LOSSY COLOR IMAGE COMPRESSION BASED ON QUANTIZATION

by

Hiba Shahid

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2015

Abstract

Significant improvements that are based on quantization can be made in the area of lossy image compression. In this thesis, we propose two effective quantization based algorithms for compressing two-dimensional RGB images. The first method is based on a new codebook generation algorithm which, unlike existing VQ methods. It forms the codewords directly in one step, using the less common local information in every image in the training set. Existing methods form an initial codebook and update it iteratively, by taking the average of vectors representing small image regions. Another distinguishing aspect of this method is the use of non-overlapping hexagonal blocks instead of the traditional rectangular blocks, to partition the images. We demonstrate how the codewords are extracted from such blocks. We also measure the separate contribution of each of the two distinguishing aspects of the proposed codebook generation algorithm. The second proposed method, unlike all known VQ algorithms, does not use training images or a codebook. It is implemented to further improve the image quality resulting from the proposed codebook generation algorithm. In this algorithm, the representative pixels (i.e. those that represent pixels that are perceptually similar) are extracted from sets of perceptually similar color pixels. Also, the image pixels are reconstructed using the closest representative pixels. There are two main differences between this algorithm and the existing lossy compression algorithms including our proposed codebook generation algorithm. The first is that this algorithm exploits the image pixel correlation according to the viewer's perception and not according to how close it is numerically to its neighboring

pixels. The second difference is that this algorithm does not reconstruct an entire image block simultaneously using a codeword. Each pixel is reconstructed such that it has the value of its closest representative pixel.

Preface

This thesis presents the research conducted by Hiba Shahid, in collaboration with Dr. Rabab K. Ward. I hereby declare that I am the first author of this thesis. Chapters 3 is based on work that will be submitted for publication. The chapters in this thesis are based on work that I have conducted by myself and performed as part of Mitacs internship for NZ Technologies Inc.

For Chapter 3 and Chapter 4, I carried out the literature review, developed the framework, implemented them in software, carried out the simulations, and analyzed the results. Dr. Ward helped in formulating the research problem, supervised the direction of the research, shaped and enhanced the ideas and provided significant editorial comments and important suggestions for the improving the content and the organization of the paper.

Table of contents

List of tables

List of figures

List of acronyms

2D	Two-dimensional
bpp	Bits per pixel
VQ	Vector Quantization
KFCG	Kekre's Fast Codebook Generation
KEVRW	Kekre's Error Vector Rotation algorithm using Walsh
KLT	Karhunen Loeve Transform
RGB	Red, Green, Blue
DCT	Electroencephalogram
JPEG	Joint Photographic Experts Group
FEC	Forward Error Correction
DWT	Discrete Wavelet Transform
H.P.F.	High Pass Filter
L.P.F.	Low Pass Filter
EBCOT	Embedded Block Coding with Optimized Truncation
EZW	Embedded Zerotree Wavelet
SPIHT	Set Partitioning In Hierarchical Trees
PIFS	Partitioned Iteration Function System
MSE	Mean Square Error
CMC	Color Measurement Committee
PSNR	Peak Signal to Noise Ratio
SSIM	Structural Similarity Index
DPCM	Differential Pulse Code Modulation

Acknowledgements

I thank my supervisor Dr. Rabab K. Ward for allowing me the freedom to pursue my research interests unhindered, for her continued guidance and support, for her patience and valuable assistance in improving and shaping up some of my ideas and great help in editing and presenting the work. I would like to thank the whole team of NZ Technologies for an interesting work environment and Pranav for his valuable inputs. I would also like to thank my family and friends for many a word of encouragement.

This research is part of internship supported by Mitacs-Accelerate Graduate Research Internship Program, partner organization being NZ Technologies Inc. (Application Ref.: IT04139) and People and Planet Friendly Home Environment (PPFH).

Chapter 1

Introduction

The process of storing and transmitting images by modifying them is referred to as **image coding**. For such purposes, an image is coded such that it is represented by fewer number of bits than the original image. This is done to satisfy the memory and transmission channel constraints. This type of image coding is called **image compression**. The thesis is primarily centered on digital image compression. We develop two innovative image compression algorithms. We first explain the nature of two-dimensional (2D) digital images in Section 1.1. Next, we explain the types of digital images that we use in this thesis. Section 1.2 discusses the correlations within an image that we exploit to develop the two algorithms. Section 1.3 outlines the results that we aim to achieve along with the expectations related to the performance of the two algorithms. Section 1.4 lays out the principles that steer our research towards achieving those results. Section 1.5 provides an outline of the thesis.

1.1 Digital images

A 2D digital image is represented by a matrix consisting of a finite set of pixels (picture elements) along its rows and columns. Such a digital image is derived from the real image. The real image is a continuous 2D signal. In order to digitize this image, its values at a fixed number of locations are measured (spatial sampling). The measured values represent pixel intensities in the digital image.

These intensities are quantized such that they are limited to a fixed set of values. Let R denote the number of rows in an image and C denote the number of columns, then the 2D image can be represented in the form of a matrix $I(i,j)$, where $0 \leq i \leq R-1$ and $0 \leq j \leq C-1$ and $I(i,j)$ represents pixel value (intensity) in row i and column j.

The number of pixel values measured from the real image determines the image resolution. The image resolution defines the amount of spatial detail that an image has. This is represented in terms of the number of pixels in an image matrix row and an image matrix column. For example, an image matrix consisting of 512 pixel values in each row and 64 in each column has an image spatial resolution of 512x64.

Images are categorized by the type of information a single pixel represents, within an image. This information is called bit depth or color depth. Bit depth is defined by the number of bits used for representing a single image pixel. An image consisting of 'b' bits per pixel (bpp) is called a b-bit image. Furthermore, the number of possible values stored by that image would be 2^b. For example, each pixel within a simple binary black and white image is represented by only 1 bit. Hence, this gives rise to $2^1 = 2$ possible colors i.e., 0 (white) and 1 (black). Furthermore, an 8-bit image stores $2^8 = 256$ possible colors. Hence, the relationship between image resolution and bit depth is such that the bit depth determines the image resolution.

On the basis of bit depth, an image can be classified as a bi-level image, a grayscale image or a color image. However, we only describe the second and the third image types below. This is because we use these image types in our thesis.

Grayscale images: Each pixel within a grayscale image is represented by 8 bits. That is, the image consists of varying gray values ranging from 0 to 255. In one of our proposed methods, RGB images are used (where an RGB image consists of Red, Green and Blue component channels). Each of these three color components is considered as an 8-bits grayscale image. The algorithm is applied on these channels separately, after that they are merged. This simplifies the image coding process to a great extent. Numerous techniques have been developed for reconstruction of grayscale images in lossy and lossless fashion [1-9].

Color images: In our thesis, we use true color image. Each pixel within a true color image consists of 24-bits. The bits in such an image are divided into three groups: 8 bits for Red channel, 8 bits for Green channel and 8 bits for Blue channel. And so, there are 16,777,216 possible colors that could be stored within such an image. True color images are typically processed by decomposing them into different channels. Different color spaces have different characteristics. Depending on these characteristics, algorithms use the color spaces according to their images coding scheme. In our thesis, RGB and CIEL*a*b* color spaces are used.

More specifically, this thesis is centered on color image compression. We design two novel color compression algorithms. One of them is implemented on the RGB color space. The other is implemented on the CIEL*a*b* color space.

1.2 Perceptual and numerical redundancies in images

The prime focus of a compression technique is to reduce any redundancy within the image. There are different forms of image redundancies. The type of redundancy that is targeted depends on the application requirements. We will first outline these requirements. We will then explain the redundancy types that can be exploited to satisfy the requirements.

The two common requirements in any image compression application are 1) the time taken to encode the image, and 2) the quality loss that occurs when the encoded image is reconstructed. Most recent applications are designed to minimize the second requirement. However, the main challenge comes when both requirements need to be minimized. This is challenging because it is difficult for an algorithm that takes minimal encoding time to provide a high reconstruction accuracy (and vice versa). In this thesis, we intend to have a good compromise between the image quality and the encoding time.

Keeping the two requirements in mind, two types of image redundancies can be exploited: numerical and perceptual.

Numerical redundancy is based on the numerical correlation of a pixel with its neighboring pixels. For example, when a pixel value is close but NOT necessarily the same as the pixel values within its immediate neighborhood, then this value can represent the neighboring pixel values. Please note that this redundancy is based on numerical values only. The perceptual significance of any of these pixel values are not taken into consideration. This could affect the image quality to a certain extent.

However, the exploitation of such a redundancy can reduce the encoding time significantly. On the other hand, when the correlation among neighboring pixels are carefully exploited, it can result in lossless compression. Lossless compression guarantees no numerical difference between the original and the decompressed image. This is useful in medical imaging applications where minute details can contain crucial information.

Perceptual redundancy is based on the observer's ability to perceptually differentiate between the original and another image e.g. the reconstructed image. That is, certain changes in small parts of the image do not affect the viewer's perception. Such redundancy is exploited by keeping only the perceptually significant pixel values. The image is reconstructed using only those values. It is highly likely that an image can have millions of colors out of which few thousand color pixels are selected as the representative color pixels. The process of representing those million colors using a few thousand color pixels can be very exhaustive. Hence, there is a possibility that the reconstructed image that looks subjectively the same as the original image might suffer from increased computational time. Furthermore, taking advantage of the perceptual correlation results in lossy compression. That is, there is a numerical difference between the original and the compressed image. This is useful in case of web images or live sport series transmission. In these cases, the transmission time is more critical than the quality, provided that no major glitch occurs during the transmission or within the image being displayed. It should be noted that the perceptually significant pixel values are extracted from a set consisting of pixel values that are perceptually close but NOT necessarily the same. Such a set is formed based

on some special metrics which are designed to compute the perceptually similar pixels in a numerical manner. In this case, one can say that the perceptual redundancy is a special case of the numerical redundancy.

In this thesis, our first compression algorithm exploits the numerical redundancy and is implemented on RGB color space. In this algorithm, a large image data set is mapped into a small image data set and groups of image data are encoded together. Such a quantization scheme is called vector quantization (VQ). Our second algorithm is implemented on CIEL*a*b* color space. It takes advantage of the perceptual redundancy in images. However, this algorithm does not follows the traditional vector quantization technique. Each of these algorithms is implemented such that a suitable compromise between image quality and encoding time is achieved. VQ is one of the popular lossy image compression schemes. There are four such popular schemes. These include DCT transform coding, wavelet based image compression and fractal compression. Each of them will be further explained in Chapter 2.

In summary, our thesis is primarily focussed on lossy color image compression using two novel VQ algorithms. The first technique takes advantage of the numerical correlation between image pixels. This technique includes the use of training images and VQ codebook. This technique can be applied to either grayscale images or color images. The second method takes advantage of the perceptual correlation between color image pixels. This technique does not include the use of codebook. Additionally, it can only be applied on color images. This is because perceptual similarity is computed between color pixels only. This will be further

explained in Chapter 4, Section 4.2.

1.3 Research goals

By carefully considering the numerical as well as perceptual redundancies for color images, we formulate two lossy color image compression techniques. We aim that each of these techniques should

1.3.1. result in reconstructed images with minimal artifacts

1.3.2. achieve a suitable balance between the computational time and the image quality

Each of these two methods directly extracts and transmits meaningful image information. Using this information, we examine if the above two conditions are satisfied. Our expectations are as follows.

1.3.1.1. With regards to Section 1.2, we expect the proposed algorithm that exploits numerical redundancy, to result in a better encoding time than our algorithm that exploits perceptual redundancy.

1.3.1.2. We expect our algorithm that exploits the perceptual redundancy to result in better image quality compared to our codebook generation algorithm.

1.3.1.3. Overall, we aim to design compression methods that have a better compression performance compared to existing lossy compression algorithms.

1.4 Proposed underlying principles

With the ongoing research and several benchmarked lossy compression algorithms, our area of work is quite broad. Our research began with the following goals, which steered our course of work.

1.4.1. Review process of existing methods

Our approach should build upon the existing benchmarked quantization algorithms. Hence, we began with understanding the limitations within existing methods. The novelty in our approach should lie in improving the existing shortcomings.

1.4.2. Robustness

The compression scheme should give good performance for several classes of color images. It should also have the potential of real time imaging / video transmission. Therefore, in such cases, existing algorithms that manually tune the parameters will not be considered.

1.4.3. Usefulness

As mentioned above, our approach should find a suitable compromise between the reconstructed image quality and the computational time. Thus, existing algorithms that extensively compromises one of these two will not be reviewed.

1.5 Thesis outline

Chapter 2 provides a brief background on the existing lossy image compression techniques. These include vector, quantization, DCT transform coding, wavelet based image compression and fractal image compression. Chapter 3 explains the limitations within existing vector quantization methods. Our innovative codebook generation algorithm is then explained. The experimental results are demonstrated as well. Chapter 4 explains the limitations within the algorithm proposed in Chapter 3 suggests few improvements. The second proposed algorithm is then explained along with the experimental results. Chapter 5 provides a summary and conclusion to this thesis.

Chapter 2

Literature review

As mentioned in Chapter 1, the two main requirements in any image compression application are image quality and encoding time. While image quality is almost always the most important requirement, in some applications, one needs to achieve an efficient balance between image quality and encoding time. There are numerous algorithms in which an efficient balance between the two has been achieved. Compression algorithms fall in two domains: lossless and lossy image compression. In this chapter, we only review the existing literature that falls within the domain of lossy image compression.

In this chapter, the four most active lossy image compression schemes are discussed. These include vector quantization, DCT transform coding, wavelet based image compression and fractal image compression.

2.1 Vector quantization

Quantization is a method of representing a large set of discrete image points by a much smaller set. This implies that the image could be approximated while maintaining its usefulness. The performance of VQ methods depend on the way the image signals are quantized. The quantization of image signals, in turn, is dependent on the choice of a codebook, the design of which is computationally demanding.

VQ methods partition each image within the training image set into R non-overlapping rectangular blocks. To construct a VQ codebook, assume the training image set consists of M images. Each rectangular block in an image consists of N pixels. Each of these blocks is converted into a vector *t* of length N. The collection of all the image vectors forms the training vector set, $T = [t_1, t_2, t_3, \dots, t_P]^T$ where t represents the vector of the rectangular block and *P=MxR*. The length of each vector is N, $t_i = (e_1, e_2, e_3, \dots, e_N)$ where e represents each element within the vector t_i. The influence of the size of training set on the quantization performance has been demonstrated quantitatively in [21]. From the training set, L vectors are extracted to form the initial codebook of size L i.e. $C = [c_1, c_2, c_3, \dots, c_L]^T$ where C represents the codebook of size LxN and $L < P$. Each vector c_i within the codebook is called a codeword. There are different ways to form the initial codebook. Some of these ways are explained later in this section. Once the initial codebook is formed, the codewords within the codebook are iteratively updated to form the final codebook. Different VQ algorithms follow different optimization processes to form the final codebook. Some of these optimization processes are explained later in this section. Such an optimization process requires several number of iterations. After each iteration, the distortion between the previous and the newly updated codebook is measured $D = \frac{1}{j}\sum_{k=1}^{j}\left\|c_k - c_{k(new)}\right\|^2$. If the difference between the current and the previously measured distortion is greater than a pre-defined threshold $\frac{D_{prev}-D_{curr}}{D_{prev}} > \epsilon$, the codebook is continued to be updated. Otherwise, the current codebook is set to be the final codebook. Once the final codebook is designed, the image is compressed and reconstructed as explained in the paragraph below.

The final codebook is placed at the encoder and at the decoder. To compress and reconstruct a test image IM, this image is partitioned into non-overlapping rectangular blocks of size $\sqrt{N}x\sqrt{N}$. Each block is converted into a vector im where $im = N$. Hence, the test image IM consists of several vectors, $IM = [im_1, im_2, im_3, \dots, im_i]$. The difference between the first image vector im_1 and each codeword $[c_1, c_2, c_3, \dots, c_L]^T$ within the final codebook C is computed using the Euclidean distance measure. The index j of the codeword c_j that yields the minimum distance from im_1 is transmitted to the decoder. Using the index j, the decoder extracts the codeword c_j from the codebook and reconstructs the image block corresponding to im_1 as c_j. The same process is repeated for each test image vector. It is evident from above that the test image block is reconstructed using the codeword in the codebook that is closest to the vector corresponding to that block.

Numerous efficient codebook design approaches have been developed. A comprehensive review of different vector quantization algorithms and comparison between them have been reported in [19]. There are two classical codebook design approaches that have been usually used as benchmarks. These are the K-means [17] and the LBG algorithms [18]. Both algorithms have the same codebook updating approach but differ in the way the initial codebook is formed. The LBG algorithm provides better reconstruction accuracy than the K-means algorithm.

Another classical approach that has shown better performance than the LBG algorithm is Kohonen's neural network algorithm [19]. A quantitative comparison between the two algorithms has been shown in [20]. The Kohonen's algorithm

attempts to incorporate essential biological mappings. The main difference between the two algorithms is that the Kohonen's algorithm can achieve good results regardless of how the initial codebook is formed. Another difference is their dependency on the number of training set vectors. The Kohonen's algorithm achieves good result using a smaller number of training set vectors than the LBG algorithm. This reduces the time for codebook generation in the Kohonne's algorithm.

Two recent VQ state-of-art approaches have been proposed [42]-[43]. These are the Kekre's Fast Codebook Generation algorithm (KFCG) [42] and the Kekre's Error Vector Rotation algorithm using Walsh sequence (KEVRW) [43]. Both algorithms significantly reduce the compression error compared to the K-means and the LBG methods. These algorithms employ a codebook design approach that is completely different. Also, the KFCG algorithm does not use the Euclidean distance measure. Few variations to the KFCG algorithm that reduce the computational cost have been proposed. These include the nearest neighbor search algorithm [44] and Partial Distortion search [45]. The KEVRW algorithm is completely different from the aforementioned algorithms, in terms of the codebook design and relies on the Walsh sequence to optimize the codebook, as will be explained later in this Section. There are few papers that use similar ideas as KFCG and KEVRW. These include the Hartley transform matrix [46], the Haar sequence [47], Slant transform [48] and Quick Sort [49].

The above VQ algorithms exploit numerical redundancies in an image. There are some quantization algorithms that exploits perceptual redundancies as well. One such algorithm [63] initiates with the conversion of an image from the RGB color

space into the CIELab color space. This image is partitioned into small areas. Each small area consists of perceptually identical color pixels. From each such area, a representative pixel is selected. The set of selected representative pixels is then reduced by only keeping the representative pixels based on the number of pixels in the area to which they belong to and distance between the areas. Once the representative pixels are selected, the approximation error is reduced using a clustering algorithm. This is done to refine the position of those representative pixels.

The KFCG and KEVRW algorithms along with the K-means and the LBG algorithms are explained below in detail.

2.1.1. K-means algorithm

Assume m images are available in the training set. The algorithm divides each image into n non-overlapping rectangular blocks, and converts each block into a vector. The set of all resulting mxn vectors forms the training set $[t_1, t_2, t_3, \dots, t_P]^T$, where t refers to a vector in the training set and P=mxn. The length of each vector is N, where N<<m. The length N and the number of codewords L (in the LxN codebook) are determined by the user. The design of the codebook starts by finding an initial codebook. The formation of the initial codebook is common to the majority of existing methods. To generate the initial codebook $[c_1, c_2, c_3, \dots, c_L]^T$, where c refers to a codeword and L<P, each of these L codewords is formed by selecting a random vector from the training set. This is followed by the codebook optimization process. This process is comprised of two steps. In the first step, the Euclidean distance between each codeword, c_v belonging to the initial codebook and each image vector in the training set t_g, g∈{1,2,…,P} is found. The vector t with the minimum

distance from c_v is categorized as belonging to c_v. The vector t is then deleted from the set t_g. This process is repeated for every codeword c_v. This whole process is repeated until T training vectors are categorized as belonging to each codeword.

In the second step, the average of the vectors belonging to each codeword c_v in the initial codebook is computed. The codeword c_v is then replaced by this average vector. In this way, the initial codebook is updated. After this process, the difference between the initial and the newly updated codebook is computed. If the difference is negligible, the algorithm is terminated; otherwise the whole process is repeated. Please note that each codeword in the codebook is an average of some vectors.

2.1.2. LBG algorithm

This algorithm is an extension of the K-means algorithm with a careful approach to initialization. In this algorithm, a splitting method is used to generate the initial codebook. This method is used so as to have a set of codewords within the initial codebook that are close to one another. This is followed by a codebook optimization process that is similar to K-means. The algorithm is initiated by the formation of a training set as described earlier. Once the training set is formed, the initial codeword, $c_1^{(1)}$ is first generated by taking the average of all the training vectors. This forms a one-level codebook, comprised of only one codeword $c_1^{(1)}$. This codeword $c_1^{(1)}$ is split to obtain two new codewords as follows. A fixed perturbation vector, ϵ which is between 0 and 1 is introduced in this method. The two new codewords are obtained as follows: $c_1^{(2)} = c_1^{(1)} + \epsilon$ and $c_2^{(2)} = c_1^{(1)} - \epsilon$. These two codewords are not added to the initial one-level codebook. The two newly generated

codewords form an initial two-level codebook, comprised of two codewords. Similarly, each of the newly generated codewords, $c_1^{(2)}$ and $c_2^{(2)}$ are split into two new codewords and four new codewords are generated. These four codewords forms the initial four-level codebook. The same process is repeated for each of these four codewords and an initial eight-level codebook is formed. This process is repeated till an initial L-level codebook, cb_0 of size L is formed. Now that we have an initial codebook, the codebook optimization step of K-means algorithm as described earlier is followed. This begins with the first step where the Euclidean distance is computed between each codeword c_j in cb_0 and each training set vector t_i. This is repeated until T training set vectors belonging to each codeword in cb_0 are found. The average vector of the T vectors belonging to each codeword c_j in cb_0 is computed. This average replaces each old codeword and thus a new codebook cb_1 is formed. The average distortion between cb_0 and cb_1 is computed. If the difference between the two is less than a pre-defined threshold, the algorithm terminates; otherwise, the whole process is repeated. Therefore, in the K-means and LBG algorithms, each codeword within final codebook is average of those vectors which are closest to the previous corresponding codeword.

2.1.3. KFCG algorithm

In the K-means and LBG algorithms, each codeword in the final codebook is formed as an average of T vectors in the training set. In the KFCG method the codewords are formed as averages of different numbers of training vectors. The training set is formed as described in Section 2.1.1. The first codeword c_1 in KFCG is taken as the average of all P vectors in the training set. The second and third

codewords are formed as averages of smaller subset of the P vectors. The fourth, fifth, sixth and seventh codewords are obtained as averages of even smaller subsets than those forming the second and third codewords and so on.

To obtain the second and third codewords, the training set is divided into two sets as follows: every first element t_{i1} of each t_i vector, where $i \in \{1,2,\dots,P\}$, in the whole training set is compared with the first element c_{11} of the codeword c_1. If $c_{11} > t_{i1}$, then t_i is placed in a set G_1 otherwise it is placed in a set G_2. The averages of the training vectors in G_1 and also of those in G_2 are obtained. These averages form the second and third codeword respectively.

The above process is repeated for vectors in G_1 and then in G_2. That is, within the group G_1, compare the second element of the second codeword c_{21} with the second element of each t vector in G_1 and divide these vectors in two subsets G_3 and G_4 depending on whether or not $c_{21} > t_{i2}$. Similarly G_2 is divided into two smaller sub-sets G_5 and G_6. Now four groups, G_3, G_4, G_5, and G_6 are formed. The codeword for each group is computed as the average of all vectors within that group. Each of these codewords is stored within the codebook. The aforementioned process is repeated for each group and more groups are formed. Updated codewords from those newly formed groups are stored within the codebook and the process is repeated until a codebook of user-specified size is reached.

2.1.4. KEVRW algorithm

This algorithm implements a concept in error correction whereby a sequence of redundant vectors, called error correcting sequence is arithmetically applied (e.g. by adding/subtracting) on an input vector, such that each element of the input vector can be recovered even if a number of errors are introduced during transmission or storage . Some error correcting sequences include Pseudo-Noise sequence [69], M sequence [60], Gold code [61] and Kasami code [62]. The error correcting sequence used in this algorithm is the Walsh sequence. These are generated from Walsh functions. A Walsh function of order N is defined as a set of orthogonal basis of the square-integrable functions denoted by $\{W_j(t); t \in (0,T), j = 0,1,\dots,N-1\}$, such that $W_j(t)$ takes the values *{+1, -1}*. Sampling a Walsh function at the center of the interval *(0, T)* generates *n* Walsh sequences when a codebook of size 2^n is required. Walsh sequences are error correcting sequences that encode input codewords into orthogonal codewords. In [43], Walsh sequences are used to split codewords to generate new codewords. The distance between the newly generated codewords and the training vectors are further used to generate another set of new codewords. This results in accumulation of error by the time the last codewords are generated. So even though the Walsh sequences can be used for Forward Error Correction (FEC), in this approach, they are used to generate codewords that represent error vectors. The first eight Walsh sequences are given as follows,

$$E = \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{matrix}$$

In this algorithm, the average of all vectors within the training set is taken as the first codeword, c_1. For codebook of user-specified size $L=2^n$ with length N, n Walsh sequences are generated.

In the first iteration, the first Walsh sequence is arithmetically added to c_1 to generate a second new codeword, c_1^1. The Euclidean distance is computed between each vector in the training set, $[t_1, t_2, t_3, \ldots, t_P]^T$ and c_1^1 to generate a new set of distance vectors represented by $[tc_1^1, tc_2^1, tc_3^1, \ldots, tc_P^1]^T$. The distance vectors are placed into a new group, G_1. Similarly, the first Walsh sequence is arithmetically subtracted from c_1 to generate a third new codeword, c_1^2. The Euclidean distance is again computed between each vector in the training set and c_1^2 to generate another set of new distance vectors represented by $[tc_1^2, tc_2^2, tc_3^2, \ldots, tc_P^2]^T$. These distance vectors are placed into another group, G_2. Both groups G_1 and G_2 have the same number of vectors.

The averages of the vectors in groups G_1 and G_2, are taken as the new codewords, c_2 and c_3. The second Walsh sequence is then added to c_2 to generate another new codeword, c_1^3 . The Euclidean distance is computed between c_1^3 and each vector in G_1, $[tc_1^1, tc_2^1, tc_3^1, \ldots, tc_P^1]^T$ to form a new set of distance vectors that forms another group G_3. Similarly, the second Walsh sequence is subtracted from c_2 and the

process is repeated where another group, G_4 is formed. The averages of the vectors in G_3 and G_4 form the new codewords c_4 and c_5, respectively. Similar process is repeated for the group G_2 to form groups G_5 and G_6 and more codewords are stored within the codebook. This process is repeated until a codebook of size specified by the user is achieved.

2.2 DCT transform image coding

Transform coding represents an image as a linear combination of basis images along with their corresponding linear coefficients. The transform encoding process, in general, consists of three steps. These steps are forward transformation, lossy quantization, and entropy coding.

Popular transform coding algorithms include Karhunen Loeve Transform (KLT) [50], [51] and Discrete Cosine Transform (DCT) [52]. An ideal transform coding algorithm should optimally decorrelate the image signals and should be computationally inexpensive. KLT algorithm achieves high decorrelation, however, the decorrelation process is computationally expensive. Due to the high computational cost, the KLT method is rendered impractical in many applications. This problem can be avoided by using an approximation for the KLT algorithm. DCT being computationally inexpensive is one such approximation whose performance efficiency reaches that of KLT. DCT is used in the Joint Photographic Experts Group (JPEG) [53] image compression standard which is described below.

2.2.1. JPEG

The RGB color image is converted into YCbCr color space. The Y channel represents pixel brightness. The Cb and Cr channels represents the chrominance information. Next, the Cb and Cr channels are downsampled by a factor of two in the horizontal and the vertical direction. This is because a human eye is more sensitive to change in brightness rather than change in color. Taking this sensitivity into consideration, the chrominance channels are downsampled. Then each of the following three steps are applied on each channel separately: the preprocessing step, the transformation step, the quantization step and the encoding step. In the preprocessing step, an image of size NxN is taken and partitioned into non-overlapping rectangular blocks of size MxM. A block of size 8x8 is commonly used. The value 127 is subtracted from each pixel value in each block. This centers the pixel values around 0. This is done because DCT works better on the pixel values ranging from -128 to 127 rather than 0 to 255. This is followed by the transformation step.

The transformation step is the key component in JPEG Image Compression Standard. In the transformation step, DCT is used. This step is applied on each image block. The underlying idea of DCT is to concentrate the large values within the image block to the upper left corner of that block. And the rest of the pixel values within the block will have small values (most of those values are close to zero).

This transformation begins with selecting an image block. A DCT coefficient matrix U is then applied on that block as follows

$$B = UAU^T \tag{1}$$

Here, U is the DCT coefficient matrix, A is the image block and B is the resultant DCT block. The size of the matrix U and the matrix A are the same. The row values within the DCT coefficient matrix U are spaced by evaluating the cosine at each value. Hence, the resultant DCT matrix B represents the image pixel values in terms of the weights of cosine functions at different frequencies. The reason behind using a matrix consisting of cosine components is that the cosine transformation does not consist of imaginary components. It also represents the frequencies that makes the image signal. For example, if the image signal is sharper, the weights with high frequencies will be more dominant. In the same manner, if the image signal is smooth, the weights with low frequencies will be more dominant.

In equation (1), matrix A is first multiplied with the DCT matrix U i.e. C=UA. This product is simply an application of U to each column of A. The resultant matrix is then multiplied by the transpose of the DCT matrix i.e. $B = CU^T$. Here, C is applied to each column of U^T. Next, we observe that the resultant DCT matrix B tends to store all the pixel values in few values and concentrates (pushes) them to the upper left-hand corner of the matrix. The remaining values are either zero or close to zero. The DCT step is invertible due to the knowledge of the coefficient matrix at the decoder. The image block can be recovered as follows, $A = U^T BU$. The resultant matrix B consists of irrational values (since the DCT coefficient matrix consists of irrational values). In order for these values to be efficiently encoded by Huffman algorithm, they need to be represented as integers. Hence, in order for the values within the matrix B to be represented as integers, a quantization step is implemented as follows.

The quantization step uses a quantization matrix. This matrix is of the same size as the resultant DCT matrix B. The quantization matrix consists of predefined values that are gives different weights to different coefficients. These weights are set according to the desired compression ratio. Each element within the DCT matrix is divided by the corresponding element within the quantization matrix. The quantization matrix scales down the coefficients within the matrix B to small values. However, the scaled down values are still irrational. These are converted to integer values as follows. The quantized irrational values within the matrix that are closer to 0 are converted to 0. The rest of the irrational quantized values are rounded to the nearest integers. This (quantization) step represents the block values with lower-precision approximations such that only few large coordinates are present and the rest are converted to zero. This step makes JPEG coding process not correctly invertible. This is because the quantized values can be recovered but the quantized values that are converted to zero and rounded cannot be recovered. This is followed by entropy coding (using the Huffman code) the resultant quantized image block using Huffman algorithm.

In order to reconstruct the image, the elements (encoded by Huffman algorithm) are first decoded. Each decoded quantized values is then multiplied with the corresponding element within the quantization matrix. This is followed by applying inverse DCT. Next, 127 is added to each element within the matrix. As the last step, the downsampled Cb and Cr channels are interpolated to their original size. These interpolated channels are then passed through a low-pass filter to smoothen the image. the three image channels are then merged to obtain the decompressed color

image. A problem with JPEG compression technique is that, partitioning the image into rectangular blocks results in blocky artifacts. Due to this, the image quality significantly starts deteriorating when the compression ratio goes higher than 25:1. This issue has been addressed in many studies [31-33]. However, images compressed using wavelets have resulted in better image quality even at high compression ratios.

2.3 Wavelet based image compression

The aim behind wavelet transform is to convert the image from spatial domain to frequency domain. The wavelet based image compression is comprised of the same steps as transform coding. The main difference lies in the transformation part where an invertible wavelet transform is used. The wavelet transform refers to image analysis at different scales and different resolutions. The image scale refers to the image size. As the image scale increases, the image shrivels (tightens). As the image scale decreases, the image expands. The image resolution refers to the frequency of the image. As the resolution increases, more amount of high intensity image information becomes visible. As the resolution decreases, the visibility of high intensity details decreases. Hence, the image decomposition involves two waveforms: one to represent the high intensity changes called the wavelet function Ψ and the other to represent the low intensity changes called the scaling function $\emptyset$.

One of the most popular wavelet transform is the Discrete Wavelet Transform (DWT) [54-57]. It has three basic architectures [34]: level-by-level, line-based and block-based. DWT is used within the lossy JPEG2000 Image Compression Standard

[58]. Like JPEG, this technique also consists of preprocessing, transformation, quantization and encoding steps. These steps are described as follows.

2.3.1. JPEG2000

JPEG2000 converts an image in the RGB color space into the YCbCr color space. Like JPEG, the two chrominance channels, Cb and Cr, are downsampled by a factor of two in the horizontal and vertical direction. Next, each of the above mentioned three steps are applied on each channel separately. In the preprocessing step, the number 127 is subtracted from each pixel value within the image matrix. This is done for the same reason as mentioned earlier in case of JPEG. This is followed by the transformation step.

In the transformation step, the Discrete Wavelet Transform with Daubecheis 9/7 filter is used. In this step, the image is first passed through a high-pass filter (H.P.F.) and a low-pass filter (L.P.F.), simultaneously, as shown in Fig. 2.1. Both filters are applied along the image columns. In this process, the image signal is split into two image sub bands, sb_1 and sb_2. The first sub band sb_1 is the high-pass filtered version of the original image. The second sub band sb_2 is the low-pass filtered version of the original image. The spatial bandwidth of each of these sub bands is half as compared to the original image. Both sub bands are then subsampled by a factor of 2 along the columns, as shown in Fig. 2.1. This results in sb_1' and sb_2'. Sub sampling by a factor of 2 means that, out of two consecutive pixel values, we keep only one of them and discard the other. This is done because both sub bands consist of redundant pixel values, and hence, one does not lose information significantly by removing

alternate pixel values. After this, the high-pass and the low-pass filters are applied along the rows of sb_1' and sb_2', as follows.

The sub band sb_1' is passed through high-pass filter ($H.P.F._1'$) and low-pass filter ($L.P.F._1'$), simultaneously, as shown in Fig. 2.1. Both filters are applied along the sub band rows. This results in two new sub bands, sb_3 and sb_4. Each of the two sub band has half the resolution as compared to sb_1'. This means that these resultant sub bands have one-fourth the resolution as compared to the original input image. The same step is repeated for sb_2'. This results in two sub bands, sb_5 and sb_6, each of which has half the resolution as compared to sb_2'. Hence, there are four resultant sub bands, sb_3, sb_4, sb_5, sb_6. Each of these sub bands are further downsampled by a factor of 2. This results in sub bands HH, HL, LH and LL. The nature of each resultant sub band is as follows. The first resultant sub band, HH, is derived by high pass filtering along the image column followed by high pass filtering along the rows of sb_1'. Thus it extracts the diagonal image edges. The second resultant sub band, HL, is derived by high pass filtering along the image columns followed by low pass filtering along the rows of sb_1'. This means that the vertical image edges are extracted. The third resultant sub band, LH, is derived by applying low pass filter along the image columns followed by applying high pass filter along the rows of sb_2'. Thus the horizontal image edges are extracted. The fourth sub band, LL, is derived by applying low pass filter along the image columns followed by applying low pass filter along the rows of sb_2'. Hence, LL consists of the low frequency image content. These four sub bands, HH, HL, LH, LL, are obtained in one iteration.

During the second iteration, similar sub band partitioning can be applied by taking any of the four sub bands as input image. However, further partitioning is only done on the LL sub band. This is because this sub band consists of the maximum image content compared to other sub bands. Hence, depending on the number of user specified iterations, at each iteration, the LL sub band can be further be partitioned into four new sub bands. This way of partitioning the LL sub band is called dyadic partitioning.

The underlying physical concept in DWT is that, at each iteration, smooth image information gets removed from the input sub band. This implies that the sub bands resulting from each next iteration are coarser as compared to the input image / sub band. Further, on each sub band, the quantization function is applied. For each sub band, different quantization step size is computed. Like JPEG, the resultant quantized irrational values are either rounded off or converted to zero. These values cannot be recovered. The quantized values are further entropy coded. In the entropy coding step, EBCOT (Embedded Block Coding with Optimized Truncation) [37] scheme is used.

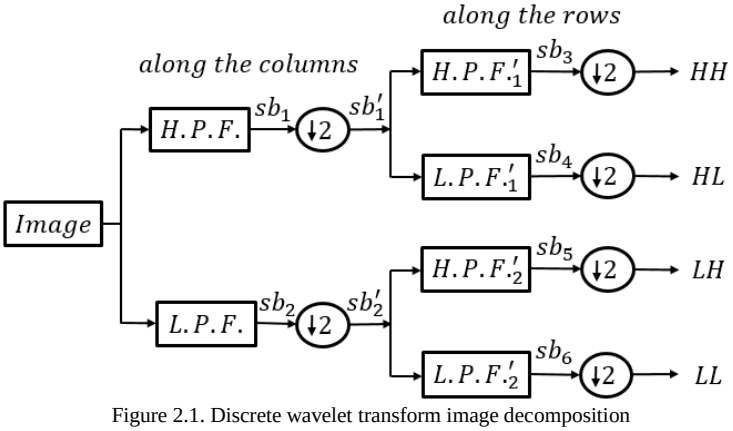

Figure 2.1. Discrete wavelet transform image decomposition

Other wavelet based image compression algorithms consists of the Embedded Zerotree Wavelet (EZW) [35], Set Partitioning In Hierarchical Trees (SPIHT) [36] and EBCOT. Of these, the EZW algorithm generates the image bits in order of their importance resulting in a sequence of binary decisions. This is one of the most efficient compression algorithm which exploits the concept of self-similarity within images to predict the significant information.

2.4 Fractal image compression

The underlying concept of this method is to find whether or not if one big part of an image when transformed becomes similar to a smaller part of the same image. If similarity is found, then at the decoder, the large image part can be made to transform into the small image part. The underlying reason in such cases is that, if similarity is found, this means that the information content within the small image part is redundant. Hence, fractal compression exploits image redundancy through self-transformability. The compression and reconstruction scheme builds on the partitioned iteration function system (PIFS). This is the underlying theory of fixed point and collage theorems [38].

The fractal compression process starts by partitioning the image of size NxN into large non-overlapping blocks of size DxD. These blocks are called domain blocks. The domain blocks are further divided into four non-overlapping small blocks of size RxR. These blocks are called range blocks. The size of each domain block is larger than the size of the range block. The number of range blocks is equivalent to $(N/R)^2$. And the number of domain blocks is equivalent to

$(N - 2R + 1)^2$. Now, the first range block is selected. Each domain block is affine transformed. Affine transform means that the block is rotated, skewed, stretched, translated and scaled. Also, the size of the transformed domain block is reduced to match the size of the range block. Next, each transformed domain block is compared with the selected range block (search block). When a particular transformed domain block closely resembles that range block, that transformed block is selected to represent the selected range block. The transform parameters are then quantized and encoded. The same process is repeated for each range block. Hence, the algorithm mainly consists of image partitioning into domain and range blocks, finding the best range block match for each transformed block, quantization and encoding of the transform parameters.

This compression technique results in exhaustive search for the best match between the transformed blocks and the search blocks. In order to speed up this process, several algorithms have been developed. One such approach [39] categorizes the search blocks. In this manner, instead of searching the entire set of search blocks, only a subset is searched. Another approach speeds up the algorithm by arranging the search blocks in a tree structure to direct the search [40]. The algorithm has also shown improvement when the transformed blocks are approximated [41]. This completely eliminates the need to search for an approximation.

Chapter 3

Codebook design for vector quantization using local image information

In Chapter 2, we discussed the four popular lossy image compression schemes. Out of the four methods, VQ has the ability to exploit statistical correlation between neighboring vectors [87], [19]. On the other hand, the extent to which the neighboring transform coefficients are correlated, is complex and depends on the image data [30]. Also, VQ has the potential to achieve good quality reconstruction at a high compression ratio [84]. As observed in Chapter 2, the VQ methods heavily depend on the codebook employed. While reviewing the existing VQ algorithms, we came across certain drawbacks in their codebook generation procedures which motivated us to propose an improved VQ codebook generation algorithm.

Our main aim behind designing a new VQ codebook generation algorithm is to reconstruct an image such that it is not only similar to the original image but also appears sharper comparatively, even at high compression ratios. Out of the four popular lossy compression schemes (described in Chapter 2), we propose an improved algorithm that is based on VQ because of the following reason. In VQ, several aspects (that are implemented) are employed in the formation of an efficient codebook. Existing codebook generation algorithms have utilized most of these aspects in certain

ways to achieve the desired result. While going through the existing VQ algorithms, we found that there are still several ways to empirically modify these aspects in order to form a codebook, such that it results in a better reconstruction quality, even at a high compression ratio. We will first explain the limitations of existing VQ algorithms. We will then explain the main differences between our proposed VQ algorithm and the existing VQ algorithms.

Within the existing VQ methods (discussed in Chapter 2), the final codewords are formed by taking the averages of specific vector groups. For example, in the K-means and the LBG algorithms, each codeword is repeatedly updated until the termination condition is satisfied; the updated value is the average of the vectors with the shortest Euclidean distances to the codeword. In the KFCG algorithm, a codeword is updated, until the termination condition is satisfied, such that its value is the average of specific vector groups that belong to a subset of a group originating from the training set. In the KEVRW algorithm, a codeword is updated, until the termination condition is satisfied, such that its value is the average of the difference between each vector within the training set and the codeword. This does affect the image quality to a certain extent. In this Chapter, we propose a new codebook generation algorithm that reconstructs the image more accurately from the beginning, without having to use iterative updating. This is the main novelty in our approach. That is, the image is partitioned into blocks and each element of the codeword is extracted directly from each block to form the codeword. This implies that a single codeword is formed from one image block. In the rest of the Chapter, we refer to ‘each element of the codeword’ as a ‘representative pixel’. These pixels represent the

less commonly occurring information within the image block, from which they are extracted. Thus, unlike existing methods, the codebook is formed directly without the use of an initial codebook and in a non-iterative fashion. There are three differences between the proposed codebook generation algorithm and the existing VQ algorithms. Those three differences (which include the one explained above) are as follows:

1) In our proposed VQ algorithm, the image is partitioned into hexagonal blocks instead of traditional rectangular blocks. This is because it has been qualitatively demonstrated in [85] that the LBG vector quantization algorithm yields less artifacts when hexagonal instead of rectangular blocks are used. In this Chapter, we demonstrate this quantitatively. The use of hexagonal blocks yields less artifacts compared to when rectangular blocks are used, due to the following reason. When hexagonal blocks are used, the resulting lower perceptual significance of the artifacts is due to the fact that the receptors of the human eyes follow a hexagonal structure [86]. More specifically, rods and cones on the retina are densely packed such that the shape of each cone cell and rod cell resembles that of a hexagon. So when looking at an image, the image signal corresponds to hexagonal cells as it reaches the retina [64]. Thus the artifacts that occur as a result of hexagonal partitioning are less significant to the human eye than those resulting from rectangular partitioning.

2) Our proposed algorithm extracts representative pixels directly from each hexagonal block. Thus unlike the existing VQ methods, we do not convert each hexagonal block into a vector and iteratively update it. The manner in which the representative pixels are extracted will be elaborated in Step 3.

3) In our proposed VQ algorithm, a training image block is much larger in size compared to a test image block. This is unlike the existing methods, where the size of a training image block is equal to the size of a test image block. This is not the case with our proposed method due to the following reason. Our main aim is to reconstruct the image such that it appears shaper than the original image. This implies that we want the reconstructed image to exhibit a high contrast. In a high contrast image, the intensity difference between any two adjacent pixels is higher compared to an image with a low contrast. In order to have a high intensity difference between two adjacent pixels, the (representative) pixels are extracted from a large image block. Within this block, each of these pixels is extracted from a set of pixels. The set of pixels represent less commonly occurring image information. The rationale behind it, is that when the representative pixels are extracted from a set of pixels (belonging to a large image block), such that each of these extracted pixels represents the less commonly occurring image information, it becomes "highly likely" that the intensity difference between any two adjacent (representative) pixels will be high. Also since the representative pixels are extracted from a large training image block and those pixels are used to reconstruct a small test image block, the contours of the original image are preserved. This, unfortunately, results in the image details not being preserved. This is one of the drawbacks of our proposed VQ method which will be taken into consideration for improvements in Chapter 4. The above has been further explained and a mathematical proof of the same has been provided in Section 3.1.

Hence, the proposed codebook generation algorithm is explained below in the following three steps.

Step 1: In this step, the image is partitioned into non-overlapping **hexagonal blocks**. In the following steps, one codeword will be extracted from each block. Each element of this codeword will be a pixel that represents the pixels covered by a small window within the block. This small window slides insides the hexagonal block.

Step 2: In this step, the size of the sliding window is determined. The window is slid such that it divides the hexagonal block into small overlapping areas. Unlike the traditional sliding window, which slides by one pixel in the horizontal direction and then in the vertical direction, this window slides by more than one pixel in the horizontal direction.

Step 3: At each location of the sliding window, a pixel that represents the unusual image features is extracted. This is based on the mean of all pixels within the sliding window and the median of a subset of these pixels.

The details of these steps are now explained:

Step 1:

This step describes how to obtain the area of the hexagonal blocks.

The proposed algorithm partitions every image in the training set into non-overlapping hexagonal blocks. Let the total number of training images be Z and the total number of all the hexagonal blocks in all the training images be J. The hexagonal blocks are oriented as shown in Figure 3.1.

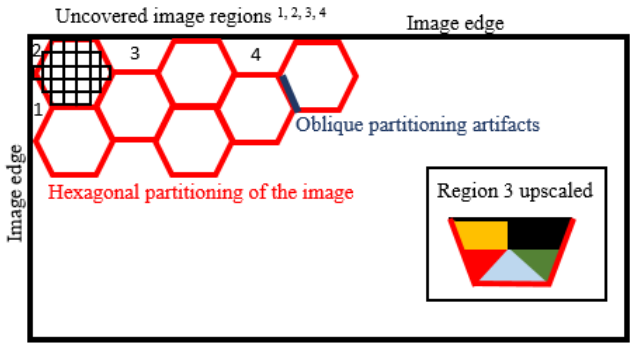

Fig. 3.1. Image partitioned using hexagonal blocks

The area A_{hex} of the hexagonal block we used (with side length of l_{hex} pixels), is similar to the area of the rectangular block A_{rect} used in existing methods. Traditionally, a rectangular block of size 16x16 has been adopted (A_{rect}=256). Hence, we used a hexagonal block with side length l_{hex}=10 pixels and area A_{hex}=260.

Due to the shape of the hexagonal blocks, four out of the six sides of each hexagon are oriented diagonally. Thus, any artifacts that may result from such a partitioning would be diagonally oriented. Such artifacts are perceptually less significant than those resulting from rectangular partitioning [64]. However, an issue that arises due to the hexagonal shape of the block is related to the uncovered regions (e.g. regions 1, 2, 3 & 4 in Fig. 3.1) along the boundary of the "test" image that do not belong to any hexagonal block. Such regions are reconstructed at the decoder which will be explained later in this Section.

Step 2:

This step describes how the size of the sliding window is selected.

Once the image is partitioned into hexagonal blocks, a *kxk* window is slid across each block so that it covers the maximum number of pixels in the block. The

window is slid horizontally by more than one pixel as this will affect the compression rate favorably.

We will show how the window is slid in the top half of the block i.e. how the sliding window covers the pixels in the top half; covering the pixels in the lower half of the block will be done in a similar fashion. As shown in Fig. 3.2a and Fig. 3.2b, the window starts at the top-left corner of the block, slides horizontally along the first row, once the end of that row is reached, the window then starts at the leftmost pixel in the nearest adjacent row in the block, and so on. As soon as the window reaches the middle row(s) of the block, it will not be able to cover the two corner pixels (shaded regions 1 & 2), unless the window size is one pixel only.

At each position of the window inside the block, one representative pixel is extracted from the window. The set of these pixels forms a codeword that represents this hexagonal block. The length of the codeword is thus equal to the total number of windows that can fit inside the block. The number of windows is in turn related to two aspects: 1) the size of the sliding step as the window moves horizontally i.e. the number of pixels that would belong to the region that gets overlapped by two consecutive horizontal windows, and 2) the window size.

With regard to the first aspect, as the window slides horizontally, we need to achieve a good compromise between the image quality (which suffers if the number of pixels that belong to the overlapping consecutive windows is small) and reconstruct a sharp image (which would not be possible if this number is large). For this purpose, we (empirically) fixed the number of pixels so it would be equal to or less than half

the window size in pixels. Once the window is slid horizontally and reaches the end of the first row within the block, the window slides vertically by one pixel only to the next row. The rationale is to have every pixel (or as many as possible) that lies at a hexagonal block boundary to belong to at least one window. This makes the codewords longer, but overall the resulting rate will be favorable as will be seen from the results in Section 3.2.

In terms of the window size, a smaller window size would result in a long codeword and thus affect our goal of reconstructing a sharp image. On the other hand, when the window size is large, three issues arise that negatively affect the image quality. First, in each window, it will be difficult to find one pixel that can well represent this large number of pixels within the window. Second, the number of windows that a pixel, which is close to a hexagonal block boundary, belongs to would be less than the number of windows that the rest of the pixels would belong to. As a representative pixel is extracted from each window, this would result in a more accurate reconstruction of the areas near the block center compared to those near the block boundaries. This affects the image quality. It should be noted that the number of windows a pixel would belong to increases as the window size increases. This particular issue is related to the first aspect mentioned above. Third, there are some pixels within the hexagonal block that would not belong to any window. This also degrades the image quality. In order for the sliding window to cover the maximum number of pixels in the hexagonal block it is slid within, the length of the window side in pixels is set to be an even number, if the length in pixels of the hexagonal block side is even, as shown in Fig. 3.2a. Also, the side length of the window in pixels

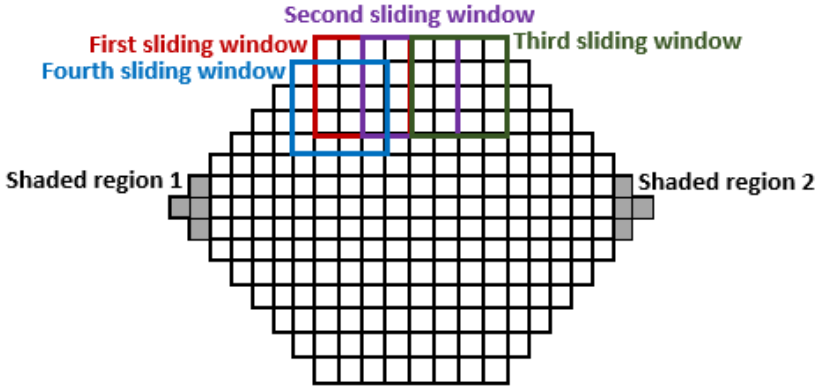

Fig.3.2.a Movement of sliding window within hexagonal block with even pixels side length

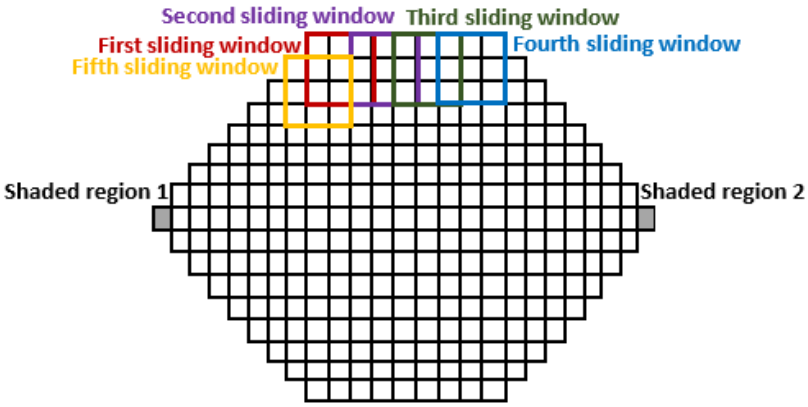

Fig.3.2.b Movement of sliding window within hexagonal block with odd pixels side length

is set to be odd if the block side length is odd as shown in Fig. 3.2.b. In terms of the window size, we empirically found that a good compromise between the two window sizes is to have the window side length in pixels to be equal to or less than half the hexagonal block side length in pixels i.e. to have $k \leq l_{hex}/2$ as is the case in Figs. 3.2a. and Fig. 3.2b.

For illustration purpose, two cases are presented in Fig. 3.2a. and Fig. 3.2b. In our experiments, however, the side of the hexagonal blocks used was l_{hex}=10 pixels.

Figures 3.2a. and Fig. 3.2b. represent hexagonal blocks with an even and odd

number of pixels on each side respectively. Each cell in both the Figures represents one pixel. We determine k using the above $k \leq l_{hex}/2$ rule. In case of Fig. 3.2a.,the $k = \frac{8}{2} = 4(even)$ i.e. window size is selected to be 4x4. In case of Fig. 3.2.b, k has to be less than 4.5, which is a fractional value and hence not acceptable. Taking into consideration the fact that the block length in pixels is odd, the window length in pixels is reduced till an odd and acceptable window size i.e. 3x3 is reached. In case of Fig. 3.2.a., the window is slid by 2 pixels horizontally. The same also applies to Fig. 3.2b in this case. In both cases, the window slides by one pixel in the vertical direction.

Step 3:

This step describes the proposed codeword extraction procedure.

From each hexagonal block, one codeword is extracted. Each element in the codeword is extracted from each location of the sliding window. The codeword is of length K+2 (114+2=116 in our algorithm) where K is the maximum numbers of possible sliding windows that can fit inside the hexagonal block, and the other 2 elements in the codeword are to represent the pixels at the two corners of the hexagonal block, that cannot be covered by the sliding window.

In order to determine the representative pixel at each location of the sliding window, we aim to select a pixel (in each such window) that best represents the less common image features within the window. These features are usually represented by high intensity changes such as sharp transitions between two regions of different intensities or image edges. Hence, a representative pixel is defined as a pixel that is

NOT similar to the majority of the pixels in the window it belongs to. There are two reasons behind selecting these image pixels. The first reason stems from the sensitivity of the human visual system to the position of high intensity changes within the image [65]. The second reason is that, as mentioned in the beginning of this Chapter, since these pixels exhibit high contrast, hence, when these pixels are selected to reconstruct any given test image, the reconstructed image is expected to be sharp then the original image. This has been mathematically proved in Section 3.1.

To find such a pixel, we obtain the mean of the pixels within each window. The pixels are then divided into two classes, those whose intensities are higher than the mean and those whose intensities are lower than the mean. The pixels in the class with less number of pixels are denoted as the representative pixels of the window (this is because they are less similar to the majority of the pixels in that window). Then a single representative pixel v is obtained from the set of representative pixels. In our implementation, we selected the median of the set of representative pixels, to represent the window. The median is selected because the sum of the distances between the median and the other pixels within the set is minimum. This implies that the median pixel is more similar to the pixels present within the set.

The above process is repeated for each sliding window in each hexagonal block h_l. The collection of these median values in h_l forms a codeword c_1 in the codebook. In other words, the codeword c_1 is a concatenation of the K median values

Due to the hexagonal shape of the block, the two shaded regions of the training blocks as shown in Fig. 3.2a. and Fig. 3.2b. at the horizontal left and

horizontal right corners of the block are not covered by the sliding windows. At each of these two corners, the median of the uncovered pixels are computed. These two medians are concatenated with the *K* representative pixels. Hence, the number of pixels in each codeword is *K+2*. The whole process is repeated for each hexagonal block h_l in the training set and hence *J* codewords are generated, $[c_1, c_2, c_3, \ldots, c_J]^T$. The collection of these codewords forms the $Jx(K+2)$ codebook, where *J* represents the total number of all the hexagonal blocks in all the images in the training set i.e. the size of the codebook.

In order to examine the improvement offered by using the hexagonal blocks only (i.e. had we used hexagonal blocks with a traditional VQ method), we apply the LBG algorithm but with instead of using the conventional rectangular blocks, we partition the image into non-overlapping hexagonal blocks. The hexagonal blocks are converted into vectors to form the training set and the codebook is formed in the same way as in the LBG algorithm described earlier.

We then study the improvement offered by our proposed codeword formation approach only, but now using rectangular blocks instead of hexagonal ones to partition the images. Each such block is partitioned into *kxk* overlapping rectangular windows and the proposed codeword formation step is applied to the windows.

Once the codebook is generated, encoding and decoding of the "test" image in our algorithm differs from the existing methods in two aspects only. First, at the encoder, as mentioned earlier in this Chapter, the hexagonal block size used for partitioning the "test" image is not the same as the size used when the "training" images were partitioned. The test image block size is equal to the length of a

codeword. Since the length of the codeword is 116, the hexagonal block size of the test image in our algorithm has a side length of 6 pixels. Hence, at the decoder, a codeword reconstructs a hexagonal block of size 116. Second, due to the hexagonal block shape, certain portions along the test image boundary would not belong to any block. Our algorithm does not encode these regions. They are simply extrapolated from the reconstructed image using the corresponding regions within the nearest hexagon. For example, in our experiments, each such region is divided into smaller regions as shown in the inset of Fig. 3.1 for region 3. All the pixels belonging to a specific row (in a sub-region represented by orange, black, red or green color) or a specific column (in the blue sub-region) are assigned the same value. This value is the median of the pixels belonging to the same row or column of the immediate neighboring hexagonal block. As long as these regions are small in size, it was found that the aforementioned reconstruction approach does not have any significant negative affect on image quality. For this purpose, we empirically found that the image quality is not significantly affected if the hexagonal block $l_{hex} \leq 14$ pixels.

3.1 Quantitative proof as to why the representative pixels give good results

As mentioned earlier, it is highly likely that the consecutive representative pixels exhibit high contrast. Hence, if we place them together, we expect the image to appear sharp than the original image. A sharp image exhibits high contrast. Ideally, the contrast should be high such that the contours within the image appears more pronounced. It should not be so high so that the image content appears unnatural. We

now demonstrate why the selected representative pixels reconstructs the image at the decoder in a sharp manner. We carry out the demonstration using the image of the girl shown in Fig. 3.3. We place the extracted representative pixels (obtained as described in our proposed VQ algorithm) together as shown in Fig. 3.4. We compare this Figure, in terms of its sharpness, with three other Figures of the same size. The pixels in those three Figures represent 1) average of every three consecutive pixels (in the horizontal direction) computed from the original image, as shown in Fig. 3.5, 2) first of every three consecutive pixels (in the horizontal direction) extracted from the original image, as shown in Fig. 3.6, and 3) median of every three consecutive pixels (in the horizontal direction)s extracted from the original image, as shown in Fig. 3.7.

TABLE 3.1
COMPARISON OF THE FOUR IMAGES WITH REPRESENTATIVE PIXELS PLACED TOGETHER, IN TERMS OF AVERAGE CONTRAST PER PIXEL

Images where pixels represent	**Average Contrast per pixel**
Less common image information	10.7604
Median	9.5139
First of three every three consecutive pixels	9.1168
Average	8.5518

Fig.3.3. Original image

Fig.3.4. Each pixel within this image is the representative pixel (as described in our VQ algorithm)

Fig.3.5. Each pixel within this image represents the average of every three consecutive pixels computed from the original image

Fig.3.6. Each pixel within this image represents the first pixels of every three consecutive pixels extracted from the original image

Fig.3.7. Each pixel within this image represents the median of every three consecutive pixels extracted from the original image

We measure the sharpness of an image in terms of average contrast per pixel as follows. An image pixel is selected. The summation of differences between that pixel and its immediate neighboring pixels is computed. The resultant summation is divided by the number of immediate neighboring pixels. This gives the average difference between an image pixel value and its immediate neighboring pixel values. In this manner, the average difference between each image pixel value and its immediate neighboring pixel values is calculated. This is followed by taking the average of all the resultant average differences.

From Table 3.1, one can conclude that Fig. 3.4 is the sharpest. Fig. 3.7 is less sharp than Fig. 3.4 but it is sharper than Fig. 3.5 and Fig. 3.6. This is followed by Fig. 3.6 and then Fig. 3.5. Now, we will explain this difference in image sharpness.

Fig. 3.4: As mentioned earlier, when the average of the set of pixels within a sliding window is taken, that value represents the balance between the large values and the small values within the window. The values falling above the average forms one class and the values falling below the average forms the other class. The representative pixel is then extracted from the class that consists of less number of pixels. The extracted representative pixel will then be the one that **occurs less commonly within that window**. Now, when we extract the representative pixel from a consecutive

window, that representative pixel will also be the one that **occurs less commonly within that window**. Since the two consecutive representative pixels occur less commonly within the two consecutive windows, it becomes highly likely that there will be a high contrast between these two pixels. If there is high contrast between consecutive pixels, the image appears sharp.

Fig. 3.5: In this Figure, the original image is downsampled by taking the average of every three consecutive pixels. The resulting downsampled image appeared the most blurred compared to the other three downsampled images. This is because the average value represents the balance between the large and the small values within the set of values from which the average is computed. Since the average value represents a balanced value (which is usually a new resultant value), the resulting image is expected to be blurred. Also, the average does not deal well with pixel values that vary extremely.

Fig. 3.6: In this Figure, the image is downsampled by selecting the first pixel out of every three consecutive pixels. The image is blurred but not as compared to Fig. 3.5. The only reason would be because Fig. 3.6 is reconstructed using pixels extracted directly from the image. This is unlike Fig. 3.5 where the pixels are average values. These average values, more often, might not be equal to any of the pixel values within the set of values from which the average is computed.

Fig. 3.7: In this Figure, the original image is downsampled by taking the median of every three consecutive pixels. The resulting image is quite sharp compared to Fig. 3.5 and Fig. 3.6. The median value is the value in the middle of the dataset. That

means this value would represent the commonly occurring values and does not get affected by the values that vary extremely within the set. The only concern when using the median pixel as a representative pixel from the complete set of pixels would be that the high intensity pixel values that a significant infrequently occurs might get ignored.

3.2 Experimental results

The proposed algorithm is implemented using MATLAB R2013a on Intel Core i7-4500U, 2.40 GHz, 8.00 GB RAM. The aforementioned codebook generation algorithms are tested on six different classes of RGB images. These images are categorized as follows. The Mandrill[1] image consists of dense texture and frequent high intensity changes. The foreground and background in the Bear[2] image consist of high amount of texture and lighting variations. Brandy Rose[3] consists of gradual changes in most part. This is the same as in the Peppers[1] image which consists of uniform intensity in most parts and variations in lighting. Lena[1] consists of dense texture, gradual and sharp intensity changes. Pills[4] mostly consists of uniform intensities along with few abrupt intensity changes. Each of these images is of size 256x256x3. Each class has a set of four training images.

The comparison of the seven methods 1) K-Mean 2) LBG, 3) KFCG 4) KEVRW 5) the proposed algorithm 6) the LBG algorithm that uses hexagonal blocks and 7) the proposed codeword formation but using rectangular instead of hexaogonal blocks are shown in Tables 3.2, 3.3, 3.4 and 3.5 for codebooks of size 64, 128, 256,

[1] Courtesy of the Signal and Image Processing Institute at the University of Southern California
[2] Copyright photos courtesy of Robert E. Barber, Barber Nature Photography (REBarber@msn.com)
[3] Copyright photo courtesy of Toni Lankerd
[4] Copyright photo courtesy of Karel de Gendre

and 512 respectively, on color images. The aim is to compare the methods in terms of the MSE and the computational time for the same compression rates. However it is extremely difficult to compress the images so that the same exact compression rate is obtained for all seven methods. Therefore the compression rates corresponding to the seven methods were very close for each codebook size. For example, for codebook size 128, the rates were 73.9% 75.5%, 75%, 75.6%, 76.1%, 73.3% and 74.1% for the 1) K-Mean 2) LBG, 3) KFCG 4) KEVRW 5) proposed algorithm 6) LBG using hexagonal blocks and 7) the proposed codeword formation using rectangular blocks. It can be observed that the compression rate for the proposed method was selected to be the highest, yet despite this fact, this method yielded the best MSE.

For methods 1, 2, 3, 4 and 7, the images were divided into non-overlapping rectangular blocks of size 16x16(area =256). Methods 5 and 6 used hexagonal blocks with l_{hex} = 10 pixels (area =260). The sliding window inside each hexagonal block in methods 5 and 7 was of size 4x4.

From the tables, one can conclude the following:

1) the MSE and the computational time are the highest for Bear image while MSE is the lowest (best) for Brandy Rose and the computational time is the lowest (best) for Lena.

2) the MSE and the computational time are the highest (worst) for K-means and the lowest (best) for the proposed algorithm. This is regardless of whether it used hexagonal or rectangular blocks.

3) the use of hexagonal blocks reduces the MSE but increases the computational time.

4) the proposed codeword formation algorithm contributes more towards reducing the

MSE and the computational time than the use of hexagonal blocks.

Hence it can be concluded that in terms of MSE and computational time, the use of hexagonal blocks results in better performance than rectangular blocks (as demonstrated by the performance of LBG and our method). It can also be deduced that computing the codewords by directly extracting the local image information improves the reconstruction accuracy as well as reduces the computational time (as demonstrated by the performance of the proposed method whether it uses hexagonal blocks or rectangular ones).

Table 3.6 shows the average percentage improvement in terms of MSE and computational time brought about by three methods: the proposed algorithm, the LBG algorithm with hexagonal blocks and the proposed codeword formation method using rectangular blocks over existing methods: the K-means, LBG, KFCG and KEVRW algorithms. The overall average percentage is computed by averaging the values of all the methods across the six images and across the four codebook sizes. From this table, we conclude that the LBG algorithm with hexagonal blocks is better than LBG with rectangular blocks in terms of MSE. Also the codeword formation process (in the proposed algorithm) contributes more towards the better performance than the contribution offered by hexagonal blocks. Even though the images compressed by the proposed algorithm had the highest compression rates, this method still provided the most improvement compared to existing methods, in terms of MSE and computational time.

Figure 3.8 and Fig. 3.9 show the variations in the MSE and the computational time respectively, for similar compression rates, using one test image, for each of the

six image classes, for all seven methods and for varying codebook sizes: 64, 128, 256, and 512. From Figure 3.8, we conclude that the MSE is highest for the K-means algorithm and lowest for the proposed method. Also, MSE is the highest for codebooks of size 64 and lowest for codebooks of size 512. From Figure 3.9, it can be concluded that the computational time is the highest for the K-means algorithm and the lowest for the proposed method. It is also the lowest for codebooks of size 64 and reaches the highest value for codebooks of size 512.

Figure 3.10 shows the six test images[5], which are also of size 256x256x3.

Figure 3.11 shows the training images[5] used to obtain the codebook of the "Face" images class. This codebook is used for the test image (Lena). Please note that the number and content of training images have been experimentally determined such that the resulting codebook generates good reconstruction quality.

Figure 3.12 shows the original and the reconstructed Lena image (test image). This image is used to compare K-Means, KFCG, and the proposed algorithm for a codebook of size 512, along with their MSE respectively.

[5] Images obtained from the University of Southern California Image Database: http://sipi.usc.edu/database/database.php?volume=misc

TABLE 3.2
COMPARISON OF THE EIGHT METHODS IN TERMS OF MSE AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR CODEBOOK SIZE: 64

Codebook size 64												
Parameters	Mandrill		Bear		Brandy rose		Lena		Peppers		Pill	
	MSE	time	MSE	Time	MSE	time	MSE	time	MSE	time	MSE	time
K-Means	118.95	1.12	121.32	1.36	92.50	1.17	113.48	0.84	99.66	0.91	95.12	1.13
LBG	101.32	0.93	106.48	1.01	81.85	1.02	94.54	0.71	83.78	0.84	82.53	1.04
KFCG	90.69	0.88	98.56	0.66	79.93	0.86	81.29	0.65	79.17	0.63	78.88	0.87
KEVRW	83.25	0.72	81.21	0.54	66.55	0.52	77.71	0.50	72.35	0.52	63.09	0.66
Proposed	65.13	0.43	68.42	0.57	47.94	0.46	51.12	0.37	59.86	0.46	46.19	0.47
LBG with hexagonal blocks	92.40	0.94	103.06	1.07	77.91	1.08	91.24	0.76	78.35	0.92	77.70	1.10
Proposed method with rectangular blocks	79.08	0.60	73.85	0.52	58.14	0.48	65.45	0.43	65.56	0.49	55.84	0.51

TABLE 3.3
COMPARISON OF THE EIGHT METHODS IN TERMS OF MSE AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR CODEBOOK SIZE: 128

Codebook size 128												
Parameters	Mandrill		Bear		Brandy rose		Lena		Peppers		Pill	
	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time
K-Means	93.56	1.66	119.31	1.75	80.34	1.44	98.73	1.05	96.83	1.07	83.58	1.49
LBG	87.43	1.07	98.75	1.43	76.89	1.29	87.87	0.91	83.03	0.93	79.81	1.32
KFCG	78.81	0.90	86.80	1.31	68.77	0.95	75.33	0.82	74.44	0.85	71.22	1.30
KEVRW	71.78	0.75	73.29	0.97	54.06	0.89	64.82	0.53	69.95	0.62	60.73	1.05
Proposed	59.55	0.51	63.68	0.62	43.48	0.62	40.19	0.41	56.85	0.51	43.80	0.77
LBG with hexagonal blocks	83.70	1.09	89.22	1.45	72.31	1.32	81.35	0.97	72.88	0.98	74.86	1.37
Proposed method with rectangular blocks	66.11	0.58	68.85	0.71	50.35	0.66	53.25	0.47	61.90	0.58	52.98	0.80

TABLE 3.4
COMPARISON OF THE EIGHT METHODS IN TERMS OF MSE AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR CODEBOOK SIZE: 256

Codebook size 256												
Parameters	Mandrill		Bear		Brandy rose		Lena		Peppers		Pill	
	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time
K-Means	85.98	1.89	104.26	1.92	71.29	1.61	85.30	1.31	85.60	1.34	76.64	1.65
LBG	73.56	1.22	91.83	1.76	68.19	1.44	76.53	1.11	74.90	1.16	75.85	1.41
KFCG	70.85	1.35	82.60	1.52	55.30	0.98	67.46	0.70	67.51	1.08	69.00	0.76
KEVRW	62.81	0.80	65.42	1.02	43.73	0.82	55.48	0.65	66.23	0.97	52.72	0.73
Proposed	44.92	0.73	50.29	0.85	32.22	0.63	36.85	0.54	49.89	0.62	38.24	0.62
LBG with hexagonal blocks	70.33	1.26	83.54	1.78	63.87	1.48	64.50	1.20	65.87	1.27	70.77	1.49
Proposed method with rectangular blocks	57.98	0.78	58.23	0.91	38.84	0.72	43.29	0.58	58.55	0.69	47.57	0.68

TABLE 3.5
COMPARISON OF THE EIGHT METHODS IN TERMS OF MSE AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR CODEBOOK SIZE: 512

Codebook size 512												
Parameters	Mandrill		Bear		Brandy rose		Lena		Peppers		Pill	
	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time	MSE	time
K-Means	69.51	2.33	95.28	2.25	54.48	2.17	74.94	1.87	72.85	2.02	69.16	1.94
LBG	65.60	2.06	85.10	2.06	50.36	2.02	60.83	1.56	63.19	1.95	56.85	1.78
KFCG	58.82	1.60	74.78	1.72	43.26	1.71	53.36	1.43	55.34	1.78	45.70	1.55
KEVRW	54.83	1.34	57.47	1.31	35.88	1.66	48.54	1.27	49.72	1.39	36.59	1.35
Proposed	35.55	0.98	38.89	1.10	21.56	1.19	29.87	0.88	37.05	1.00	25.86	0.92
LBG with hexagonal blocks	61.02	2.11	79.41	2.13	45.67	2.12	51.52	1.61	58.01	1.98	52.29	1.82
Proposed method with rectangular blocks	46.69	1.05	49.58	1.16	27.33	1.24	40.48	0.93	42.39	1.07	29.46	1.04

TABLE 3.6
THE AVERAGE PERCENTAGE IMPROVEMENT OF THE PROPOSED ALGORITHM; THE LBG ALGORITHM WITH HEXAGONAL BLOCKS, THE PROPOSED ALGORITHM WITH RECTANGULAR BLOCKS OVER K-MEANS, LBG, KFCG, AND KEVRW, IN TERMS OF MSE AND COMPUTATIONAL TIME FOR SIX RGB IMAGES

Improvements of proposed algorithm over	K-Means	LBG	KFCG	KEVRW
MSE	49.62%	42.98%	36.33%	25.94%
computational time	56.41%	49.25%	39.29%	24.44%
Improvements of LBG with hexagonal blocks over				
MSE	18.38%	7.61%	-3.06%	-16.66%
computational time	10.90%	-3.60%	-19.42%	-35.25%
Improvements of proposed codeword formation with rectangular blocks over				
MSE	40.16%	32.27%	24.37%	12.03%
computational time	52.56%	44.78%	33.93%	17.78%

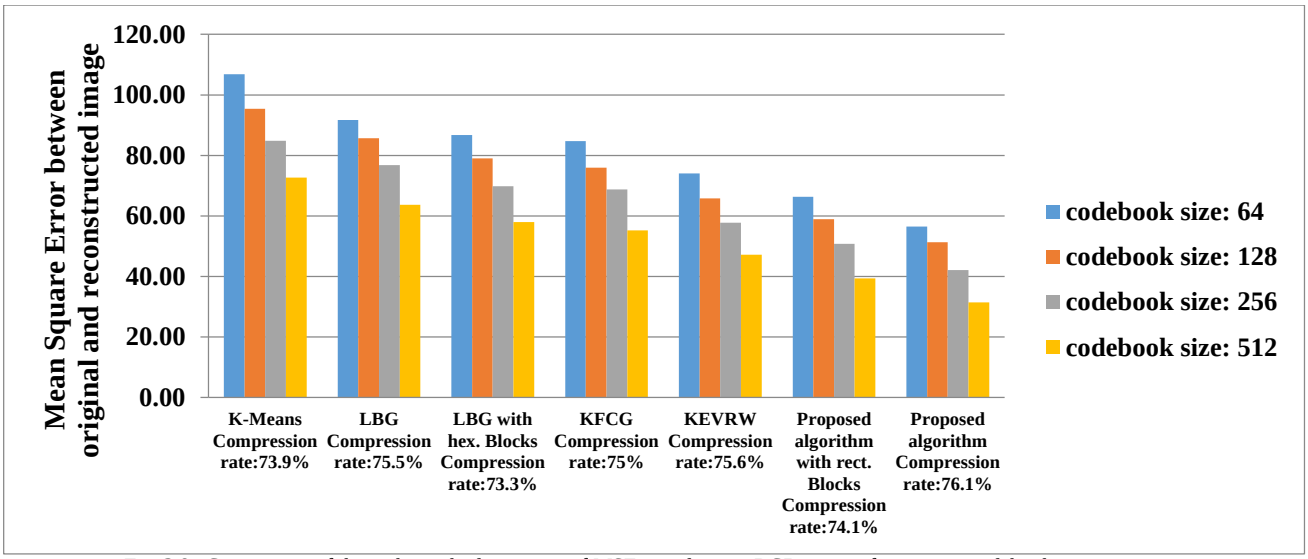

Fig. 3.8. Comparison of the eight methods in terms of MSE tested on six RGB images for varying codebook size

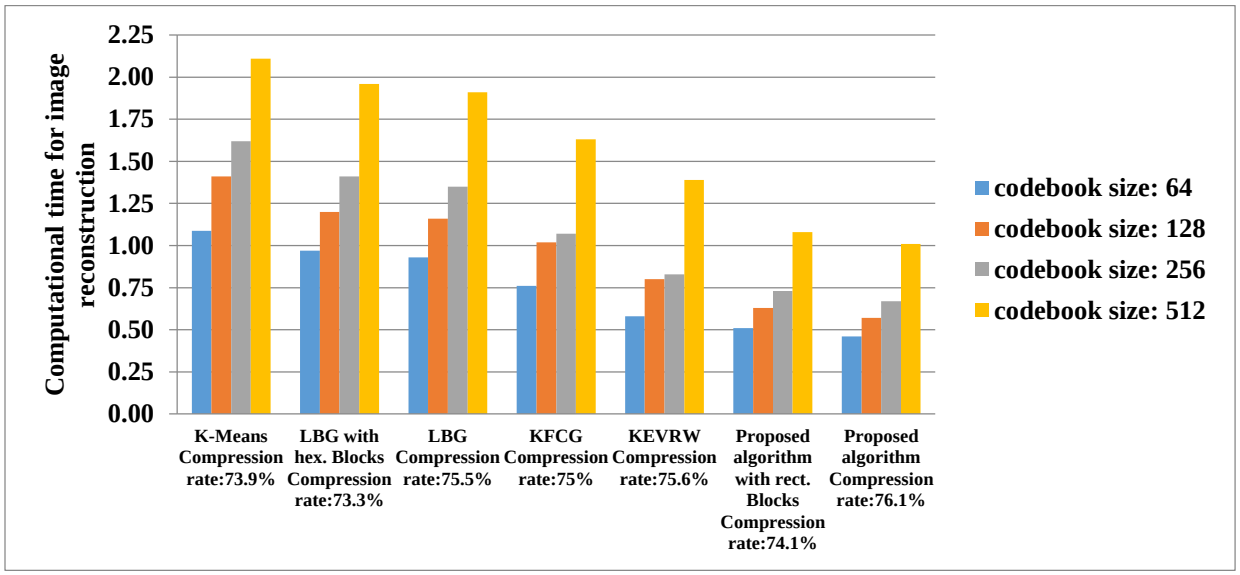

Fig. 3.9. Comparison of the eight methods in terms of computational time tested on six RGB images for varying codebook size

Fig. 3.10. Six original color test images, each of size 256x256x3

Fig. 3.11. Training images used to obtain codebook that was then tested using the Lena image

Original image (Zoomed)

K-Means, MSE: 74.94

KFCG, MSE: 53.36

Proposed Algorithm, MSE: 29.87

Fig. 3.12. Original and reconstructed Lena image (test image) for K-means, KFCG and the proposed method for codebook of size 512

Chapter 4

Color image quantization using perceptual information

In Chapter 3, we presented a new vector quantization compression method which we call the codebook generation algorithm. From the many aspects of this algorithm, there are three, which if modified or replaced by different procedures, would further improve the image quality. These three aspects relate to: 1) the set of infrequent pixel values from which a representative pixel is selected, 2) the use of codewords to reconstruct the image, 3) the use of training images to form the codebook. These three aspects along with the manner in which they can be modified or replaced with better ones, are discussed as follows:

1) In constructing the codewords for our codebook generation algorithm, a representative pixel is selected (from a small region) that represents the less commonly occurring information in this region. This information is used to reconstruct the image. This results in reconstructed images that appear sharper than the original image. This, however, has the disadvantage that the image details are not well preserved. The image quality can be further improved if the "information" represented by the pixels selected from within the set is modified. We can modify this information such that the reconstructed image is truer to the original image. The information is modified such that the set would consist of pixels that are similar, as

determined from a particular point of view (this particular point of view is based on how the image is perceived and will be explained later in this Section). This set would not be biased towards a particular image region (image region exhibiting high or low intensity changes). The selection of a representative pixel from the set consisting of similar pixels basically implies that the pixels that are redundant, as determined from a particular point of view, are not coded.

2) At the decoder of our codebook generation algorithm, the codeword which is nearest to a particular test image vector, is selected to reconstruct the test image region corresponding to that vector. This means that the particular image region is reconstructed using a codeword that is "approximately" equal to that region. This implies that not each pixel value within that region might be similar (equal or very close) to the corresponding pixel value within the codeword. This still results in an overall good image quality. However, if each image pixel value within the small image region is reconstructed with a value that is similar to the former pixel value, the image quality would improve even further. More specifically, we do not have to reconstruct the small image region using an approximately equivalent vector. We can reconstruct each pixel within that region using a representative pixel that is similar, as determined from a particular point of view, to the pixel being reconstructed. Such a technique eliminates the use of training images and codebook i.e. it is not a VQ method in the traditional sense.

3) In case of our proposed codebook generation method, in order to obtain a good result, we need to carefully select the training images such that the brightness of most of those images is the same as the brightness of the test image (that is being

compressed and transmitted). The reconstruction quality would be adversely affected otherwise. This problem of taking the image brightness into consideration can be solved by eliminating the use of training images. This implies that instead of forming a codebook from the training images and further using that codebook to compress and reconstruct a test image, the test image can be reconstructed using the information extracted from the same image.

We aim to incorporate the above two modifications to form a new compression algorithm, that is also lossy. This is done to improve the image quality compared to that resulting from the codebook generation algorithm proposed in Chapter 3. There are five differences between the proposed codebook generation algorithm and the newly proposed algorithm. Those five differences (which include the two aspects explained above) are as follows:

1) The method proposed in this Chapter is implemented on the CIEL*a*b* color space instead of the RGB color space used in the former method. We will explain in Section 4.2 as on why we use a different color space.

2) This method divides the image into non-overlapping rectangular blocks instead of non-overlapping hexagonal blocks which is used in Chapter 3. This difference will be further elaborated in Section 4.5.

3) In this algorithm, a representative pixel represents a set of pixels that are redundant as determined from a particular point of view. This particular point of view is the human perception. Hence, the redundant pixels are “color pixels” which are perceptually similar to one another. Perceptual similarity of color pixels will further

be explained in Section 4.1. This is unlike the former method where the representative pixel represents a set of less common image information.

4) In the present method, as mentioned above, we also deal with "color" pixels. This means that at each stage of this method, the three channels are processed for every pixel, in a sequential manner. This is unlike the former method, which like the traditional VQ algorithms, forms a separate codebook for each image channel (color). Each codebook then encodes the corresponding test image channel. Using the encoded information, each test image channel is then reconstructed at the decoder. The three reconstructed image channels are then merged.

5) In the former algorithm, an index of the codeword that is closest to a test image region is transmitted. The codeword corresponding to that index is then used to reconstruct that image region. This is unlike the present method, where the following three information items that are related to a block are transmitted and used for reconstruction. The representative pixel values that are selected from an image block are transmitted. The index of the block from which these values are selected is also transmitted. The index of pixel values within that block from which the representative pixels are selected, are transmitted as well. At the decoder, the pixel values corresponding to the sent indices are reconstructed using their representative pixel values. This will be explained in Section 4.6.

The latter method is however similar to (but not the same as) the former method in terms of how the representative pixel is extracted. This will be explained in Section 4.5.

4.1 Key idea

The key idea behind our newly proposed algorithm is to compress the image by removing “perceptual color redundancies”. The perceptual color redundancies are removed by transmitting the minimal number of pixels that represent perceptually similar color pixels. Two pixels are said to be perceptually similar if the human eye perceives the color information (e.g. luminance, chrominance) of both pixels to be the same or very close to one another. Also, if any color information within a pixel is changed by a certain amount, the human eye should perceive that change by the same (corresponding) amount. The perceptual similarity between two pixels can mathematically be determined only in a perceptually uniform color space.

The term perceptually uniform and the choice of color space is explained in Section 4.2. In Section 4.4, we discuss the metric that will be used to mathematically determine the perceptual similarity between two color pixels. In Section 4.5, our proposed compression algorithm is explained. This algorithm does not require the use of a codebook.

4.2 Perceptually uniform color space

Perceptual uniformity in a color space has a similar definition as perceptual similarity. A color space is said to be perceptually uniform if the perceived difference between two color pixels in a color space is proportional to the Euclidian distance between them. Also, in a single color pixel, small changes in one parameter, such as brightness, should be perceived proportionally by the human eye [81].

In RGB color space, the difference between two color pixels is mathematically not proportional to how the human eye perceives the difference [66]. This is true for CMYK and HSV color spaces as well [76]. The color space which exhibits perceptual uniformity is the CIEL*a*b* color space [76], [77]. Hence, the algorithm we propose In this Chapter is processed in the CIEL*a*b* color space.

4.3 CIEL*a*b* color space

The CIEL*a*b* color space consists of three channels L*, a*, and b*. The L* channel corresponds to the perceived brightness. Its value ranges from 0 (black) to 100 (white). The a* channel corresponds to green-red values. And the b* channel corresponds to blue-yellow values. The two colors within the a* and the b* channels oppose each other. So, both the color channels are based on the facts that colors cannot be both green and red or both blue and yellow. Each channel ranges form positive to negative values. Within the a* channel, the positive values indicate the amounts of red and negative values indicate the amounts of green. Similarly, within the b* channel, the positive value indicate the amounts of yellow while the negative values indicate the amounts of blue. For both the channels, zero indicates a neutral gray color. Both the channels range from either $-100\ to + 100$ or $-128\ to + 127$. However, in our implementation, instead of the traditional channel ranges as stated above, the range for each channel varies from 0 (L*: black, a*: green and b*: blue) to 255 (L*: white, a*: red and b*: yellow). This is because we implemented our algorithm in MATLAB. In MATLAB, the channel values are represented using unsigned bit integer value. This, by definition, cannot be negative [88]. The asterisk after each channel is placed to distinguish this color space from the Hunter's Lab

color space [82]. These channels are related to the X, Y, Z trichromatic values [71] as follows:

$$L = 116\, f\left(\frac{Y}{Y_n}\right) - 16$$

$$a = 500[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)]$$

$$b = 200[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)]$$

Here, $f(\mu) = \begin{cases} \sqrt[3]{\mu} & \mu > 0.008856 \\ 7.787\mu + \frac{16}{116} & \mu < 0.008856 \end{cases}$

where $X_n = 0.95047$, $Y_n = 1.00000$ and $Z_n = 1.08883$ are the XYZ trichromatic values representing the reference white light.

4.4 Metric to determine the perceptual similarity

The perceptual similarity between two colors is computed using a metric that is specific to the CIEL*a*b* color space. Some of the customized CIEL*a*b* metrics includes CIE76, CIE94, CIEDE2000 and CMC metrics [68]. Numerous studies have been conducted to demonstrate how accurately the different metric systems, specific to CIEL*a*b* color space, perform in determining whether one color pixel is perceptually similar to another color pixel [69], [78], [79], [80]. One such study [4] carried out a visual data comparison for the different CIEL*a*b* color space metrics. In this study, the CIEDE2000 color metric [78] outperformed the other metrics in terms of its accuracy in characterizing the perceptual color similarity. It performed slightly better than the Color Measurement Committee (CMC) metric [72].

However, the CIEDE2000 metric is computationally complex while the CMC metric is computationally simpler. Hence, taking into consideration the computational complexity, the CMC color difference model is used in our algorithm. CMC is the Color Measurement Committee of the Society of Dyers and Colourists (UK) widely used in the textile industry to match bolts of cloth [83].

The base formula of the CMC (l:c) metric used in our algorithm which determines if two color pixels (L_1^*, A_1^*, B_1^*) and (L_2^*, A_2^*, B_2^*) are perceptually similar is given by ΔE^* as follows:

$$\Delta E^* = \sqrt{\left(\frac{\Delta H^*}{Sh}\right)^2 + \left(\frac{\Delta L^*}{l.Sl}\right)^2 + \left(\frac{\Delta C^*}{c.Sc}\right)^2}$$

Two color pixels in CIEL*a*b* color space are determined to be perceptually similar if the ΔE^* value of the two pixels is less than one [67]. Here, ΔC^* is the chroma difference:

$$\Delta C^* = \sqrt{A_1^{*\,2} + B_1^{*\,2}} - \sqrt{A_2^{*\,2} + B_2^{*\,2}}$$

ΔL^* is the lighting difference:

$$\Delta L^* = L_1^* - L_2^*$$

ΔH^* is the hue perceptual difference:

$$\Delta H^* = \sqrt{2(\left(\left(\sqrt{A_1^{*\,2} + B_1^{*\,2}}\right)\left(\sqrt{A_2^{*\,2} + B_2^{*\,2}}\right)\right) - ((A_1^*)(A_2^*)) - ((B_1^*)(B_2^*)))}$$

l and c are weight factors that can be changed to vary the effect of lightness and

chroma, respectively. Usually $l = 1$ and $c = 1$, then CMC (l:c) is written as CMC (1:1). In case the lightness and chroma are varied, the values should be such that the $l: c$ ratio is equal to 2:1. In this case, CMC (l:c) is written as CMC (2:1). Sh, Sl and Sc are semi-axes corresponding to hue, chroma and lightness represented as follows.

$$Sl = \begin{cases} \dfrac{0.040975L_1^*}{1 + 0.01765L_1^*} & L_1^* \geq 16 \\ 0.511 & L_1^* < 16 \end{cases}$$

$$Sc = \frac{0.0638\sqrt{A_2^{*\,2} + B_2^{*\,2}}}{1 + 0.0131\sqrt{A_2^{*\,2} + B_2^{*\,2}}} + 0.638$$

$$Sh = Sc(FT + 1 - F)$$

Here, $F = \sqrt{\dfrac{(\sqrt{A_2^{*\,2}+B_2^{*\,2}})^4}{(\sqrt{A_2^{*\,2}+B_2^{*\,2}})^4+1900}}$ and

$$T = \begin{cases} 0.56 + |0.2\cos(168 + h^*)| & 164^o < h^* < 345^o \\ 0.36 + |0.4\cos(35 + h^*)| & otherwise \end{cases}$$

where, $h^* = \tan^{-1}(\frac{B_2^*}{A_2^*})$

The ΔE equation corresponds to an ellipsoid shape of equal perceptual distances which represents a difference in color sensation [7]. The accuracy of the CMC metric in representing the average perceived color difference is about 65% [70]. This means that there is 35% chance that the two color pixels that are determined to be (or not to be) perceptually similar to one another, is not true. The representative pixel is computed by taking this error bar into consideration as explained in Section 4.5.

We again mention that two color pixels are “perceptually similar”, then both the pixel values are perceptually equal or very close to one another. That is, two perceptually similar color pixels might appear the same or slightly different.

4.5 Proposed compression algorithm

As discussed at the start, the proposed algorithm in this chapter mainly differs from the proposed method that uses a codebook, in two aspects. The two aspects are 1) a representative pixel is selected from a set of perceptually similar pixels instead of a set of less common pixels, and 2) instead of reconstructing the whole pixels of the small image region simultaneously using a codeword from a codebook, each pixel value within the small image region is reconstructed using a representative pixel. As mentioned in Chapter 2, vector quantization is one of the many popular lossy image compression techniques. Some of the other commonly used lossy compression techniques includes JPEG, JPEG2000, and fractal compression. The proposed algorithm in this Chapter does not use a codebook. Its performance is compared with these three methods along with proposed method (in Chapter 3) that use a new codebook generation algorithm. Below, we briefly summarize the discussions in Chapter 2 on each of these three techniques and highlight the step within each that makes them lossy.

In case of JPEG, an image is partitioned into small non-overlapping image blocks. For each image block, DCT is applied as follows, $B = UAU^T$ where U is DCT coefficient matrix and A is the small image block. Since the DCT coefficient matrix U is known, this step becomes reversible. However, the values of the coefficients within

the resultant DCT matrix B are mostly irrational. In order for these coefficients to be efficiently encoded, they need to be quantized and entropy encoded (by Huffman coding). The values are quantized. In the quantization step, each value within the DCT matrix is divided by a predefined value. The resultant irrational quantized values within the matrix that are closer to zero are converted to zero. The rest of the irrational quantized values are rounded to the nearest integers. Hence, the values near zero that are converted to zero and those rounded to the nearest integers cannot be recovered. This quantization step is similar to that used in JPEG2000 lossy compression algorithm. In JPEG2000, the entire image tile is transformed into a number of different image sub bands using Daubechies 9/7 filter. Each image sub band is quantized in a similar manner as described above. As in JPEG, the resultant irrational quantized values are converted to zero and rounded to the nearest integers which cannot be recovered.

In case of fractal compression, an image is partitioned into non-overlapping rectangular blocks (domain blocks). These blocks are further partitioned into four non-overlapping small blocks (range blocks). A range block is selected. Next, each domain block is affine transformed. Each transformed domain block is then compared with the selected range block to see if a particular transformed domain block closely resembles a range block. The address of the range block that closely resembles that of the transformed domain block is transmitted along with the domain block and the affine transform components. At the decoder, the range block is reconstructed using the corresponding transform block pixel values.

As observed from the above, the numerical format of the resultant quantized image matrices in case of JPEG and JPEG2000 are changed so that they can be easily encoded. Such a change (the rounding of pixel values or conversion to zero) cannot be reversed. Hence, these values forever lose their true identity. This leads to a loss in image content thus affecting the image quality. In case of fractal compression, a transform block that is closest to a range block reconstructs that range block. In this case, each pixel value within the transform block may or may not be equal to the corresponding pixel value within the block being reconstructed. Such an approximate reconstruction process does affect the image quality to a certain extent. Hence, in our proposed algorithm, we reconstruct each image pixel using a representative pixel that is perceptually similar to that pixel. This is explained below in the following steps.

Step 1: The image is divided into non-overlapping rectangular blocks. Each image block is converted into a vector.

Step 2: In every vector, the CMC (1:1) metric is used to determine the perceptually similar color pixels. Based on the results obtained from this metric, this vector is divided into a number of sub vectors (groups). Each sub vector consists of perceptually similar color pixels. The number of sub vectors is equal to the number of perceptually dissimilar color pixels present in that vector.

Step 3: From each sub vector of each block, a representative pixel is extracted. The representative pixel is extracted in a similar (but not the same) manner as that extracted in our codebook generation algorithm in Chapter 3.

Step 1:

The test image (of size *NxN*) is converted from the RGB color space to the CIEL*a*b* color space. Each image channel i.e. *L**, *a** and *b**, is partitioned into non-overlapping rectangular blocks of size *MxM*. Unlike our codebook generation algorithm, we do not divide the image into hexagonal blocks due to the following reason. In the proposed codebook generation algorithm, all pixel values within a small hexagonal image block are reconstructed together using a codeword. Such a reconstruction may result in artifacts at the boundaries of the hexagons. Diagonal artifacts have the advantage of not being as significant to the human eye as compared to vertical or horizontal artifacts. However, in our newly proposed algorithm, the pixels are reconstructed separately. Such a reconstruction would not result in partitioning artifacts. Hence, there is no need to divide the image into hexagonal blocks as there will not be diagonal artifacts (that are not significant to the human eye) to take advantage of. So for simplicity, we divide the image into rectangular blocks instead of hexagonal ones. Also, we divide the image into small sized blocks instead of large sized block. This is because when the information is formed from small blocks, we save more number of bits as will be explained in Section 4.6.

Once the test image of size *NxN* is divided into rectangular blocks of size *MxM*, as shown in Fig. 4.1, the total number of blocks into which the image is partitioned will be *J* where $J = N*N/M*M$. During the partitioning of the image into rectangular blocks, we ensure *M* to be a multiple of *N* so that the total number of image blocks in a particular row or a particular column i.e. *N/M*, is an integer. Each image block is composed of three image block channels i.e. *L**, *a** and *b** channels.

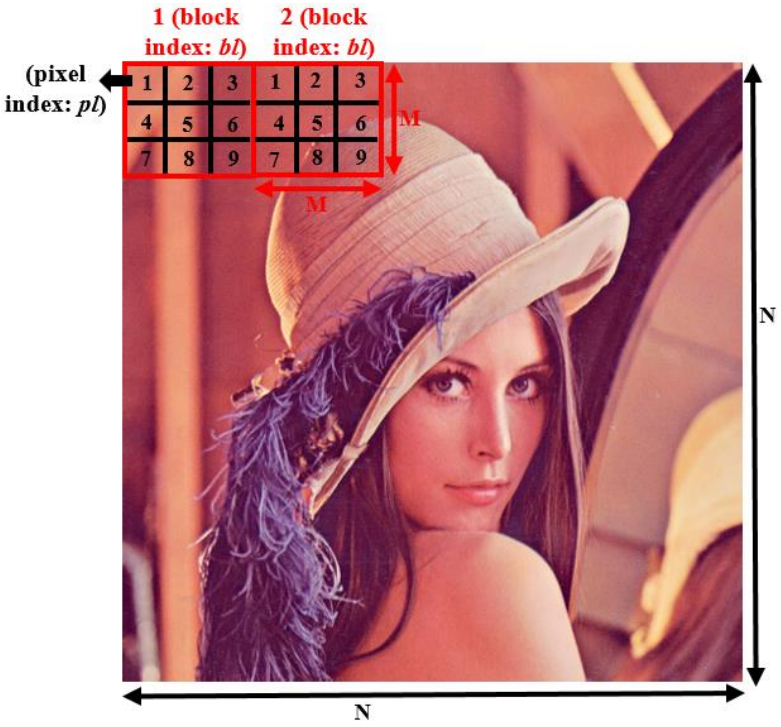

Fig. 4.1. Illustration of image dimension, block index and pixel index

Each such block is assigned an index *bl* where $bl \in \{1,2,\ldots,J\}$ as shown in Fig. 4.1. This index *bl* is assigned for reconstruction purposes as will be explained in Section 5. The total number of pixel values within each such block is *s* such that $s = M*M$. Furthermore, each pixel within each image block is assigned an index *pl* where $pl \in \{1,2,\ldots,s\}$ as shown in Fig. 4.1. This index *pl* is also assigned for reconstruction purposes as will be explained in the Section 4.6. For each channel, each image block is converted into an image vector. This implies that each block within each channel is converted into a vector. Hence, for each image channel, we have *J* vectors. This implies that for each image, we have $3*J$ i.e. $3*((N/M)*(N/M))$ vectors ('3' because one image has 3 channels). For each of these $3*J$ image vectors, the CMC (1:1) metric is used to determine the perceptually similar color pixels.

Step 2:

We will first provide an overview of this Step. The detailed implementation will be provided next.

For each image block, the CMC (1:1) metric is used to determine the perceptually similar color pixels. Based on the results obtained from the metric, numerous groups, each consisting of perceptually similar color pixels, are formed from each block. From each group, a representative color pixel is extracted. The extraction of the representative color pixel will be explained in detail in Step 3.

We now explain this Step in detail. The first image (color) vector is selected. Each pixel within this vector is assigned a label 1. The first image (color) pixel from the vector is selected. Please note that each image pixel consists of three pixels corresponding to L*, a*, b* channels. The selected image (color) pixel becomes a reference pixel. The label of the reference pixel is changed to 0. Next, the perceptual similarity between the reference image pixel and the rest of the (color) pixels within the first image vector is determined using the CMC (1:1) metric. If any of the image pixels is determined to be perceptually similar to the reference image pixel, the label of that particular image pixel is changed to 0. Next, within the first image vector, only those image pixels whose labels are 0 are selected and placed together in a new image sub vector (group). Also, these image pixels are deleted from the first image vector. The size of the newly formed image group is equal to the number of image pixels that are perceptually similar to the reference image pixel.

In the same manner, the above process is repeated again and a new group is formed where each pixel is perceptually similar to a newly selected reference pixel. More new groups are formed from the first image vector until all image pixels are deleted from that vector. The number of newly formed groups from the first image vector is equal to the number of perceptually dissimilar image pixels present within that vector. In the same manner, groups are formed from each image vector corresponding to each image block. Hence, one can safely conclude from this Step that the algorithm is exhaustive and the resultant computational time is quite high.

Step 3:

So far, the image is divided into vectors *J* and every vector is divided into a group of pixels. Each group is an image sub vector where every pixel is represented by three entries corresponding to its three (L*, a* and b*) components. We then extract a representative pixel from each group. There are two conditions for selecting this representative image pixel. We first outline these two conditions and then explain the process of extracting the representative pixel.

The two conditions are as follows. First, as mentioned before, there is 35% chance that any two image pixels that are determined to be perceptually similar, might not be true. This implies that each pixel in a group would be 1) an image pixel that is perceptually the same as the reference image pixel; 2) an image pixel that perceptually differs by a small amount from the reference image pixel, or 3) an image pixel that is not perceptually similar to the reference image pixel (it happens to be in the group due

to 35% error probability). We do not want the representative vector to belong to those mentioned in the third class above.

The second condition is as follows. The representative pixel is a "color" pixel. This means, a representative pixel must have three components, one from each L*, a* and b* channel. If for each channel in every group, we select a representative pixel but these selected pixels do not correspond to the same image pixel, then the combination of the three channel pixels would result in a different color. Hence, in this algorithm, the representative pixel is extracted from the L* channel only. Once the representative pixel is extracted from the L* channel, the a* and the b* entries corresponding to the L* value will be selected. The representative pixels are selected based on the L* channel since our eyes are more sensitive to changes in brightness rather than changes in color [73], [74], [75].

The representative image pixels are selected as follows. Within a group, the entries corresponding to the L* channel are sorted. The average of the sorted L* values is computed. The pixel values above the average form one class and the pixel values below the average form another class. The number of pixels in each class is computed. The class having the larger number of pixels is selected to be the set of pixels from which the representative pixel is selected. The class we selected in this algorithm is different from the class selected in our codebook generation algorithm. This is why we mentioned earlier that the representative pixel extraction approach we use in this method is similar "but not the same" as the one used in our codebook generation algorithm in Chapter 3. Here, we select the class with the larger number of pixels because we do not want the representative pixel to belong to the third class

mentioned earlier in this Step. Since such pixels occur less frequently i.e. there is 35% chance of their occurrence, they are more likely to fall within the class having a smaller number of pixels.

It is highly likely that the class with the higher number of pixels might consist of color pixels that perceptually differ from one another by a small amount. Hence, we select the representative pixel value as follows. From the class having the larger number of pixels, the pixel value that corresponds to the median within the L* channel is selected as the representative pixel of the group. Thus, the values within the a* and b* channels corresponding to the selected value within the L* channel are automatically chosen. As mentioned in Chapter 3, we select the median pixel value since it is more similar to the pixels that are present within the class. In the same manner, the representative image pixel from each group is extracted. We mention again that the number of representative image pixels is equal to the number of perceptually dissimilar color pixels within an image block. This significantly affects the computational time but is much favourable in terms of the image quality as can be observed in Section 4.7. Other than an increased computational time, there are two other drawbacks of this method. The first drawback is that, as mentioned above, it is highly likely that a pixel (that happens to be in a group due to 35% inaccuracy of the metric) would fall within the class consisting of less number of pixels. This implies that there still a (small) "possibility" that such a pixel would fall within the class consisting of large number of pixels. If this happens, there is a chance that this pixel could be selected as a representative pixel. The reconstruction of image using such a representative pixel would adversely affect the image quality. The second drawback is

that, there is also a possibility that an image content which is large and perceptually significant is reconstructed using the pixels that are not the same as the pixels that are being reconstructed. This also does affect the image quality in a negative manner to certain extent. Once the representative color pixels are extracted, they are indexed, transmitted and the image is reconstructed as described in the Section below.

4.6 Image reconstruction

As mentioned earlier, each image block is assigned an index *bl* that identifies the location of the block in the image. Within an image block, each pixel is assigned an index *pl* that identifies the location of each pixel within a block. Each image block is converted into a vector. Each pixel within this vector is represented by three entries corresponding to the three image channels (L, a*, and b*). The vector of each image block is then divided into many image groups. Each group consists of perceptually similar color pixels. From each group, a representative image pixel is extracted. The indices *bl* and *pl* of the representative image pixel and its three values are then transmitted (or stored) as follows.

For the first group within an image block, the index *bl* of the block is first transmitted. The "number of color pixels" in the first group is sent next. This is followed by transmitting the index *pl* of every pixel in that group. Then the three channel values of the representative pixel of the group is transmitted. In the same manner, the same information is transmitted for the second group and so on until the last group within that image block is reached. For the last sub vector for the same image block, only its representative image pixel value is transmitted. Please note that

for the last sub vector, the number of pixels within the last sub vector and the index of those pixels are not transmitted. This is because once the indices of all the color pixels from the previous sub vectors within a particular block are transmitted, the pixel index within the last sub vector will be known as they are the only ones left. Hence, transmitting only the representative pixel value would take care of those pixels. In this manner, we save space by not transmitting the number of pixels and the indices of those pixels for the last group. As mentioned earlier, this has an advantage when small sized blocks are used instead of large sized blocks. For example, the number of bits transmitted from a block of size 16x16 would be much less compared to the number of bits transmitted from a block of size 64x64.

We illustrate the above process as follows. Assume an image is divided into two blocks and each block consists of nine pixels. Assume the first block consists of two image sub vectors, each consisting of perceptually similar color pixels. This means there will be two representative image pixel values for this block. The first color sub vector consists of 4 pixels of index 2, 3, 4, 8 and the representative "image" pixel value (110, 116, 108) corresponding to L*, a* and b* channels, respectively. The second sub vector belonging to the first block consists of 5 pixels, indexed 1, 5, 6, 7, 9 and a representative image pixel value (26, 22, 27). The second block consists of two image sub vectors. This means there will be two representative image pixel

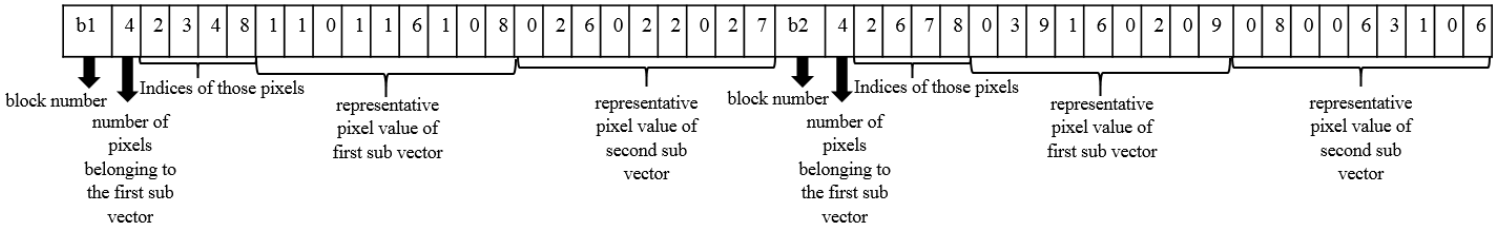

Fig. 4.2. Example indexing of blocks and pixels belonging to that block

values. The first sub vector belonging to the second block consists of 4 pixels of index 2,6,7,8 and a representative image pixel value (39, 160, 209) corresponding to L, a*, and b* channels. The second sub vector belonging to the second block consists of 5 pixels, each indexed 1, 3, 4, 5, 9 and a representative image pixel value (80, 63, 106). The information would be transmitted as shown in Fig. 4.2.

4.7 Experimental results

The proposed algorithm is implemented using MATLAB R2013a on Intel Core i7-4500U, 2.40 GHz, 8.00 GB RAM. This algorithm is tested on the same six RGB images as the one used to test the method using codebook in Chapter 3: Mandrill[6], Bear[7], Brandy Rose[8], Peppers[1], Lena[1] and Pills[9]. Each of these images is of size 256x256x3.

This method is compared with our proposed codebook generation algorithm (in Chapter 3) along with the three popular lossy image compression algorithms: 1) JPEG, 2) JPEG2000, and 3) Fractal compression. The results are shown in Tables 4.1, 4.2, 4.3 and 4.4 for compression ratios 5:1, 8:1, 10:1, and 16:1 respectively. The aim is to compare the methods in terms of PSNR, SSIM and computational time for the same compression rates.

For JPEG and fractal compression methods, the images were divided into non-overlapping rectangular blocks of size 16x16 (area =256) (we refer to domain block size in case of fractal compression). For JPEG2000, we applied three iterations of DWT using Daubechies 9/7 filter. The resulting image consisted of 1 blur block + 9

[6] Courtesy of the Signal and Image Processing Institute at the University of Southern California
[7] Copyright photos courtesy of Robert E. Barber, Barber Nature Photography (REBarber@msn.com)
[8] Copyright photo courtesy of Toni Lankerd
[9] Copyright photo courtesy of Karel de Gendre

detail blocks = 10 blocks. In the proposed VQ algorithm of Chapter 3, we used hexagonal blocks with l_{hex} = 10 pixels (area =260). The sliding window inside each hexagonal block in this method was of size 4x4.

From the tables, one can conclude the following:

1) In terms of algorithm performance, the PSNR, the SSIM and the computational time are the highest (best) for the algorithm in this Chapter and the lowest (worst) for JPEG.

2) In terms of image type, the PSNR and the SSIM are the lowest (worst) for Bear image. This is closely followed by Mandrill image. This is because in both images, it can be observed that the pixel values change frequently from one pixel to the next, compared to other images. On the other hand, the PSNR and the SSIM are the highest (best) for Pills image. This is because Pills image mostly consists of uniform intensities. For the same reason, the computational time is the highest (worst) for Bear image and the lowest (best) for Pills image.

Table 4.5 shows the average percentage improvement in terms of the PSNR, the SSIM and the computational time of the proposed algorithm (without codebook) over the proposed codebook generation algorithm in Chapter 3 and the existing methods: JPEG, JPEG2000, Fractal compression. The overall average percentage is computed by averaging the values of all the methods across the six images and across the four compression ratios. From this table, we conclude that the method proposed in this Chapter is highly exhaustive. It does not only has the highest computational time but the computational time difference between this method and any of the four methods is significantly large. However, for the same compression rate, this method

provided the most improvement compared to existing methods and the proposed codebook generation algorithm, in terms of PSNR and SSIM.

Figures 4.3, 4.4 and 4.5 show the variations in the PSNR, the SSIM and the computational time, respectively, for each of the six image classes, for all five methods and for varying compression ratios: 5:1, 8:1, 10:1, and 16:1. From Figure 4.3, 4.4, and 4.5, we conclude that the PSNR, the SSIM and the computational time is the highest for the proposed algorithm and the lowest for JPEG method. Also, the same measures are the highest for compression ratio 5:1 and the lowest for compression ratio 16:1.

Figure 4.6 shows the six images[5], which are of size 256x256x3.

Figure 4.7 shows the original and the reconstructed Lena image. This image is used to compare JPEG, JPEG2000, JPEG, fractal compression, the proposed codebook generation algorithm and the proposed algorithm without codebook of compression ratio 5:1, along with their PSNR, respectively.

TABLE 4.1

COMPARISON OF THE FIVE METHODS IN TERMS OF PSNR, SSIM AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR COMPRESSION RATIO 5:1

Compression ratio 5:1																		
	Mandrill			Bear			Brandy Rose			Lena			Peppers			Pill		
Parameters	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time
Proposed algorithm	57.44	0.95	3.35	52.16	0.91	3.12	66.54	0.97	2.95	70.19	0.98	2.02	70.19	0.98	2.79	76.58	0.99	1.16
Proposed VQ algorithm	55.62	0.90	1.30	48.77	0.88	1.36	59.13	0.97	1.20	65.33	0.95	1.23	65.33	0.97	0.99	68.00	0.98	0.94
JPEG 2000	45.66	0.90	0.98	41.08	0.85	1.10	51.27	0.95	1.19	55.02	0.92	0.88	55.02	0.95	1.00	59.81	0.96	0.92
Fractal compression	39.81	0.86	0.94	36.63	0.82	1.02	48.97	0.91	0.80	50.04	0.90	0.83	50.04	0.92	0.77	53.67	0.91	0.70
JPEG	35.50	0.87	0.84	31.19	0.81	0.96	41.54	0.87	0.67	43.81	0.87	0.70	43.81	0.87	0.63	47.55	0.89	0.59

TABLE 4.2

COMPARISON OF THE FIVE METHODS IN TERMS OF PSNR, SSIM AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR COMPRESSION RATIO 8:1

Compression ratio 8:1																		
	Mandrill			Bear			Brandy Rose			Lena			Peppers			Pill		
Parameters	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time
Proposed algorithm	49.27	0.90	2.64	49.11	0.95	2.40	57.98	0.96	2.38	51.82	0.93	1.89	67.14	0.96	1.83	70.04	0.97	1.95
Proposed VQ algorithm	40.81	0.88	0.84	38.76	0.84	0.91	47.33	0.94	0.70	43.41	0.92	0.78	60.03	0.94	0.66	64.98	0.96	0.66
JPEG 2000	37.63	0.85	0.73	35.07	0.82	0.85	45.05	0.90	0.63	40.19	0.88	0.54	51.12	0.93	0.62	53.21	0.94	0.62
Fractal compression	32.18	0.82	0.77	28.74	0.80	0.80	40.45	0.89	0.50	36.66	0.85	0.52	46.61	0.89	0.46	48.74	0.91	0.42
JPEG	29.50	0.80	0.70	24.44	0.78	0.75	39.13	0.88	0.39	32.24	0.82	0.43	39.88	0.87	0.33	42.10	0.91	0.31

TABLE 4.3

COMPARISON OF THE FIVE METHODS IN TERMS OF PSNR, SSIM AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR COMPRESSION RATIO 10:1

Compression ratio 10:1																		
	Mandrill			Bear			Brandy Rose			Lena			Peppers			Pill		
Parameters	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time
Proposed algorithm	43.52	0.89	2.43	43.72	0.87	2.12	50.13	0.95	1.76	47.62	0.92	1.62	54.29	0.95	1.89	66.80	0.97	1.76
Proposed VQ algorithm	38.87	0.86	0.75	40.44	0.82	0.84	44.69	0.93	0.45	40.86	0.90	0.60	49.73	0.94	0.41	59.37	0.95	0.62
JPEG 2000	35.21	0.83	0.51	35.55	0.81	0.62	43.12	0.91	0.62	37.54	0.88	0.41	46.03	0.91	0.51	50.24	0.93	0.77
Fractal compression	29.85	0.81	0.46	30.16	0.77	0.60	39.02	0.86	0.40	32.43	0.84	0.42	41.59	0.87	0.40	43.33	0.90	0.35
JPEG	24.14	0.79	0.41	26.43	0.76	0.54	36.22	0.85	0.31	27.65	0.83	0.34	39.00	0.86	0.29	40.19	0.88	0.25

TABLE 4.4
COMPARISON OF THE FIVE METHODS IN TERMS OF PSNR, SSIM AND COMPUTATIONAL TIME FOR SIX RGB IMAGES, FOR COMPRESSION RATIO 16:1

Compression ratio 16:1																		
	Mandrill			Bear			Brandy Rose			Lena			Peppers			Pill		
Parameters	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time	PSNR	SSIM	time
Proposed algorithm	33.79	0.80	1.87	33.43	0.77	2.00	39.87	0.88	1.23	37.15	0.83	1.31	44.72	0.90	1.10	54.23	0.94	0.90
Proposed VQ algorithm	29.81	0.78	0.56	30.87	0.74	0.73	36.81	0.85	0.37	33.41	0.83	0.48	40.85	0.88	0.40	49.84	0.94	0.32
JPEG 2000	20.22	0.75	0.43	22.52	0.70	0.57	30.11	0.82	0.46	26.99	0.79	0.37	33.90	0.85	0.46	44.32	0.91	0.47
Fractal compression	16.85	0.72	0.35	20.70	0.68	0.40	21.44	0.80	0.20	19.48	0.74	0.23	24.12	0.82	0.18	39.81	0.87	0.15
JPEG	16.29	0.69	0.21	18.43	0.67	0.37	18.12	0.77	0.15	16.65	0.71	0.18	20.39	0.81	0.16	34.88	0.86	0.14

TABLE 4.5
THE AVERAGE PERCENTAGE IMPROVEMENT OF THE PROPOSED ALGORITHM OVER PROPOSED CODEBOOK GENERATION ALGORITHM, JPEG2000, FRACTAL COMPRESSION, JPEG , IN TERMS OF PSNR, SSIM AND COMPUTATIONAL TIME FOR SIX RGB IMAGES

Improvements of proposed algorithm over	Proposed algorithm with codebook	JPEG2000	Fractal compression	JPEG
PSNR	11.76%	28.98%	47.78%	66.71%
SSIM	2.22%	5.75%	9.52%	12.20%
Computational Time	-169.33%	-197.06%	-281.13%	-359.09%

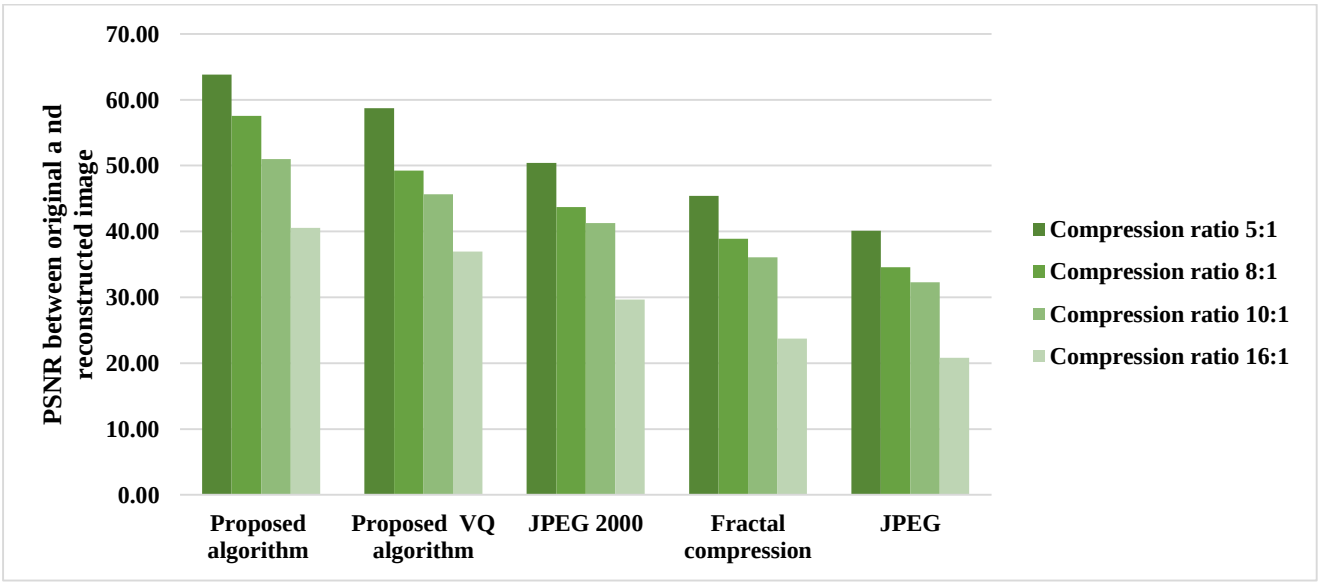

Fig. 4.3. Comparison of the five methods in terms of PSNR tested on six RGB images for varying compression ratios

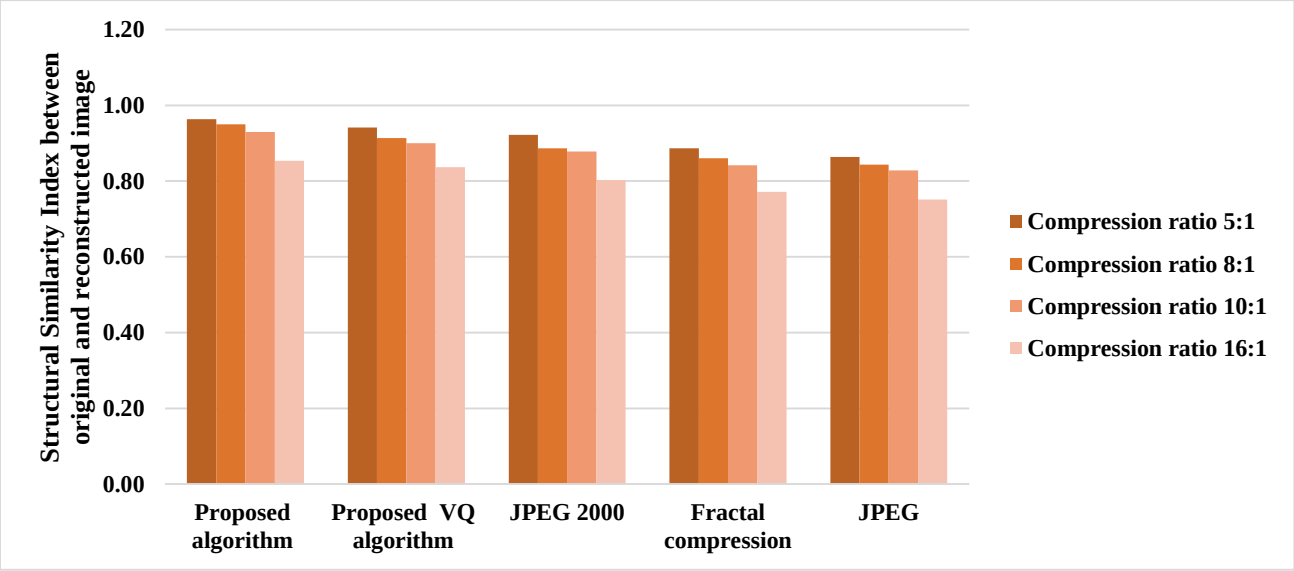

Fig. 4.4. Comparison of the five methods in terms of SSIM tested on six RGB images for varying compression ratios

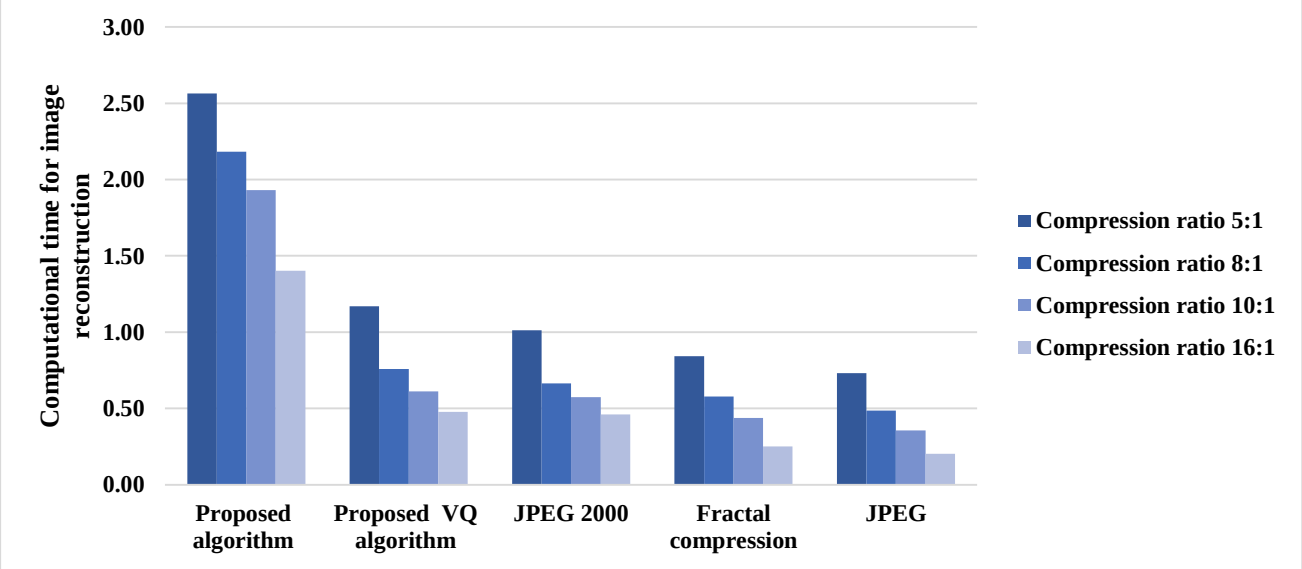

Fig. 4.5. Comparison of the five methods in terms of computational time tested on six RGB images for varying compression ratios

Mandrill

Bear

Brandy rose

Peppers

Lena

Pills

Fig. 4.6. Six original color test images, each of size 256x256x3

Original image (Zoomed)

Fractal compression, PSNR: 50.04

JPEG2000, PSNR: 55.02

JPEG, PSNR: 43.81

Alg. without codebook, PSNR: 70.19

Alg. with codebook, PSNR: 65.33

Fig. 4.7. Original and reconstructed Lena image for the proposed codebook generation algorithm JPEG, JPEG2000, fractal compressions, proposed algorithm using codebook for compression ratio 5:1

Chapter 5

Summary and Conclusion

We propose two new quantization algorithms for color image compression. The first proposed algorithm has two steps: 1) dividing the training images into hexagonal blocks and 2) extracting the representative pixels from a set of less commonly occurring pixels within these blocks to form codewords. Thus, unlike existing methods which takes the average of certain training vector groups as codewords and use many iterations to update them, we form the codewords directly from the image blocks without using iterations. This is done by extracting pixels that represent infrequent (unusual) information in each block in the image. This is the main aspect that differentiates our proposed algorithm from existing VQ methods. This method can be used for compression and reconstruction of grayscale or color images. This method is compared with two prominent and two recent codebook generation algorithms. The second proposed method improves the image quality compared to that resulting from the first proposed algorithm. The following two main changes are incorporated in the second proposed method: 1) the representative pixels are extracted from a set of "perceptually" similar color pixels instead of a set of less commonly occurring pixels and 2) the image pixels are reconstructed using the closest representative pixels instead of constructing an entire small image region simultaneously using codeword. Since the representative pixels are extracted from a set of perceptually similar color pixels, this method only applies to compression and

reconstruction of color images. This method is compared with our proposed codebook generation algorithm along with three prominent lossy image compression algorithms.

From the experimental results, one can conclude that the proposed codebook generation algorithm gives the best results amongst all methods compared to. The improvements are 49.62%, 42.98%, 36.33% and 25.94% over the K-means, LBG, KFCG and KEVRW algorithms respectively, in terms of the MSE for similar compression rates. Also a significant reduction in the computational time is observed. The proposed algorithm reduces the computational time by 56.41%, 49.25%, 39.29% and 24.44% compared to these algorithms, respectively. For the proposed algorithm that does not use a codebook, one can conclude that this method has better performance compared to the former algorithm and JPEG, JPEG2000 and fractal based algorithms. The improvements are 11.76%, 28.98%, 47.78% and 66.71% over proposed codebook generation algorithm, JPEG2000, fractal compression and JPEG, in terms of the PSNR for similar compression rates. However, this algorithm is computationally expensive compared to the rest. The proposed algorithm increases the computational time by 169.33%, 197.06%, 281.13% and 359.09% compared to the aforementioned algorithms, respectively. For all methods, it is observed that the reconstructed image of 'Bear' is the most distorted amongst the test images. This implies that images with high texture or having frequently occurring high intensity changes tend to have the highest error rates.

In order to demonstrate the effectiveness of each of the two aspects of our proposed codebook generation algorithm (i.e. the use of hexagonal blocks and the proposed codeword extraction procedure), we tested the contribution of each of these

aspects in terms of compression error and computational time. The first test used hexagonal blocks but varied the codeword formation process and the second test used the proposed codeword formation process but varied the type of the blocks. The proposed algorithm (using hexagonal blocks and the proposed codeword extraction procedure) yielded better results than 1) the LBG method that used hexagonal blocks and 2) the proposed codeword extraction procedure that used rectangular blocks (where both types of blocks had similar sizes). From the results obtained by the LBG method using hexagonal blocks, one can conclude that dividing the image into hexagonal blocks (instead of rectangular blocks) improves the performance of the LBG algorithm by 7.61% in terms of MSE. However, the computational time is increased by 3.60%. When the proposed method used rectangular blocks instead of hexagonal ones, the improvements over existing methods also increased but to a lesser degree than when hexagonal blocks were used. The improvements here ranged from 40.16% to 12.03% over the K-means and KEVRW algorithms respectively, in terms of MSE; and from 52.56% to 17.78% for the same methods in terms of the computational time. Thus the proposed codeword extraction procedure plays a more important role than that of using hexagonal blocks, in terms of reducing the error rate and the computational time. Also, we show why the representative pixels selected from the set of less commonly occurring pixels reconstructs the image with better contrast. We placed the extracted representative pixels together. We compare this image with the three other images, in terms of contrast per pixel, where representative pixels are computed using three different statistical methods. It was observed that the average contrast per pixel for the image using representative pixels (extracted in our

proposed codebook generation algorithm) is the highest compared to the other three images, thus, implying that the selected representative pixels reconstructs sharper looking images. The image reconstructed by the second proposed method however yield images that are truer to the original ones.

Bibliography

[1] X.Wu and N. Memon, "Context-based, adaptive, lossless image coding," IEEE Trans. Commun., vol. 45, no. 4, pp. 437{444, Apr. 1997.

[2] P. G. Howard and J. S. Vitter, “Fast and efficient lossless image compression," in Proc. IEEE Data Compression Conference (DCC'93), Snowbird, UT, pp. 351-360, Mar/Apr 1993.

[3] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The loco-i lossless image compression algorithm: Principles and standardization into jpegls," Hewlett-Packard Laboratories and Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, HP Laboratories Tech. Report HPL-98-193R1, Oct. 1999. [Online]. Available: http://www.hpl.hp.com/loco/HPL-98-193R1.pdf

[4] A. Said and W. A. Pearlman, "A new, fast, and effcient image codec based on set partitioning in hierarchical trees," IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 3, pp. 243-250, Jun. 1996.

[5] X. Chen, S. Kwong, and J.-F. Feng, "A new compression scheme for color-quantized images," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 10, pp. 755-777, Oct. 2002.

[6] I. Avcibas, N. Memon, B. Sankur, and K. Sayood, "A progressive lossless/near-lossless image compression algorithm," IEEE Signal Process. Lett., vol. 9, no. 10, pp. 312-314, Oct. 2002.

[7] A. J. Pinho and A. J. R. Neves, "A context adaptation model for the compression of images with a reduced number of colors," in Proc. IEEE International Conference on Image Processing (ICIP'05), vol. 2, Genova, Italy, Sep. 2005, pp. 738-741.

[8] T. Acharya and P.-S. Tsai, "JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures," 1st ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., Oct. 2004.

[9] K. Sayood, "Introduction to Data Compression," 3rd ed., ser. The Morgan Kaufmann Series in Multimedia Information and Systems. San Francisco, CA: Morgan Kaufmann, Dec. 2005.

[10] C. E. Shannon, "A mathematical theory of communication," Bell System Technical Journal, vol. 27, pp. 379-423, 1948.

[11] G. R. Grimmett and D. R. Stirzaker, "Probability and Random Processes," 3rd ed. New York: Oxford University Press, Jul. 2001.

[12] D. Soloman, "Variable-length Codes for Data Compression," Springer, Oct. 2007.

[13] D. Huffman, "A method for the construction of minimum redundancy codes," Proc. of the Institute of Radio Engineers, 40:1098-1101, 1952.

[14] M. Nelson, "The Data Compression Book," Prentice Hall, 1991.

[15] D. Salomon, "Handbook of Data Compression," Springer, Dec. 2009.

[16] S.Jayaraman, S.Esakkirajan, T. Veerakumar, “Digital Image Processing” Mc Graw Hill Publishers, 2009

[17] S.P. Lloyd, “Least square quantization in PCM,” IEEE transactions on information theory, vol.28, no. 2, pp.129-137, Mar. 1982.

[18] Y. Linde, A. Buzo, and R. M. Gray, “An algorithm for vector quantizer design,” IEEE Trans. Commun., vol. COM-28, no. 1, pp. 84-95, Jan. 1980.

[19] R. Gersho and R. M. Gray, “Vector quantisation and signal compression,” Kluwer Academic Publishers, Boston, 1992.

[20] J. D. McAuliffe, L. E. Atlas, and C. Rivera, “A comparison of the LBG algorithm and kohonen neural network paradigm for image vector quantisation,” In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 2293-2296, 1990.

[21] P. C Cosman, K. O. Perlmutter, S. M. Perlmutter, R. A. Olshen, and R. M. Gray, “Training sequence size and vector quantiser performance,” In Proceedings of the 25th Asilomar Conference on Signals, systems and Computers, pp. 434-438, 1991.

[30] Y. Liang, "Rate-distortion Optimized Methods for Wavelet-based Image/video Coding," Utah State University, Department of Electrical and Computer Engineering, 2007

[31] P. Auscher, G. Weiss, and M. V. Wickerhauser, “Local sine and cosine bases of Coifman and Meyer and the construction of smooth wavelets,” In C. K. Chui, editor, Wavelets: A Tutorial in Theory and Applications, pp. 15-51. Academic Press, 1992.

[32] H. Malvar, “Signal Processing with Lapped Transforms,” Artech House, 1992.

[33] H. S. Malvar, “Extended lapped transforms: Properties, applications and fast algorithms,” IEEE Transactions on Signal Processing, vol. 40, no. 11, pp. 2703-2714, 1992.

[34] N.D. Zervas, G.P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C.E. Goutis, “Evaluation of Design Alternatives for the 2-D-Discrete Wavelet Transform,” IEEE Transactions in Circuits and Systems for Video Technology, vol. 11, no. 12, pp. 1246-1262, Dec. 2001.

[35] J.M. Shapiro, “Embedded Image Coding using ZeroTrees of Wavelet Coefficients,” IEEE Transactions on Signal Processing, vol. 41, pp. 3445-3462, Nov. 1993.

[36] A. Said and W.A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 243–250, Jun. 1996, doi:10.1109/76.499834. ISSN 1051-8215.

[37] D. S. Taubman, “High performance scalable image compression with EBCOT,” IEEE Trans. Image Proc., vol. 9, pp. 1158-1170, July 2000

[38] Y. Fisher, “Fractal Image Compression: Theory and Application,” New York: Springer-Verlag, 1994.

[39] A. E. Jacquin. “Image coding based on a fractal theory of iterated contractive image transformations,” IEEE Transactions on Image Processing, vol. 1, no. 1, pp. 18-30, 1992.

[40] B. Bani-Eqbal, “Enhancing the speed of fractal image compression,” Optical Engineering, vol. 34, no. 6, pp. 1705-1710, 1995.

[41] D. M. Monro and F. Dudbridge, “Fractal approximation of image blocks,” In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 485-488, 1992.

[42] H. B. Kekre, and Tanuja K. Sarode, “Fast Codebook Generation Algorithm for Color Images using Vector Quantization,” International Journal of Computer Science and Information Technology (IJCSIT), vol. 1, no. 1, pp: 7-12, Jan. 2009.

[43] H. B. Kekre, T. K. Sarode and J. K. Save, “New Clustering Algorithm for Vector Quantization using Walsh Sequence,” International Journal of Computer Applications, vol. 39, no. 1, pp. 975 – 8887, Feb. 2012

[44] Z. Li, and Z. M. Lu., “Fast codevector search scheme for 3D mesh model vector quantization”, Electronic Letter, vol. 44, no. 2, pp. 104-105, Jan. 2008.

[45] C. D. Bei and R. M. Gray, “An improvement of the minimum distortion encoding algorithm for vector quantization,” IEEE Trans. Commun., vol. 33, no. 10, pp. 1132–1133, Oct. 1985.

[46] S.D. Thepade, V. Mhaske and V. Kurhade, “Thepade's Hartley Error Vector Rotation for codebook generation in Vector Quantization,” Advances in Computing, Communications and Informatics (ICACCI), pp. 2007-2012, Aug. 2013

[47] S.D. Thepade and V. Mhaske, “New clustering algorithm for Vector Quantization using Haar sequence,” Information & Communication Technologies (ICT), pp. 1144-1149, Apr. 2013

[48] S.D. Thepade, V. Mhaske and V. Kurhade, “New clustering algorithm for Vector Quantization using Slant transform,” Emerging Trends and Applications in Computer Science (ICETACS), pp. 161 - 166, Sept. 2013

[49] H.B. Kekre and T.K. Sarode, “An Efficient Fast Algorithm to Generate Codebook for Vector Quantization,” Emerging Trends in Engineering and Technology, pp. 62-67, Jul. 2008

[50] K. Karhunen, “Ueber lineare Methoden in der Wahrscheinlichtskeitsrechnung,” Annals Academy Science Fennicae Series A. I. vol. 37, 1947.

[51] M.M. Loève, “Fonctions Aleatories de Seconde Ordre,” In Process Stochastiques et Movement Brownien (P. Levt, ed.), Hermann, Paris, 1948.

[52] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” IEEE Trans. Computer, vol. COM-23, no. 1, pp. 90-93, 1974.

[53] W. Pennebaker and J. Mitchell, “JPEG, Still Image Data Compression Standard,” New York: Van Nostrand, 1993.

[54] I. Daubechies, “Orthonormal bases of compactly supported wavelet,” Commun. Pure Appl. Math., vol. 41, pp. 909-996, Nov. 1988.

[55] I. Daubechies, “Ten Lectures on Wavelets,” SIAM, 1992.

[56] M. J. Shensa, “The discrete wavelet transform: Wedding the a trous and mallat algorithms,” IEEE Trans. on Signal Processing, vol. 40, no. 10, pp. 2464-2482, Oct. 1992.

[57] G. Strang, “Wavelets and dilation equations: A brief introduction,” SIAM Review, vol. 31, pp. 614-627, Dec. 1989.

[58] “ISO/IEC 15444-1”, “JPEG 2000 Part I Final Committee Draft Version 1.0,” March 2000.

[59] C. K. Chan and W. H. Lam, “Efficient use of pseudo-noise sequences in synchronous direct-sequence spread-spectrum multiple-access communication systems,” Vehicular Technology Conf., Stockholm, Sweden, pp. 540-545, Jun. 1994

[60] X. Li and J. Ritcey, “M Sequences for OFDM Peak-to-average Power Ration Reduction and Error Correction,” IEEE Elect. Lett., vol. 33, pp.554 -555, Mar. 1997.

[61] C. Yuan and J.J. Komo, “Error correcting CDMA communication system,” Proc. 22nd Southeastern Symp. on System Theory, pp. 215–218, Mar. 1990

[62] S. Lin and D. J. Costello, “Error Control Coding: Fundamentals and Applications,” Englewood Cliffs, NJ: Prentice-Hall, 1983

[63] A. Pujol and L. Chen, "Color quantization for image processing using self information," 6th International Conference on Information, Communications & Signal Processing, 2007

[64] H. Wang, X. He, T. Hintz and Q. Wu, “Fractal image compression on hexagonal structure”, Journal of Algorithms & Computational Technology, vol. 2, no. 1, pp.79-97, Jan. 2008

[65] G.M. Johnson and M.D. Fairchild, "Darwinism of Color Image Difference Models," IS&T/SID 9th Color Imaging Conference, Scottsdale, pp. 24-30, 2001

[66] Pappas M., Pitas I., “Digital Color Restoration of Old Paintings,” IEEE Transactions on Image Processing, vol. 9, no. 2, pp. 291-294, 2000.

[67] A. Pujol, and C. Liming, "Color Quantization for image processing using self information," 6th IEEE International Conference on Information, Communication and signal processing, 2007.

[68] M. Vik, "Industrial color difference evaluation: LCAM textile data," AIC 2004 Color and Paints, Interim Meeting of the International Color Association, Proceedings, 2004

[69] H. Xu and H. Yaguchi, "Visual evaluation at scale of threshold to suprathreshold color difference," Color Research and Application, vol. 30, no. 3, pp. 198-208, 2005.

[70] R.G. Kuehni, "Color vision and Technology," Research Triangle Park, NC : American Association of Textile Chemists and Colorists, 2008

[71] M. Bartkowiak, “Vector Oriented Methods for Compression of Color Images and Video,” Politechnika Poznańska, 1999

[72] EBU Operating EuroVision and EuroRadio, COMPARISON OF CIE COLOUR METRICS FOR USE IN THE TELEVISION LIGHTING CONSISTENCY INDEX (TLCI-2012), TECH 3354, Geneva, 2012

[73] R.J. Osgood, and M.J. Hinshaw, "Cengage Advantage Books: Visual Storytelling: Videography and Post Production," Second edition, Wadsworth Publishing Co Inc, 2013.

[74] B. Andersson, and J.L. Geyen, "The DSLR Filmmaker's Handbook: Real-World Production Techniques," Second edition, John Wiley & Sons, Inc., Indianapolis, Indiana, 2015.

[75] A.L. Todorovic, "Television Technology Demystified: A Non-technical Guide," Elsevier Inc., 2006

[76] R. Parekh, "Principles of Multimedia," Tata McGraw-Hill Education, 2013

[77] C. Chen, W.K. Härdle and A. Unwin, "Handbook of Data Visualization," Springer, 2008

[78] H. Xu, H. Yaguchi, S. Shioiri, "Correlation between visual and colori-metric scales ranging from threshold to large color difference," ColorRes Appl, vol. 27, pp. 349–359, 2002.

[79] D.H. Alman, R.S. Berns, G.D. Snyder, and W.A. Larsen, "Performance testing of color-difference metrics using a color tolerance dataset," Color ResAppl,vol. 14, pp. 139–151, 1989.

[80] S.S. Guan and M.R. Luo, “Investigation of parametric effects using small colour differences,” Color Res Appl, vol. 24, pp. 331–343, 1999.

[81] B. Preim and C.P. Botha, “Visual Computing for Medicine: Theory, Algorithms and Applications,” Second ed., Elsevier Science and Technology Books, Nov. 2013.

[82] S. Sahin and S.G. Sumnu, “Physical Properties of Foods,” Springer Science+Business Media, 2006

[83] Z. Khatri, G.Y. Sheikh, K.M. Brohi and A.A. Uqaili, "Effective Colour Management for Textile Coloration-An instrumental way towards perfection," International Conference on Industrial Engineering and Operations Management Kuala Lumpur, Malaysia, 2011

[84] P. Yu and A.N. Venetsanopoulos, "Improved hierarchical vector quantization for image compression," Proceedings of the 36th Midwest Symposium on Circuits and Systems, 1993

[85] M.A. Robers and A.K. Katsaggelos, "Reducing blocking artifacts within vector quantization algorithms," IEEE Transactions on Consumer Electronics, vol.43, pp. 1118-1123, Nov. 1997

[86] T. Shima, S. Saito and M. Nakajima, “Design and Evaluation of More Accurate Gradient Operators on Hexagonal Lattices,” IEEE Trans. Pattern Anal. Mach. Intell., vol 32, no. 6, pp. 961-973, Jun. 2010

[87] Ying Liu, Low-Complexity Scalable Multi-Dimensional Image Coding with Random Accessibility, Ph.D. dissertation, Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Institute, August 2008. CIPR Technical

[88] A.L. Jones, "Cosmetics alter biologically-based factors of beauty: evidence from facial contrast," Evolutionary Psychology, vol. 13, no. 1, pp. 210-229, 2015.

Appendix A

Underlying Mathematical Principles

In this chapter, we review the underlying mathematical concepts related to image compression and, in particular, lossy image compression.

The area of image compression belongs to the domain of signal coding. Signal coding is a generalized area of study whose foundation lies in information theory. Information theory is a mathematical framework related to communication laid down by Claude E. Shannon in his paper [10]. More specifically, in this chapter, we first briefly review the information theory concepts: information source and entropy of a source in Section A.1. We then explain them in terms of lossless image compression, briefly. In Section A.2, we discuss the Rate-Distortion theory where lossy image compression will come into light.

A.1. Information Source and Entropy

According to source coding theory, a sequence of data $d_0, d_1, \dots, d_i$ that is encoded, is emitted from an information source. In this context, the each data point within the above sequence can mean each pixel with an image. Also, the term information, related to some event, represents the probability of occurrence of that event. In other words, according to information theory, the occurrence of a data point

d_i gives $log_2 \frac{1}{p_i}$ bits of information. This implies that the information we get from a data is inversely related to the probability of its occurrence. This means that the occurrence of data that has already been predicted gives us no new information. And hence, the occurrence of data that is least predicted gives a lot of information. In the intermediate case where the occurrence is equally likely and unlikely to happen, we receive 1 bit information when the data occurs. More specifically, we do get $log_2 \frac{1}{p_3}$ bits of information when $d_3$occurs. This means that the average number of bits of information we obtain from observing d_3 is $\sum_{i=0}^{3} p_i \, log_2 \frac{1}{p_i}$. This quantity is called entropy of a source, denoted by H. It was proposed by Shannon in 1948.

Definition given a source that outputs data $d_0, d_1, \dots, d_i$ with probabilities $p_0, p_1, \dots, p_i$ respectively, the entropy of a source is defined as

$$H = \sum_{i=0}^{M} p_i \, log_2 \frac{1}{p_i}$$

In this manner, the significance of the information source lies in entropy. The significance of entropy in turn lies in coding the data that are emitted from a source. In simple terms, entropy represents the average number of bits required to encode the data emitted from a source. This implies that the particular source will be represented by atleast H number of bits. These bits will be unique such that they are decodable at the reconstruction end. Since entropy is the measure of uncertainty, the more random the consecutive data values are, the higher will be the entropy. Hence, now the

question is, how the unpredictability, within a sequence, can be modelled so as to reduce the entropy.

Realistically, the sequence of data that occurs is time varying and unpredictable. In case the data is predictable, the transmitter would not even bother sending it to the user. Such an unpredictability is modelled by assuming that the data is received according to some probability distribution. Encoding an image is one such example where each data represents the occurrence of a pixel value. The model in this application would be to assume that each image pixel ranging fom $d_0, d_1, to\ d_{255}$, has an associated probability of occurrence, $p_0, p_1, \dots, p_{255}$. The unpredictability modelling of the pixel value based on such probability distribution gives way to the concept of information source [11] that emits data from a set called alphabet $\mathcal{A}$. Only finite alphabet of length n is considered in this thesis since digital images consists of finite alphabets.

Definition The discrete alphabet $\mathcal{A}$ consists of a sequence of data. These data are random variables $X = \{X\}_{i=1}^{n} = \{X_1, X_2, \dots\}$ where $\{X\}_{i=1}^{n} \in \mathcal{A}$. The emission of successive variable within a sequence is based on their probabilities in relation to the *previous variable*. X is associated with the probability space $(\mathcal{A}^n, \mathfrak{F}, P_X)$, where $\mathfrak{F}$ is σ-field and P_X is probability measure on $(\mathcal{A}^n, \mathfrak{F})$. P_X measures the probability of a particular variable of alphabet $\mathcal{A}^n$ within σ-field based on the previous variable. Since the outcomes are represented by a random variable, the probability is characterized by joint distribution where $x_1^n \in \mathcal{A}^n$:

$$P_X(X_1^n = x_1^n) = \Pr(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = p(x_1^n)$$

Here, $p(x_1^n)$ is the probability of the occurrence of variables. The probability intersection of n variables is defined based on which the source output can be predicted.

Now we will explain the information theory concept in terms of lossless digital image compression. In image processing, due to spatial smoothness of the image, we expect the pixel values to be similar to the pixel values within its immediate neighborhood. Hence, the dependency of a certain variable on the previous variable could be learnt. In this manner, the variables can be predicted in the future. This results in lower entropy since its future occurrence gives no information to the user. Such a technique that uses previous pixel value to reduce the information of the current pixel and lower the entropy is called *predictive coding*. This technique enables lossy as well as lossless compression. An example of lossy predictive coding scheme is Differential Pulse Code Modulation (DPCM). In case of 2D image compression, a lossless predictive coding technique is *raster-scan order* that is used to generate analog television.

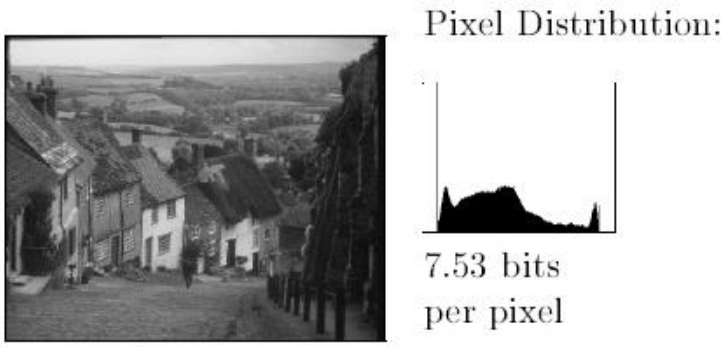

Figure A.1. Without prediction: The entropy of the pixel values is close to the original data size

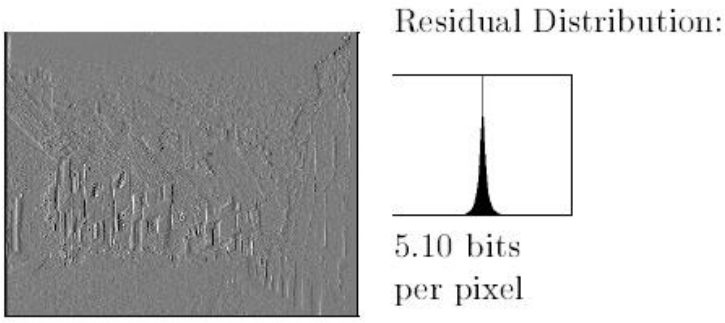

Figure A.2. With prediction: Prediction using the previous pixel value results in reducing the entropy measure

Once the unpredictability of the data distribution is modelled, transmission becomes the point of concern. Over the years, various lossless data coding techniques have been developed for efficient data transmission. The first method that has been widely used is the Variable Length Codes [12]. In this algorithm, the shortest possible code is selected for data that occurs frequently and slightly longer codes for data that do not occur frequently. Another coding technique, Huffman coding [13] is built on the same line as Variable Length Codes. This is entropy coding scheme. This scheme is based on probabilities and predictive models can be built using Huffman coding block as an entity. Hence, Huffman coding scheme cannot be for predictive modelling without using additional algorithms. However, it is designed to assign optimal length integer codes. Around 1970s, another entropy coding scheme was developed in IBM, called Arithmetic coding. This made coding process easier by removing the integer coding constraint. Like Huffman coding, Arithmetic coding can also be adapted to use a predictive model. Continuous research is still being done to further ease the compression algorithm as documented in [14], [15].

A.2. Rate-Distortion Theory [16]

Rate is defined as the number of bits required to represent each variable emitted from the source. And distortion is defined as the measured difference between the input and the output images. This theory gives an analytical expression of how much a data sequence can be compressed at an acceptable level of distortion. In case of lossless compression, there is no distortion between the variables at the transmission and the reconstruction end. However, in certain cases, the predicted variable at the reconstruction end might not be exactly the same as the one at the

transmission side. This results in lossy compression. This type of compression is non-linear, non-invertible and hence lossy. In lossy compression, distortion in the data has to be allowed in order to satisfy the rate constraint. That is, the distortion is introduced in the data to allow us to lower the number of bits required. It is quite reasonable to have the distortion level low enough provided that the output image is still of perceptual importance. The level of distortion is computed in terms of a distortion measure. Within the alphabet $\mathcal{A}$, the distortion measure assigns a non-negative number $d(x, \hat{x}) \geq 0$ between any two letters x and $\hat{x}$, where x is the original variable, $\hat{x}$ is the approximated variable and $d(x, \hat{x})$ is the distortion between x and $\hat{x}$. Hence, the rate-distortion theory states that for a given source and a given distortion measure, there exists a rate-distortion function $R(D)$.

Definition The rate-distortion function $R(D)$ of a source X is the infimum of the distortion of any source-coding scheme which satisfies the rate constraint. Hence, for a discrete source whose distortion measure does not exceed the maximum average distortion threshold, the rate-distortion function is represented as follows:

$$R(D) = \inf_{f_{\hat{x}|x}: d(x,\hat{x}) \leq D} I(x, \hat{x})$$

Here, $d(x, \hat{x}) \leq D$ is the distortion criterion and $I(x, \hat{x})$ is the mutual

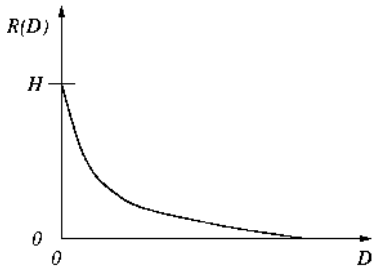

Figure A.3. Rate-distortion function

information between x and $\hat{x}$. Mutual information is the average information that variables x and $\hat{x}$ convey about each other. For example, one variable may convey the reduction in uncertainty about the other variable. The above definition implies that $R(D)$ is the lowest attainable rate needed for the reconstruction of variable x with the given distortion D. The function $R(D)$ basically gives a theoretical bound for the rate required to encode the variable emitted from a source with the maximum distortion D. The rate-distortion function is a convex and monotonically non-increasing function of D represented in Figure A.3.

In Figure A.3, lossless coding is performed when $D = 0$, that is, when the curve touches the y-axis at $R(D)$. On the other hand, lossy compression is a trade-off between rate and distortion. The compressed image would not yield perfect reconstructed due to distortion. $R(D) = 0$ implies the occurrence of maximum distortion.