However, when the original data is reproduced, the images have lost some of the information. This compression is said to be lossy because the decompressed image is not exactly the same as the original.

Chapter 2

Block Truncation Coding

In this scheme, the image is divided into non overlapping blocks of pixels. For each block, threshold and reconstruction values are determined. The threshold is usually the mean of the pixel values in the block. Then a bitmap of the block is derived by replacing all pixels whose values are greater than or equal (less than) to the threshold by a 1 (0). Then for each segment (group of 1s and 0s) in the bitmap, the reconstruction value is determined. This is the average of the values of the corresponding pixels in the original block.

2.1 Introduction and Historical Overview

The problem of how one stores and transmits a digital image has been a topic of research for more than 40 years and was initially driven by military applications and NASA. The problem, simply stated, is how does one efficiently represent an image in binary form? This is the image compression problem. It is a special case of the source coding problem addressed by Shannon in his landmark paper [1] on communication systems. What is different about image compression is that techniques have been developed that exploit the unique nature of the image and the observer. These include the spatial nature of the data and of the human visual system. The "efficiency" of the representation depends on two properties of every image compression technique: data rate (in bits/pixel) and distortion in the decompressed image. The date rate is a measure of how much bandwidth one would require to transmit the image or how much space it would take to store the image. Ideally one would like this to be as small as possible. If the decompressed image is exactly the same as the original image, the technique is said to be lossless. Otherwise the technique is lossy and the decompressed image has distortion or coding artifacts in it. Depending on the application, one can often trade distortion for data rate, hence if a user is willing to accept images with more distortion the data rate can often be lower.

Statistical and structural methods have been developed for image compression [2], the former being based on the principles of source coding with emphasis on the algebraic structure of the pixels in an image, whereas the latter methods exploit the geometric structure of the image. In recent years there has been a great deal of activity in formulating standards for image and video compression. Most statistical image compression methods are implemented by segmenting the image into non-overlapping blocks, since dividing the images into blocks allows the image compression algorithm adapt to local image statistics. The disadvantage, however, is that the borders of the blocks are often visible in the decoded image.

In this chapter we describe a lossy image compression technique known as *Block Truncation Coding (BTC)*. In the simplest possible terms: BTC is a block-adaptive binary encoder scheme based on moment preserving quantization.

The basic concepts of BTC were born on March 17, 1977 in the office of O. Robert Mitchell at Purdue University during a conversation between Mitchell and his Ph.D. student Edward J. Delp. Delp and Mitchell discussed many ideas relative to how one could exploit statistical moments in the context of image compression. Delp began working on this concept as part of his Ph.D. thesis.3 The first papers on BTC appeared at the IEEE International Conference on Communications in 1978 [3] and 1979 [4]. The first journal articles also appeared in 1979 [5, 6] along with Delp's thesis [7]. Since 1977 a great deal of work has been done on BTC. There have been more than 200 journal papers, 400 conference papers, 40 Ph.D. Thesis and one book [8] published on BTC. BTC was a final candidate for the JPEG compression standard in 1987.

2.2 Basics Concept of BTC

The basic BTC algorithm is a lossy fixed length compression method that uses a *Q* level quantizer to quantize a local region of the image. The quantizer levels are chosen such that a number of the moments of a local region in the image are preserved in the quantized output. In its simplest form, the objective of BTC is to preserve the sample mean and sample standard deviation of a grayscale image. Additional constraints can be added to preserve higher order moments. For this reason BTC is a block adaptive moment preserving quantizer.

The first step of the algorithm is to divide the image into non-overlapping rectangular regions. For the sake of simplicity we let the blocks be square regions of size n x n, where n is typically 4. For a two level (1 bit) quantizer, the idea is to select two luminance values to represent each pixel in the block. These values are chosen such that the sample mean and standard deviation of the reconstructed block are identical to those of the original block. An n x n bit map is then used to determine weather a pixel luminance value is above or below a certain threshold. In order to illustrate how BTC works, we will let the sample mean of the block be the threshold; a "1" would then indicate if an original pixel value is above this threshold, and "0" if it is below. Since BTC produces a bitmap to represent a block, it is classified as a binary pattern image coding method [10]. The thresholding process makes it possible to reproduce a sharp edge with high fidelity, taking advantage of the human visual system's capability to perform local spatial integration and mask errors. Figure 1 illustrates the BTC encoding process for a block. Observe how the comparison of the block pixel values with a selected threshold produces the bitmap.

By knowing the bit map for each block, the decompression/reconstruction algorithm knows whether a pixel is brighter or darker than the average. Thus, for each block two gray scale values, a and b, are needed to represent the two regions. These are obtained from the sample mean and sample standard deviation of the block, and are stored together with the bit map. Figure 2 illustrates the decompression process. An explanation of how a and b are determined will be given below.

For the example illustrated in Figures 1 and 2, the image was compressed from 8 bits per pixel to 2 bits per pixel (bpp). This is due to the fact that BTC requires 16 bits for the bit map, 8 bits for the sample mean and 8 bits for the sample standard deviation. Thus, the entire 4x4 block requires 32 bits, and hence the data rate is 2 bpp. From this example it is easy to understand how a smaller data rate can be achieved by selecting a bigger block size, or by allocating less bits for the sample mean or the sample standard deviation [5, 7].

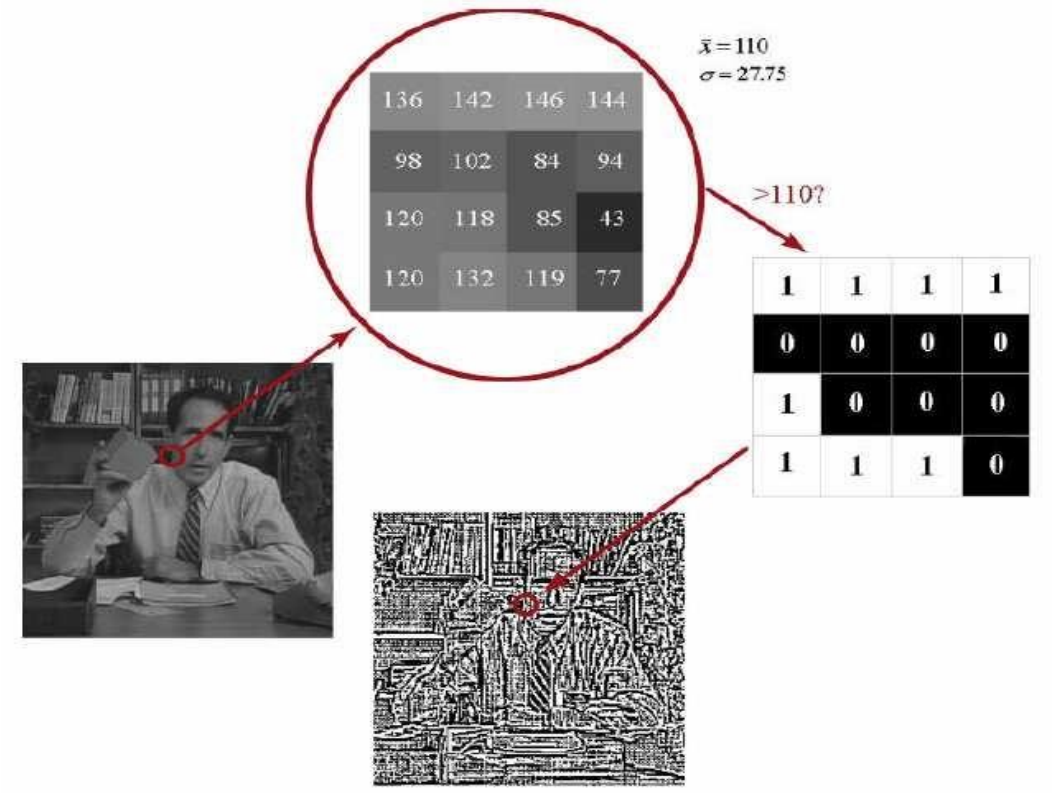


Figure 2.1: Illustration of the BTC Compression Process

To understand how a and b are obtained, let k be the number of pixels of an nxn block (k=n2) and x1,*x*2, … ,*x*k be the intensity values of the pixels in a block of the original image. The first two sample moments *m*1 and *m*2 are given by

$$m_1 = \bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i$$

$$m_2 = \frac{1}{k}\sum_{i=1}^{k} x_i^2 \quad ,$$

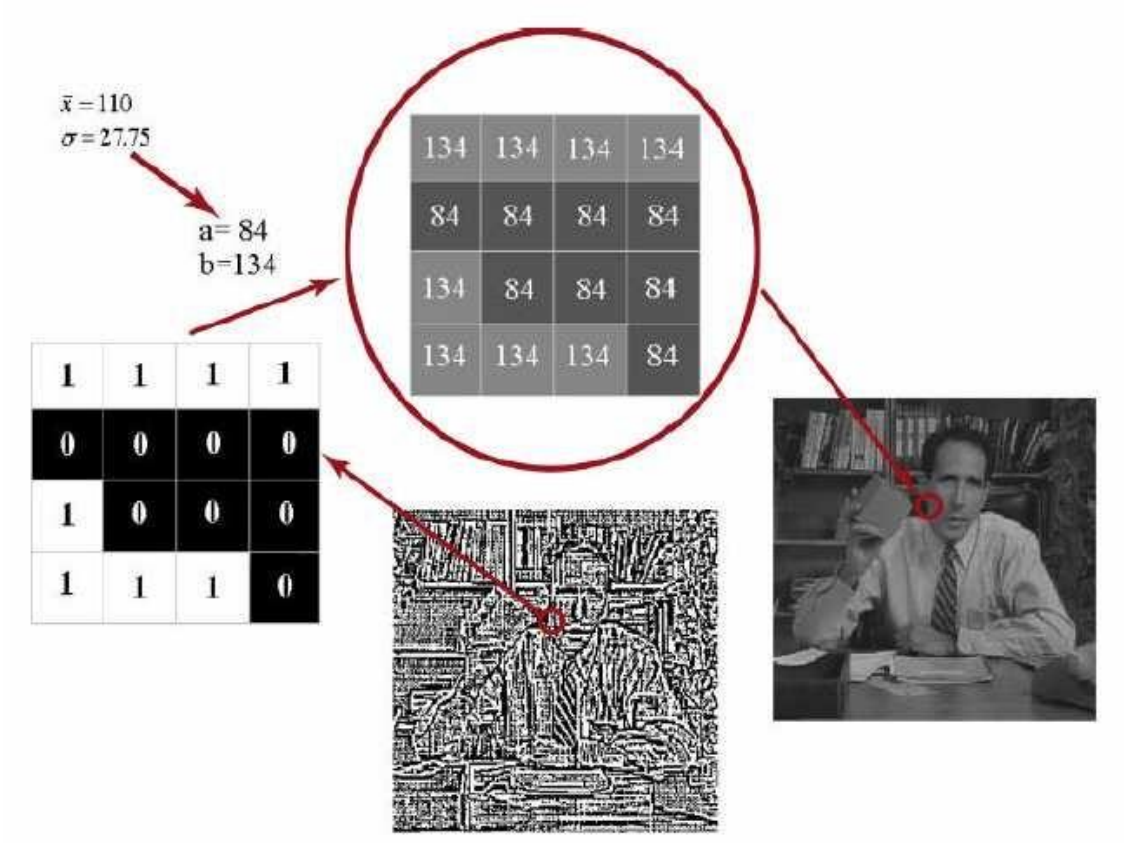


Figure 2.2: Illustration of the Decompression Process

and the sample standard deviation σ is given by

$\sigma_2 = m_2 — m_1^2$

The one bit quantizer for a block and threshold, *xth*, as shown in Figure 3, is defined by

$$Output = \begin{cases} b & if\ x_i \geq x_{th} \\ a & if\ x_i < x_{th} \end{cases} \quad for\ i = 1,2,\cdots,k\ .$$

As the example illustrated, the mean can be selected as the quantizer threshold. Other thresholds could also be used such as the sample median. Another way to determine the threshold is to perform an exhaustive search over all possible intensity values to find a threshold that minimizes a distortion measure relative to the reconstructed image [7].

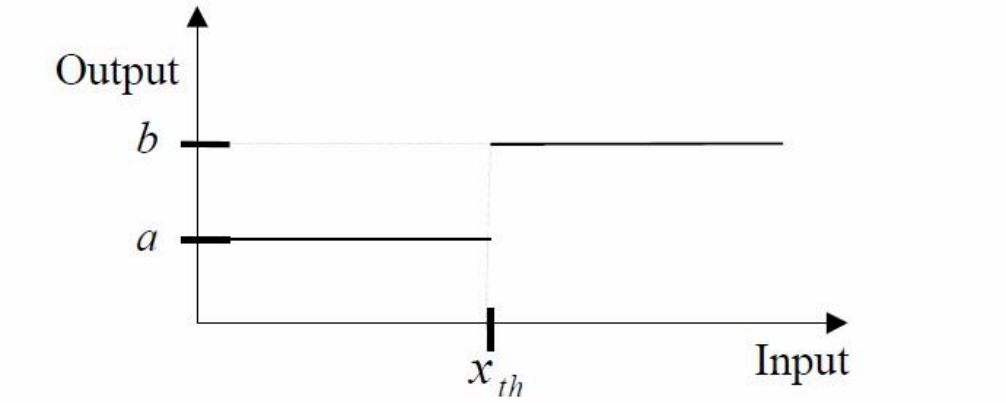


Figure 2.3: Binary Quantizer

Once a threshold, *xth*, is selected (mostly the mean is selected as *xth* which is nothing but m_1) the output levels of the quantizer (*a* and *b*) are found such that the first and second moments are preserved in the output. If we let *q* be the number of pixels in a block that are greater than or equal to *xth* in value, we have:

$$a = m_1 - \sigma\sqrt{\frac{q}{k-q}}$$

$$b = m_1 + \sigma\sqrt{\frac{k-q}{q}}.$$

Each image block is reconstructed by calculating a and b and assigning these values to pixels in accordance with 0s and 1s in the logical bit block. In the logical bit blocks there can be blocks with all the values either 0 or 1, such blocks are visually continuous indicating no edges in them. The sample mean is used to represent that block. So while reconstructing the image, such blocks are given a reconstruction value equal to the mean. In the proposed method, the compressed image is described by the values of block mean m_1, block standard deviation σ, and an n×n bit block consisting of 0s and 1s. Assigning 8 bits to each of m_1 and σ, and 16 bits to bit block results in a data rate of 2 bits/pixel. Thus the bit rate obtained for the proposed method is the same as the conventional BTC.

2.2.1. The Performance Criterion

The performance of the image compression technique is measured by computing the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The MSE gives the difference between the original image and the reconstructed image and is calculated using the equation

$$MSE = \left[\frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} \left[I(x,y) - I'(x,y) \right]^2 \right]^{1/2}$$

The PSNR (Peak Signal to Noise Ratio) is the quality of the reconstructed image and is the inverse of MSE. The PSNR is defined as follows:

$$PSNR(dB) = 10 \log_{10} \frac{\max_{x,y} I^2(x,y)}{\frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} \left[I(x,y) - I'(x,y) \right]^2}$$

Where $I(x, y)$ is the original image, $I'(x, y)$ is the reconstructed image, and $X \times Y$ is the dimensions of the images. A lower value of RMSE means lesser error in the reconstruction, and as seen from the inverse relation between the RMSE and PSNR, this translates to a high value of PSNR. Logically, a higher value of PSNR is preferable because of the higher signal to noise ratio. Here, the signal is the original image and the noise is the error in reconstruction. So a compression method having lower RMSE and corresponding high PSNR values could be recognized as a better scheme.

2.2.2 BTC Effectiveness Calculation

The effectiveness of the algorithm is shown in the figure below. Here the 32 bit is required for a block of 4x4 so 2 bit per pixel is needed hence we can say that bit rate is 2 bpp (bit per pixel).

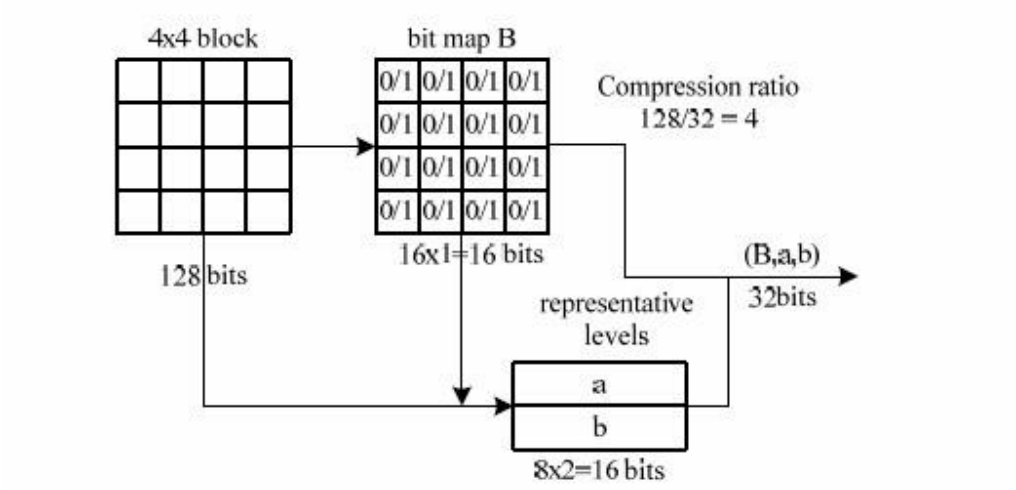


Figure2.4: bit per pixel

2.3 BTC Level Three

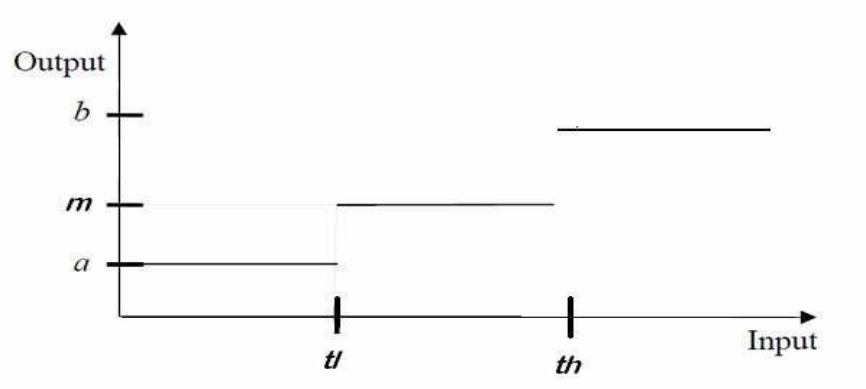
The concept of BTC levels three is a very simple. As in two level or conventional BTC we have a threshold x_{th} that tries to partition data into the block of 4x4 matrixes according to the pixel intensity lower than or greater than the value of threshold. So is here in BTC level three we just inherit the same concept but we have three region separated by two threshold value called th_{low} and $th_{high.}$ These threshold values are defined by and shown in figure

$$th_{low} = (3*a+b)/4$$

$$th_{high} = (3*b+a)/4$$

The value of ‘a’ and ‘b’ is same as in conventional BTC and here we use that ‘a’ and ‘b’ for the calculation of threshold values in BTC level three.

Hence the pixel below or equal to threshold low (th_{low}) is encoded as -1, the pixel intensity greater than th_{low} and less then th_{high} is encoded as 0 and the pixel value greater then or equal to th_{high} is encoded as 1.The extra of 'a3' and 'b3' will attached with the encoded block. The calculation of 'a3' and 'b3' is shown later.



At the decoder end the value of -1 is replaced by value 'a3', the value 0 is replaced by the mean of 'a3' and 'b3' i.e. 'm' and finally the value 1 is replaced by 'b3'.Where The 'a3' is the mean of 'p' no of intensity value X_{ip}. The 'b3' is the mean of 'q' no of intensity value X_{iq} in that block. Value 'a3' and 'b3' is calculated as

$$a3 = 1/p\ (\sum X_{ip})$$

$$b3 = 1/q\ (\sum X_{iq})$$

So at the encoder side -1 in two bit can be represented by '11' and the one bit required for each 0 and 1. So maximum of 2x16=32 bit required encoding 4x4 blocks in the case if all the pixel intensity will be taken below threshold low. In this case the data rate will be equal to **(32 +8x2) / 16pixels = 3.0 bpp** and the compression ratio becomes 128/48=2.67

2.4 BTC Level Four

The BTC level for is slightly different than the conventional BTC algorithm. Here we try to calculate the maximum and minimum for each 4x4 block (as shown as 'A' and 'D'). And then calculating 'B' as the pixel value nearer to the minimum and 'C' as the pixel value nearer to the maximum by the following equations

$$A = \min$$
$$B = (2*\min+\max) / 3$$
$$C = (2*\max+\min) / 3$$
$$D = \max$$

The second step is calculation of the threshold values. In BTC level four we have divided four regions by three threshold values calculated by the following equations

$$th_{low} = (A+B) / 2$$
$$th_{mid} = (B+C) / 2$$
$$th_{high} = (C+D) / 2$$

With calculation of threshold value we also calculate a4, b4, c4 and d4 are the mean value of 'p' no of pixel intensity value in region less then equal to $th_{low,}$, 'q' no of pixel intensity value in region greater then th_{low} and less then equal to th_{mid} , 'r' no of pixel intensity value in region greater then th_{mid} and less then equal to $th_{high,}$'s' no of pixel intensity value in region greater then th_{high} respectively.

$$a4 = 1/p \left(\sum X_{ip}\right)$$

$$b4 = 1/q \left(\sum X_{iq}\right)$$

$$c4 = 1/r \left(\sum X_{ir}\right)$$

$$d4 = 1/s \left(\sum X_{is}\right)$$

Now the encoder encode with 2 bit value as 00, 01, 10 and 11 for the region less then equal to $th_{low,}$, region greater then th_{low} and less then equal to th_{mid} , region greater then th_{mid} and less then equal to $th_{high,}$ region greater then th_{high} respectively. And the extra of a4, b4, c4 and d4 of 8 bit each will be pad up with the encoded block. So the total block size becomes 16x2 + 4x8=64bits. So the data rate becomes **64bit/16pixels=4.0bpp** and the compression

ratio becomes 128/64=2.0. In the decoder part the value 00, 01, 10, and 11 will be replaced by the value of a4, b4, c4 and d4 respectively.

Chapter 3

Implementation JAVA

3.1 BTC level II Encoder Code snippet

```
// Initialization part   int
x=4;                     int
ivalue=0,jvalue=0;   int
icond=x,jcond=x;  long
brk=0;

  long matrix(int array[][],int x,int n,String str)
   {
     int m=x*x;//no of integer data in the broken matrix

try      {
     FileWriter  fwrite= new  FileWriter(str+"\\ImageCompression\\OutputFile\\newout.txt");
PrintWriter pr = new PrintWriter(fwrite);

do
    {
     int i,j,q=0;//q counts the no of '1's in a 4x4 matrix
int xi=0,xi2=0;       double m1=0.0,m2=0.0;
double xth=0.0;       double sig=0.0;       double
sigma=0.0;       double sqa=0.0,sqb=0.0;       double
sqra=0.0,sqrb=0.0;
      int a=0,b=0;

      brk=brk+1;

       for(i=ivalue;i<icond;i++)
        for(j=jvalue;j<jcond;j++)
         {
```

```
        xi=xi+array[i][j];          //calculation of m1
xi2=xi2+(array[i][j]*array[i][j]); //Calculation of m2
        }

//START OF CALCULATION

      m1=(double)xi/m; //calculation of m1
      xth=m1;

      m2=(double)xi2/m; //Calculation of m2

      sig=m2-m1*m1; //calculation of σ

      sigma=Math.sqrt(sig) ; //calculation of σ

      for(i=ivalue;i<icond;i++)
      {
       for(j=jvalue;j<jcond;j++)
       {

         if(array[i][j]>=xth)
         {
          q++;
          pr.println(1); //Encoding as the value >= xth as '1'
         }
         else
          pr.println(0); //Encoding as the value < xth as '0'
       }
      }

      if(q==16)
        q=q-1;


    sqa=(double)q/(m-q);
    sqb=(double)(m-q)/q;

    sqra=Math.sqrt(sqa);
    sqrb=Math.sqrt(sqb);

      a=(int)(m1-sigma*sqra); //Calculation of a
      b=(int)(m1+sigma*sqrb); //Calculation of b

//END OF CALCULATION
```

```
      pr.println(a); // padding 'a' with encoded block
      pr.println(b); // padding 'b' with encoded block



      if(jcond<n) //chekc whther the value of columns is less then 512
      {
       jvalue=jvalue+x; //increase value by 4(points to the next 4x4 matrix first position)in row
jcond=jcond+x; //increase value by 4(points to the next 4x4 matrix last position)in row
       }
else
       {
        ivalue=ivalue+x; //goes to the next position for another 4x4 matrix
icond=icond+x;
        jvalue=0; //in the new row of 4x4 matrix set value of column to 0
jcond=x;
       }
      }
      while(icond<=n);//go to the last position in the row of 512x512 matrix

       System.out.println("\n\n The matrixis Broken Into :"+brk+" matrix of ");
System.out.println(" 4X4 matrix and stored into File newout.txt");


   pr.close();
   }
   catch(IOException e)
   {
    System.out.println("IOException:");
e.printStackTrace();
   }
return 0;
  }
```

3.2 BTC level II Encoder Code snippet

```
  String r="null";  // r keeps the track of values in file   int
array[][]=new int[4][4];
  int i,j,k=0,l=0,a=0,b=0;



do
   {
    for(i=0;i<4;i++)
    {
     for(j=0;j<4&&((r = br.readLine()) !=null);j++)
      array[i][j]=Integer.parseInt(r); // reading block
    }

   if(r!=null)
   {
    a=Integer.parseInt(r=br.readLine()); // reading value of 'a'
    b=Integer.parseInt(r=br.readLine()); // reading value of 'b'
   }
if(r!=null)
    {
for(i=0;i<4;i++)
      for(j=0;j<4;j++)
      {
       if(array[i][j]==1)
        array[i][j]=b; // Decoding by replacing value '1' as value of 'b'
else
        array[i][j]=a; // Decoding by replacing value '0' as value of 'a'

       pr.println(array[i][j]); // Displaying replaced happens as Decoding
      }
    }
   }
  while(r!=null); // Complete reading blocks
```

3.3 BTC level III Encoder Code snippet

```
 int     x=4;              int
ivalue=0,jvalue=0;   int
icond=x,jcond=x;  long
brk=0;
 String r="null";

 long matrix(int array[][],int x,int n,String str)
{
  try
{

     FileReader  fread = new FileReader(str+"\\ImageCompression\\OutputFile\\newout.txt");
BufferedReader brab = new BufferedReader(fread);

     FileWriter  fwrite= new FileWriter(str+"\\ImageCompression\\OutputFile\\newout1.txt");
PrintWriter pr = new PrintWriter(fwrite);

    do
    {       int p=0,q=0;
int i,j,l,k,xiq=0,xip=0;
int th=0,tl=0;       int
a=0,b=0;       double
a3=0,b3=0;
     int arr[][]=new int[4][4];

     brk=brk+1;

//START OF CALCULATION

   for(l=0;l<4;l++)
   {
    for(k=0;k<4&&((r = brab.readLine()) !=null);k++)
     arr[l][k]=Integer.parseInt(r);
   }

    if(r!=null)
```

```
    {
     a=Integer.parseInt(r=brab.readLine());  //reading 'a' of BTC II
b=Integer.parseInt(r=brab.readLine()); //reading 'b' of BTC II
    }

   th=(3*b+a)/4; // Calculating Threshold high
   tl=(3*a+b)/4; // Calculating threshold low

      for(i=ivalue;i<icond;i++)
      {
       for(j=jvalue;j<jcond;j++)
       {

         if(array[i][j]<=tl)
         {
          p++; // counting no of pixel <= tl
          xip=xip + array[i][j];
      pr.println(-1); //Encoding as the value <= tl as '-1'

         }
         else if(array[i][j]>th)
         {
          q++; // counting no of pixel > th
          xiq=xiq + array[i][j];
       pr.println(1); //Encoding as the value > th as '1'
          }
else
        {
      pr.println(0); //Encoding as the value >tl and <= xth as '0'

        }
       }
}
     if(q==0)
q=1;       if(p==0)
       p=1;

      a3=(double)xip/p; // Calculation of 'a3'
      b3=(double)xiq/q; // Calculation of 'b3'

//END OF CALCULATION

       pr.println((int)a3); // padding 'a3' with encoded block
       pr.println((int)b3); // padding 'b3' with encoded block
```

```
        if(jcond<n)
        {
         jvalue=jvalue+x;
         jcond=jcond+x;
        }
else        {
ivalue=iva
lue+x;
icond=ico
nd+x;
jvalue=0;
         jcond=x;
        }
       }
       while(icond<=n&&r!=null);
```

3.4 BTC level III decoder Code snippet

```
do
   {
    for(i=0;i<4;i++)
    {
     for(j=0;j<4&&((r = br.readLine()) !=null);j++)
      array[i][j]=Integer.parseInt(r); // reading block
    }

   if(r!=null) // r keeps the track of values in file
   {
    a=Integer.parseInt(r=br.readLine()); // reading value of 'a3'
b=Integer.parseInt(r=br.readLine()); // reading value of 'b3'
m=(a+b)/2; // Calculating mean m as (a+b)/2
   }
if(r!=null)
    {
     for(i=0;i<4;i++)
      for(j=0;j<4;j++)
      {
       if(array[i][j]==-1)
        array[i][j]=a; // Decoding by replacing value '-1' as value of 'a3'
else if(array[i][j]==0)
        array[i][j]=m; // Decoding by replacing value '0' by m as mean of of 'a3' 'b3'
else if(array[i][j]==1)
        array[i][j]=b; // Decoding by replacing value '1' as value of 'b'
pr.println(array[i][j]);
```

```
    }
   }

  }
  while(r!=null);
```

3.5 BTC level IV Encoder Code snippet

```
 int     x=4;              int
ivalue=0,jvalue=0;   int
icond=x,jcond=x;  long
brk=0;

 long matrix(int array[][],int x,int n,String str)
  {
   int m=x*x;

try     {
     FileWriter  fwrite= new FileWriter(str+"\\ImageCompression\\OutputFile\\newout2.txt");
PrintWriter pr = new PrintWriter(fwrite);

do
{
     int i,j;
     int min=0,max=0;
     int count=0;       int END=0,countLoop=1;       double
A=0.0,B=0.0,C=0.0,D=0.0; //taking A,B,C,D       double
Ax=0.0,Bx=0.0,Cx=0.0,Dx=0.0;
     double A1=0.0,B1=0.0,C1=0.0,D1=0.0;//taking A1,B1,C1,D1
double th=0.0,tm=0.0,tl=0.0;       int p=0,q=0,r=0,s=0;       int
xp=0,xq=0,xr=0,xs=0;
     double diffA=0.0,diffB=0.0,diffC=0.0,diffD=0.0;

     int arr[][]=new int[4][4];


     brk=brk+1;//counting 4X4 matrices operation no:

//------>MAX AND MIN CALCULATION

      for(i=ivalue;i<icond;i++)
```

```
       for(j=jvalue;j<jcond;j++)
        {

        if(count==0)
          {
           min=array[i][j]; // Calculating minimum value in block
max=array[i][j]; // Calculating maximum value in block
          }
        count++;

        if(min>=array[i][j])
          min=array[i][j];

        if(max<=array[i][j])
          max=array[i][j];

       }
//--------> Copping the array into another array:

      int k=0,l=0;
      for(i=ivalue;i<icond;i++)
      {
       for(j=jvalue;j<jcond;j++)
        {
          arr[k][l]=array[i][j];
          l++;
}        k++;
l=0;
      }


//-------->CALCULATION END HERE:

//---->START CALCULATION:

       A=min; // minimum value
       B=(double)(2*min+max)/3; // Calculating value nearer to minimum in block
       C=(double)(2*max+min)/3; // Calculating value nearer to maximum in block
       D=max; // maximum value

      Ax=A;
      Bx=B;
      Cx=C;
      Dx=D;
```

```
      th=(double)(Ax+Bx)/2;  // Calculating threshold low
tm=(double)(Bx+Cx)/2; // Calculating threshold mid
tl=(double)(Cx+Dx)/2;   // Calculating threshold high
for(k=0;k<4;k++)
       for(l=0;l<4;l++)
       {
        if(arr[k][l]<=th)
         {
          p++;       // counting no of pixel <= threshold low
xp=xp+arr[k][l];
      pr.println(0);  //Encoding as the value <= threshold low as '00'

         }

        else if(arr[k][l]>th && arr[k][l]<=tm)
         {
          q++; counting no of pixel between  threshold low and threshold mid
xq=xq+arr[k][l];            pr.println(1);//Encoding as the value falls between  threshold low and
threshold mid as '01'
         }
        else if(arr[k][l]>tm && arr[k][l]<=tl)
         {
          r++; // counting no pixel falls between  threshold mid and threshold high
xr=xr+arr[k][l];            pr.println(2);//Encoding as the value falls between  threshold mid and
threshold high as '10'
         }
        else if(arr[k][l]>tl)
         {
          s++;  // counting no pixel  > threshold high
xs=xs+arr[k][l];                pr.println(3); // Encoding as the value >
threshold high as '11'
         }
       }

        if(p==0)
p=1;
if(q==0)
q=1;
if(r==0)
r=1;          if(s==0)
s=1;
```

```
        A1=((double)xp)/p;// Calculating  a4
        B1=((double)xq)/q; // Calculating  b4
        C1=((double)xr)/r; // Calculating  c4
        D1=((double)xs)/s; // Calculating  d4



//-------->END CALCULATION:


       pr.println((int)A1); // padding 'a4' with encoded block
pr.println((int)B1); // padding 'b4' with encoded block       pr.println((int)C1);
// padding 'c4' with encoded block       pr.println((int)D1); // padding 'd4'
with encoded block

        if(jcond<n)
        {
         jvalue=jvalue+x;
         jcond=jcond+x;
        }
else
        {
         ivalue=ivalue+x;
icond=icond+x;         jvalue=0;
         jcond=x;
        }
       }
       while(icond<=n);

        System.out.println("\n\n The matrixis Broken Into :"+brk+" matrix of ");
System.out.println(" 4X4 matrix and stored into File newout.txt");


   pr.close();
   }
   catch(IOException e)
   {
    System.out.println("IOException:");
e.printStackTrace();
   }
return 0;
  }
```

3.6 BTC level IV decoder Code snippet

```
do
   {
    for(i=0;i<4;i++)
    {
     for(j=0;j<4&&((r = br.readLine()) !=null);j++)
      array[i][j]=Integer.parseInt(r);// reading block
    }

   if(r!=null)
   {
    a=Integer.parseInt(r=br.readLine());// reading a4
b=Integer.parseInt(r=br.readLine());//reading b4
c=Integer.parseInt(r=br.readLine());//reading c4
    d=Integer.parseInt(r=br.readLine());//reading d4
   }
if(r!=null)
    {
     for(i=0;i<4;i++)
      for(j=0;j<4;j++)
      {
       if(array[i][j]==0)        array[i][j]=a; //Decoding value <= threshold
low(replacing 00 by a4)       else if(array[i][j]==1)        array[i][j]=b;//
Decoding value between th low and th mid(replacing 01 by b4)       else
if(array[i][j]==2)        array[i][j]=c;// Decoding value between th mid and th high
(replacing 10 by c4)       else if(array[i][j]==3)        array[i][j]=d; // Decoding
value > threshold high(replacing 11 by d4)

       pr.println(array[i][j]);
      }
    }

}
   while(r!=null);
```

3.6 MSE and PSNR Calculation

```
  String old="null",nw="null";
double MSE=0.0;    int n=0;
double PSNR=0.000;
  int max=0;

   FileReader  fread_old = new FileReader(str+"\\ImageCompression\\OutputFile\\out.txt");
   FileReader  fread_new = new FileReader(str+"\\ImageCompression\\OutputFile\\Final2.txt");


   BufferedReader br_old = new BufferedReader(fread_old);
   BufferedReader br_new = new BufferedReader(fread_new);

    while(n<262144&&(old=br_old.readLine())!=null&&(nw=br_new.readLine())!=null)
    {
     int x1=0,x2=0,val=0;

     x1=Integer.parseInt(old);
     x2=Integer.parseInt(nw);

     if(x1>max)
max=x1;

     val=x1-x2;

     MSE=MSE+(val*val);
n++;
    }


    MSE= Math.sqrt((double)(MSE/262144));

    PSNR=10*(Math.log(((double)(max*max)/(MSE*MSE))));


  System.out.println("\n MSE Levl IV is :"+MSE);
```

System.out.println("\n PSNR Levl IV is:"+PSNR);

Chapter 4

Results Analysis

The results are shown in the table for the different level of BTC. These results are obtained by running the java code for the different algorithm on system having windows XP and JDK6 installed on it. Different images in dat file formate are given input to the system and the Output is produced as the performance value MSE & PSNR along with the encoded file (text file) and decoded file (text file which is later converted to PGM formate by adding heard (P2 \n 512 512 \n 256)to it)

4.1 The Performance tabulation

Image	**Bit Rate**					
	2.0 (BTC II**)**		**3.0(**BTC III**)**		**4.0(**BTC IV**)**	
	MSE	PSNR	MSE	PSNR	MSE	PSNR
Lena	5.79	72.17	4.20	78.57	2.58	88.33
Boat	6.98	69.26	5.06	75.70	3.05	85.81
Peppers	5.67	73.78	3.76	81.97	2.53	89.89
Baboon	11.89	58.79	7.98	66.75	5.53	74.10
Couple	5.58	71.04	3.77	78.89	2.43	87.66
Man	6.62	68.05	4.44	76.01	2.89	84.59
Airplane	6.68	69.48	9.50	62.36	2.80	86.77
Barb	8.49	65.07	5.64	73.26	3.86	80.82
Lake	10.56	63.91	7.55	69.91	4.69	79.42
Kiel	10.48	63.82	9.89	65.01	4.49	80.77
Zelda	3.90	77.26	2.63	85.16	1.85	92.14
The Data from the program run on JDK 6.0 and Window XP						

Table: Performance of BTC algorithms

Dat = A file with the DAT file extension is a Data file.

PGM =PGM files may also be called "portable greymap" files. Grayscale image file encoded with either one or two bytes (8 or 16 bits) per pixel; PGM format type ("P2" for text or "P5" for binary)

PGM file formate Header +Data

P2
512 512 } Header formate
256
…….. Rest of data

4.2 Pictorial Analysis

4.2.1 Lena Image before and after decoding

Orignal lena image

Lena Image in Different bit rate and BTC algorithm

BTC level II bit rate:2.0

BTC level III bitrate:3.0

BTC level IV bit rate:4.0

4.2.2 Boat Image before and after decoding

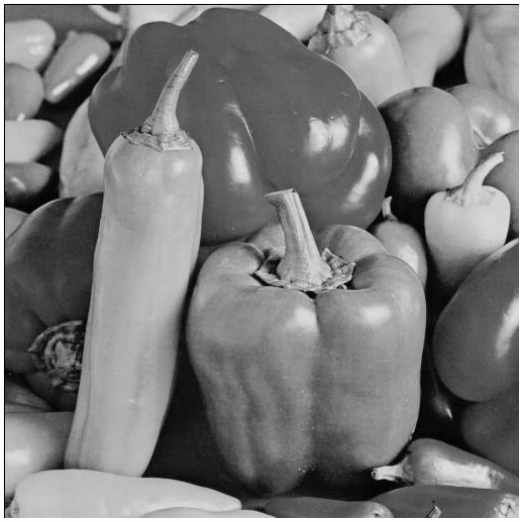
Orignal boat image

BTC level II bit rate:2.0

BTC level III bitrate:3.0

BTC level IV bit rate:4.0

4.2.3 Peppers Image before and after decoding



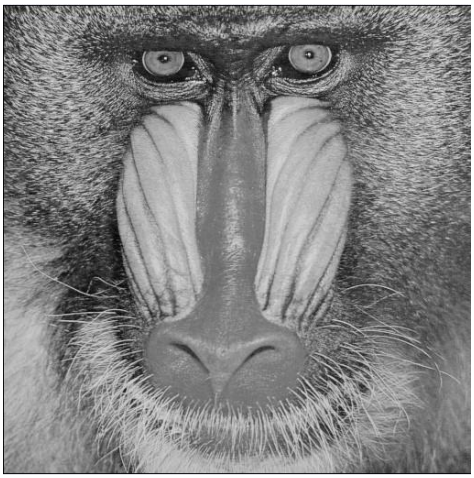
Original peppers Image

BTC level II bit rate:2.0

BTC level III bitrate:3.0

BTC level IV bit rate:4.0

4.2.4 Baboon Image before and after decoding



Original peppers Image

Original Clip peppers Image

BTC level II bit rate:2.0

BTC level III bitrate:3.0

BTC level IV bit rate:4.0

4.2.5 Couple Image before and after decoding

Image adjacent Left is the original couple image and

(a) Original clip of couple
(b) BTC level II: 2.0 bpp
(c) BTC level III: 3.0 bpp
(d) BTC level IV: 4.0 bpp

(a) **(b)** **(c)** **(d)**

Chapter 5

Conclusion and Future Scope

This project deals with the idea that exists presently. And the implementation of the algorithms is done with the help of JAVA. Calculation of MSE and PSNR shows the success of the implementation. Here we have implemented the algorithm and study the results. We can further try to find a solution that's leads to get the minimum MSE and maximum PSNR on lower data rate. So far the study shows that we can achieve minimum MSE on increasing data rate. See table 1. Hence on lower data rate minimum MSE is said to be a good compression technique.

Chapter 6

References

1 C. L. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal,* vol. 27, July 1948, pp. 379-423, 623-656.

2 M. M. Reid, R. J. Millar, and N. D. Black, “Second-generation image coding: an overview,” *ACM Computing Surveys,* vol. 29, no. 1, March 1997, pp. 3-29.

3 O. R. Mitchell , E. J. Delp, and S. G. Carlton, “Block truncation coding: a new approach to image compression,” *Proceedings of the IEEE International Conference on Communications,* vol. 1, June 4-7 1978, pp. 12B.1.1-12B.1.4.

4 E. J. Delp and O. R. Mitchell , “Some aspects of moment preserving,” *Proceedings of the IEEE International Conference on Communications,* vol. 1, June 10-14 1979, pp. 7.2.1-7.2.5.

5 E. J. Delp and O. R. Mitchell, “Image compression using block truncation coding,” *IEEE Transactions on Communications,* vol. 27, no. 9, September 1979, pp. 1335-1341.

6 E. J. Delp, R. L. Kashyap, and O. R. Mitchell, “Image compression using autoregressive time series models,” *Pattern Recognition,* Vol. 11, No. 5/6, 1979, pp. 313-323.[17]

7 E. J. Delp, *Moment preserving quantization and its application in block truncation coding,* Ph.D. thesis, School of Electrical Engineering, Purdue University, August 1979.

8 B. V. Dasarathy, *Image Data Compression: Block Truncation Coding*, IEEE Computer Society Press, Los Alamitos, California, 1995.

9. Gonzalez, Rafael C.Digital Image Processing / Richard E.Woods *Digital Image Processing,* Second Edition, Prentice-Hall, Inc ISBN 0-201-18075-8 2002