1995

Lossless image compression in vector transform and vector subband domains

Asaf Sofu
Lehigh University

Sofu, Asaf

Lossless Image Compression in Vector Transform and Vector Subband Domains

October 8, 1995

LOSSLESS IMAGE COMPRESSION IN VECTOR TRANSFORM AND VECTOR SUBBAND DOMAINS

by
Asaf Sofu

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Electrical Engineering

Lehigh University
1995

This thesis is accepted in partial fulfillment of the requirements for the degree of Master of Science.

4/28/95
(Date)

Thesis Advisor

Chairperson of Department

Acknowledgments

First of all, I would like to present my deep thanks to Prof. Weiping Li for his guidance and support during my graduate studies and thesis.

I would like to also express my gratitude to every member of the DSP group individually for their assistance and consideration. I wish to thank also Ahmad Y. Banafa for his help for providing me with the essential materials to complete my thesis. Special thanks to Stephen Corbesero, Maggie and Linda in the EECS Department for all their help.

I want to especially mention the support and assistance of my family from back home during my graduate studies.

Contents

Acknowledgments iv

Abstract 1

1 Introduction 2

1.1 The Information Age 2

1.2 The Necessity of Image Compression 4

1.3 Information Theory and Image Compression 5

1.3.1 Discrete Memoryless Sources (DMS) 5

1.3.2 Markov Sources 7

1.4 Digital Image Compression in General 8

1.4.1 Lossy Compression 9

1.4.2 Lossless Coding 10

2 Huffman Coding 13

3 Adaptive Huffman Coding 17

4 LZW Coding 21

5 Analysis of Lossless Compression Algorithms in Vector Transform and Vector Subband Domains 24

5.1 Description of the Compression Environment 24

5.1.1 Source Characteristics 25

5.2 Huffman Coding 26
5.2.1 Discussion 26
5.3 Adaptive Huffman Coding 30
5.3.1 Discussion 30
5.4 LZW Coding 32
5.4.1 Discussion 33

6 Conclusions **37**

7 Appendix **39**

Bibliography **55**

A Biography **58**

List of Tables

1.1 Data Volumes of Image Sources (in Millions of Bytes). 4
1.2 Storage Capacities (in Millions of Bytes). 4

4.1 LZW decoding example 22
4.2 LZW decoding example 23

5.1 The Results of Huffman Coding for the Lenna Image 27
5.2 The Results of Huffman Coding for the Lenna Image by training with the training image 28
5.3 Huffman Coding of the Video Sequence. Index (0 0) is included in the Huffman table. 29
5.4 Huffman Coding of the video sequence. Index (0 0) is excluded from the Huffman Table. 30
5.5 The Results of Adaptive Huffman Coding For the Lenna Image . . . 31
5.6 Adaptive Huffman Coding of the Video Sequence. Index (0 0) is included in the Huffman table. 32
5.7 Adaptive Huffman Coding of the video sequence. Index (0 0) is excluded from the Huffman Table 33
5.8 The Results of LZW Coding for the Lenna Image 34
5.9 LZW Coding of the Video Sequence. Index (0 0) is included in LZW coding. 35
5.10 LZW Coding of the video sequence. Index (0 0) is excluded from LZW coding. 36

5.11 Summary of coding results of the video sequence with index (0 0) treated as a regular image index. 36
5.12 Summary of coding results of the video sequence with index (0 0) treated seperately from other image indices. 36

List of Figures

1.1 A fundamental communication system 5
1.2 Three stage compression technique 8
1.3 General Image Compression Framework 9

2.1 Huffman Algorithm Source Reduction Process 13
2.2 Huffman Algorithm Codeword Construction Process 14

3.1 An example of a Huffman Tree 18
3.2 Incrementing the Huffman Tree 19
3.3 Swapping the nodes in the Huffman Tree 20
3.4 After several increments in the Huffman Tree 20

5.1 Block Diagram of the Vector Subband Coding Algorithm. 24

7.1 The frequency of occurence in descending order for the image indices in band 0 of the football video sequence. 39
7.2 The frequency of occurence in descending order for the image indices in band 1 of the football video sequence. 40
7.3 The frequency of occurence in descending order for the image indices in band 2 of the football video sequence. 40
7.4 The frequency of occurence in descending order for the image indices in band 3 of the football video sequence. 41
7.5 The frequency of occurence in descending order for the image indices in band 4 of the football video sequence. 41

7.6 The frequency of occurence in descending order for the image indices in band 5 of the football video sequence. 42
7.7 The frequency of occurence in descending order for the image indices in band 6 of the football video sequence. 42
7.8 The frequency of occurence in descending order for the image indices in band 7 of the football video sequence. 43
7.9 The frequency of occurence in descending order for the image indices in band 8 of the football video sequence. 43
7.10 The frequency of occurence in descending order for the image indices in band 9 of the football video sequence. 44
7.11 The frequency of occurence of the indices in descending order for band 0 of the Lenna image. 45
7.12 The frequency of occurence of the indices in descending order for band 1 of the Lenna image. 45
7.13 The frequency of occurence of the indices in descending order for band 2 of the Lenna image. 46
7.14 The frequency of occurence of the indices in descending order for band 3 of the Lenna image. 46
7.15 The frequency of occurence of the indices in descending order for band 4 of the Lenna image. 47
7.16 The frequency of occurence of the indices in descending order for band 5 of the Lenna image. 47
7.17 The frequency of occurence of the indices in descending order for band 6 of the Lenna image. 48
7.18 The frequency of occurence of the indices in descending order for band 7 of the Lenna image. 48
7.19 The frequency of occurence of the indices in descending order for band 8 of the Lenna image. 49
7.20 The frequency of occurence of the indices in descending order for band 9 of the Lenna image. 49

7.21 The frequency of occurence of the indices in descending order for band 0 of the training image. 50
7.22 The frequency of occurence of the indices in descending order for band 1 of the Lenna image. 50
7.23 The frequency of occurence of the indices in descending order for band 2 of the Lenna image. 51
7.24 The frequency of occurence of the indices in descending order for band 3 of the Lenna image. 51
7.25 The frequency of occurence of the indices in descending order for band 4 of the Lenna image. 52
7.26 The frequency of occurence of the indices in descending order for band 5 of the Lenna image. 52
7.27 The frequency of occurence of the indices in descending order for band 6 of the Lenna image. 53
7.28 The frequency of occurence of the indices in descending order for band 7 of the Lenna image. 53
7.29 The frequency of occurence of the indices in descending order for band 8 of the Lenna image. 54
7.30 The frequency of occurence of the indices in descending order for band 9 of the Lenna image. 54

Abstract

Lossless compression algorithms are utilized in image and video compression systems such as JPEG and MPEG. Three lossless compression algorithms, Huffman, Adaptive Huffman and Lempel-Ziv-Welch coding are discussed and analyzed to understand which one is the most efficient and convenient lossless compression scheme for vector transform and vector subband coding algorithms. After a general overview of each of the algorithms, the compression performance of the lossless algorithms and the overall lossy system are compared and analyzed. To achieve the best compression possible, the coding algorithms and the source are modified. The maximum available lossless compression with these techniques are studied and analyzed.

Chapter 1

Introduction

In order to comprehend this discussion fully and to shed light on some background information, some introductory material will be presented.

1.1 The Information Age

" A picture is worth a thousand words "

We live in a world of information. Everyday we are bombarded with information through various electronic media such as television, radio, e-mail etc. Electronic revolution and the advent of the computer technology have made it possible to receive and process information at incredible rates and speeds. Even if written text through books, papers and the like are the main source of information, as the old adage above states, images, pictures and the video convey information and messages much more effectively than written material. Hence, in today's world we can not ignore the power that image and video processing brings in our daily lives.

Image and video processing has many applications in many phases of life, such as in medical diagnostics, legal processes, entertainment, home and business applications, remote sensing of the earth by LANDSAT images, teleconferencing, facsimile operations. With the upcoming "information superhighway", image and video applications will be even more essential in our daily activities. New frontiers will

be opening in videotelephony, electronic retailing, distance learning and many other fields. Companies are fiercely competing now to develop new applications to achieve the the leading edge to enter this highly demanding market.

With so many present applications and many others in sight, image and video compression techniques help us utilize image and video more effectively and conveniently. The research in image and video compression goes back to 1940s when Shannon and his friends formulated the requirements for error-free transmission of digital data [1, 2]. The original research in picture coding was in terms of analog techniques to reduce the analog bandwidth compression [3].

Today, instead of bandwidth compression, digital image and video compression is implemented. The advancement in information theory and electronic and computer technology made it possible to develop sophisticated data compression techniques and apply these algorithms to image and video sequences. The point that image and video compression technology has achieved today is mainly due to three reasons: the development of new compression schemes that decreased the communication transmission requirements, the advances in VLSI technology which made it possible to implement complex compression algorithms and the standardization efforts in order to provide general rules for communication of different media. [4].

1.2 The Necessity of Image Compression

National Archives	$12.5 * 10^9$
1 h of color television	$28 * 10^3$
Encyclopedia Britannica	$12.5 * 10^3$
Book (200 pages of text Characters)	1.3
One page viewed as an image	0.13

Table 1.1: Data Volumes of Image Sources (in Millions of Bytes).

Human brain	125 000 000
Magnetic cartridge	250 000
Optical disc memory	12 500
Magnetic disc	760
2400-ft magnetic tape	200
Floppy disc	1.25
Solid-state memory modules	0.25

Table 1.2: Storage Capacities (in Millions of Bytes).

As seen in the above tables, the storage requirements for image and video sequences are very large. Even though today, there is enough technology to satisfy these requirements, the speed of the system slows down as the amount of data being manipulated increases [5].

If we consider a 720 * 480 pixel/frame television image at 30 frames/second and 8 bits/color and 3 colors needs approximately 250 Mbits/second for transmission. Using a 14.4K modem, it takes more than 4 hours for transmission over the phone lines. The transmission that requires only one second takes more than 4 hours in today's technology. Even if better results have been obtained in transmission of video over the phone lines, the above requirement has not been satisfied yet. Hence, this example shows the importance of image compression. As faster telecommunication transmitters and receivers are invented, more sophisticated compression algorithms are discovered, and the bandwidth of communication channels are increased through

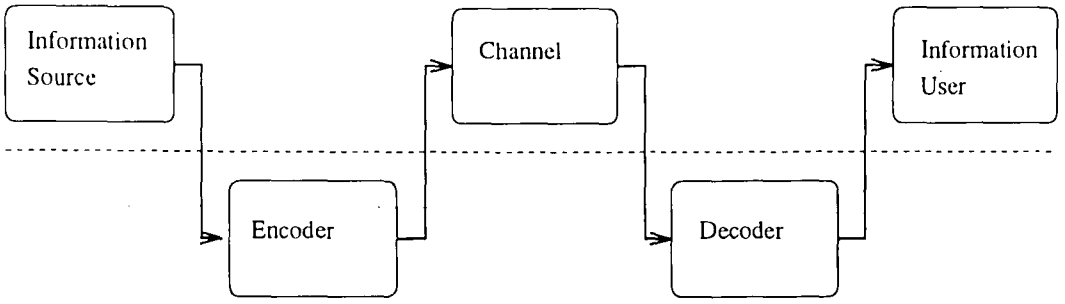

Figure 1.1: A fundamental communication system

the use of fiber optic cables, the *efficient* and *convenient* transmission of video over the telecommunication lines will be possible in the very near future. However, by ever-increasing demand for information and communication, the need for image and video compression will always be there.

1.3 Information Theory and Image Compression

The theoretical fundamentals of image compression are based on information theory. The compression techniques are also implemented in a communication system.

Figure 1.1 shows the functional block diagram of a communication system. A source is an information generator that gives out symbols from a given alphabet. For a system involving image sequences, the output can be 0 and 1s of a binary process or index values of images from 1-D or 2-D scans. The source systems can be further explained as follows.

1.3.1 Discrete Memoryless Sources (DMS)

Information theory provides us with the concept of source entropy and channel capacity for a communication system. The source entropy is the minimum amount of information needed to accurately represent the source and the channel capacity is the maximum amount of information that the system could support in transmitting the source. The simplest of the information sources is a discrete memoryless source (DMS) which can be defined by its source alphabet S_1,S_2,S_3,...,S_n and the related

probability of occurence, $P(s_1)$,$P(s_2)$,$P(s_3)$,...$P(s_n)$. In order to comprehend the definition of the source entropy, one has to understand how "information" can be explained in a discrete memoryless source. [6]

It is intuitive that the occurrence of more probable symbols in a source provide more information than the occurrence of less probable symbols. The total information due to the occurrence of independent events should be equal to the sum of the information of individual events. Hence, according to the above statements, information can be defined as:

$$I(s_i) = log(\frac{1}{P(s_i)})$$

where $P(s_i)$ is the probability of the symbol.

The entropy of the system is defined as the average amount of information provided per symbol i.e.

$$H(s) = \sum_{i=1}^{n} P(s_i) * I(s) = -\sum_{i=1}^{n} P(s_i) log_2 P(s_i)$$

In a binary DMS, the entropy of the system tells us the minimum amount of bits that the system should provide without a loss. In other words, if no distortion is desired in the system, the minimum amount of bit assignment per symbol can not be lower than the entropy of the source. Hence Shannon's source coding theorem draws a lower bound for the compression of a discrete memoryles source[2]. This is a milestone achievement and all the lossless compression algorithms try to approach the entropy of the system. [6]

Often, it is more convenient to deal with blocks of symbols, rather than single symbols. For example for a discrete memoryless system, DMS, with a source S and alphabet of size n, if the symbols are grouped into N blocks with each symbol represented by $\sigma_i = (s_{i_1}, s_{i_2}, s_{i_3}, ..., s_{i_N})$ from the source S^N, the probobality of each block of symbols, σ_i, is represented by $p(\sigma_i) = p(s_{i_1})p(s_{i_2})p(s_{i_2})...p(s_{i_N})$. The entropy of the system can be shown as the product of the entropy of the single symbol, i.e.,[6]

$$H(s^N) = NH(s)$$

1.3.2 Markov Sources

A discrete stochastic process X_1,X_2, ... is said to be a Markov chain or a Markov process if for n=1,2,3,...,

$$Pr(X_{n=1} = x_{n+1}|X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = Pr(X_{n+1} = x_{n+1}|X_n = x_n)$$

for all x_1, x_2,...,x_n,x_{n+1} ϵ X [13]

In images, the probability of a pixel value does not exist in isolation but depends on the probability of pixels surrounding itself. Hence the probabilities of image indices are not as independent as discrete memoryless system assume. In Markov processes or Markov sources the probabilities of individual symbols depend on the preceeding probabilities of symbols:

$$p(s_i|s_{j1}, s_{j2}, ..., s_{jm}) \text{ where i,} j_p(p = 1, ..., m) = 1, 2, ..., n.$$

Hence, the probability of symbol i depends on the probabilities of preceeding m symbols. Zeroth order Markov source is a Discrete Memoryless System. First order Markov source has n states for every symbol s_i, second order Markov source has n^2 states for every symbol s_i. Generally m^{th} order Markov source has n^m states.

The source entropy given that it is in a particular state can be stated as follows:

$$H(S|s_{j1}, s_{j2}, ..., s_{jm}) = -\sum_{i=1}^{n} P(s_i|s_{j1}, , ..., s_{jm}) logp(s_i|s_{j1}, ..., s_{jm}).$$

The entropy of the m^{th} order Markov source can be found by summing through all the possible states, the source entropy multiplied by the probobality of its state.

$$H(S) = \sum_{S^m} p(s_{j1}, s_{j2}, ..., s_{jm}) H(S|s_{j1}, s_{j2}, ..., s_{jm})$$

In our examples, we utilized zeroth order Markov processes, i.e. Discrete Memoryless Sources, because when we increase the order of the system, the number of statistics that one has to take increase drastically. Hence, in turn this increases the memory and speed requirements of the algorithm.

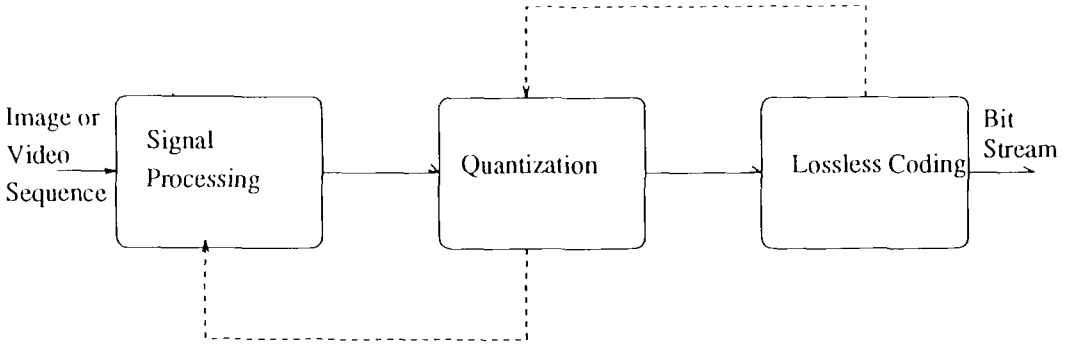

Figure 1.2: Three stage compression technique

1.4 Digital Image Compression in General

Digital image compression techniques can be basically divided into two categories: Lossy and Lossless compression methods.

Figure 1.2 shows a general block diagram of a video and image compression algorithm. Signal processing stage is utilized to transform the image indices into a form more suitable for quantization. This stage does not introduce any noise or does not compress the bits. Loss occurs due to quantization, and the indices after quantization are further compressed using lossless coding[4].

There are basically three types of data redundancies that exist in image and video [6]. These are namely,

- spatial redundancy due to the correlation among pixels in an image.
- spectral redundancy due to the correlation between color and spectral bands.
- temporal redundancy due to the correlation between image frames in a video sequence.

The lossless compression algorithms try to reduce the spatial redundancy without causing any distortion in the image or video.

Fig. 1.2 is also a general block diagram of a lossy compression scheme. As seen from the diagram, lossy compression encompasses a lossless compression algorithm which makes up the last stage of an overall sytem.

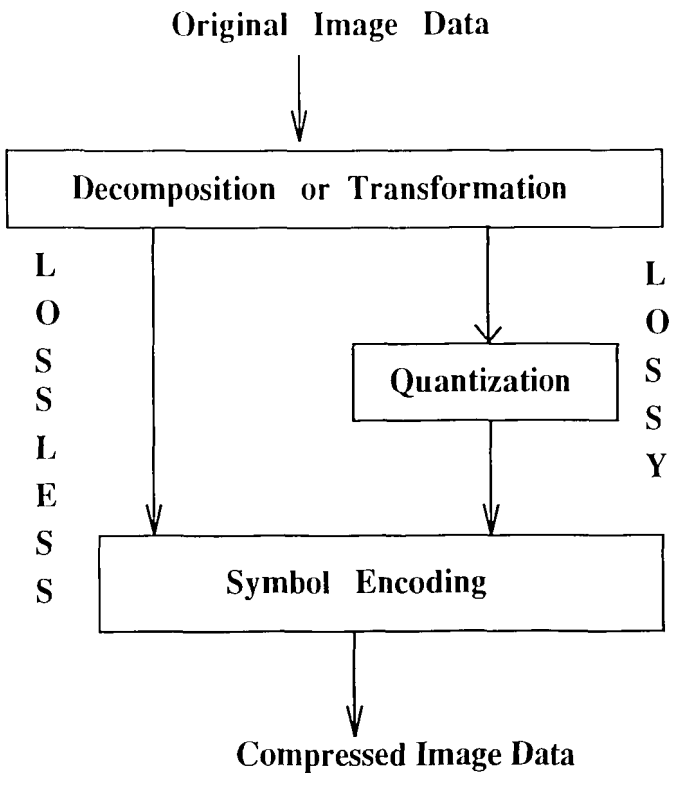

Figure 1.3: General Image Compression Framework

1.4.1 Lossy Compression

Since lossless image and video compression will be comprehended better with an understanding of lossy compression, we would like to address this topic in this section.

Lossy compression is the compression of digital pictures and video with a loss. It is also called irreversible compression since after compression the original data can no longer be retrieved.

The lossy compression scheme can be divided into sections shown in figure 1.3 [7]. In lossy compression, a considerable amount of compression is achieved sacrificing the quality of the image. The visibility of the image might be distorted depending on the compression ratio. The image might be lossy compressed and "visually lossless." However, as higher compression ratios are achieved, the more distorted the image will be.

The first stage of the lossy compression scheme is the deformation or transformation stage. Various transform algorithms might be utilized in this stage, such as DCT, KLT, Walsch transform etc. Several lossy compression algorithms are predictive coding, transform coding, subband coding. The quantization stage introduces a loss into the system. Quantization scheme can be scalar or vector quantization depending on the compression algorithm[8, 9]. In this study, we are using lossy Vector Transform and Vector Subband coding schemes. Vector Transform is a better compression scheme than vector quantization and DCT [10]. Symbol encoding is the lossless compression scheme and comes in the last stage of the lossy compression. Lossless coding improves the compression ratio of the lossy algorithm. Commercial applications such as JPEG and MPEG can be categorized under transform coding lossy compression, since they utilize DCT and scalar quantization in the first two stages of compression. They use run-length encoding (RLE) and a modified Huffman coding for lossless compression [11, 12].

1.4.2 Lossless Coding

Lossless coding is the compression of images or video without any loss of information. Hence, the original picture can be retrieved exactly after decompression. This kind of compression methods might be necessary in medical images where diagnostic accuracy is vital, in business applications for legal reasons and LANDSAT images for the clarity of images.

Compression ratios for lossless coding of still images range from expansion to exceptionally good 4:1 compression. General medical image compression is in the ratio of 1.7:1 to 2:1 [23].

In lossless compression, there are two general methods of compression, i.e minimum-redundancy coding and dictionary based compression algorithms [14, 16]. In our case, we will investigate three lossless compression schemes, namely Huffman coding, Adaptive Huffman Coding and Lempel-Ziv-Welch coding. The first two algorithms can be classified as variable length or entropy coding, and LZW algorithm is classified as dictionary based compression algorithm. We will investigate the two

classes of compression algorithms here.

Variable Length Coding Algorithms

This type of compression method tries to reduce only the coding redundancy. It tries to assign the most probable symbols the shortest code. The information theory tells us that average number of bits needed to accurately represent a source symbol should be greater than the entropy of the source, H(s), i.e. $\sum_{i=1}^{n} P(s_i) log P(s_i)$. Hence, at least entropy amount of bits should be utilized to code the source symbols without any loss and thus the coding techniques try to approximate or achieve close to the entropy of the system. Huffman and Adaptive Huffman algorithms utilize statistical models to perform the compression. They assign a table of symbols, variable length bit streams based on their probability of occurrence. Non-uniform and concentrated probabilities of symbols provide better compression.

Dictionary Based Coding

In dictionary based compression, the algorithms assign variable length symbols indices based on their previous occurence. The scheme that the algorithm follows is easier to understand. This can be further explained by the following example:

If one uses the Random House Dictionary of English, second edition, unabridged and assuming the following references.

1/1 822/3 674/4 1343/60 928/175 550/32 173/46 421/2

The first number refers to the page number and the second number refers to the number of the word on this page. By this method one needs only 12 bits to code 2200 pages and assuming 256 words per page, 8 bits are enough to select any word on a page. Hence a total of 20 bits are needed to code any word on any page in the dictionary.

In the Random House Dictionary of English, totally 43 bytes are needed to represent the corresponding words for these 8 references. However by the above

dictionary based algorithm 2.5 $\frac{byte}{reference}$ * 8 references = 20 bytes are enough for coding. Hence, more than 50 percent compression is achieved through this method.

The important task is to build the dictionary table in the algorithm and either adaptive or statistical methods can be utilized for this purpose. Both of the statistical and adaptive dictionary based algorithms have their advantages and drawbacks, however, adaptive dictionary based algorithms are more well-known and widespread. Adaptive algorithms start with an empty dictionary or a predefined base-line dictionary and add the new phrases as the compression progresses. Statistical compression schemes build a dictionary prior to encoding and perform compression based on this dictionary without modification. Statistical methods have to send the dictionary to the decoding side before the compression starts. This seems to be a major drawback that increases the size of the compression. Adaptive compression algorithms can build the dictionary in real time without any need of transfering the dictionary, however initally in compression they can not build an accurate dictionary to represent the symbols due to the lack of occurence of the symbols in the input sequence. A considerable amount of information has to be read till a precise dictionary of occurence of symbols can be built [15]. The compression algorithms that utilize the adaptive and statistical techniques to build the dictionaries will be analyzed more thoroughly in the following chapters.

Chapter 2

Huffman Coding

Huffman coding is a minimum redundancy compression algorithm that assigns bit-streams to individual symbols [17]. It is proved to produce the optimum code, when symbols are encoded individually. Hence, it is the best minimum-redundancy coding algorithm, if symbols are coded one at a time [1].

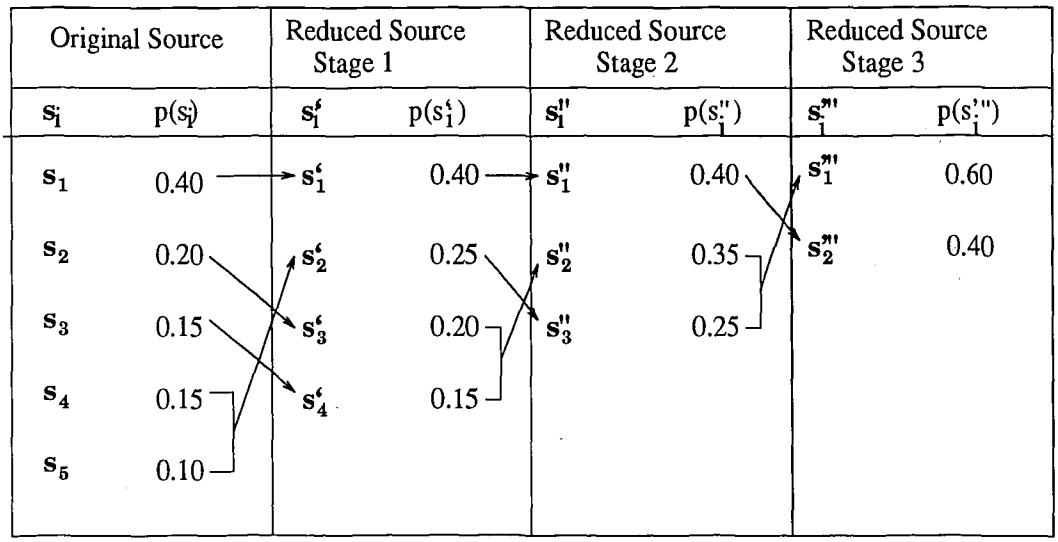

Figure 2.1: Huffman Algorithm Source Reduction Process

Similar to all compression algorithms, Huffman coding is divided into decoding and encoding phases. Encoding consists of statistical modelling, source reduction and codeword construction. As seen in figure 2.1 and figure 2.2, the steps involved

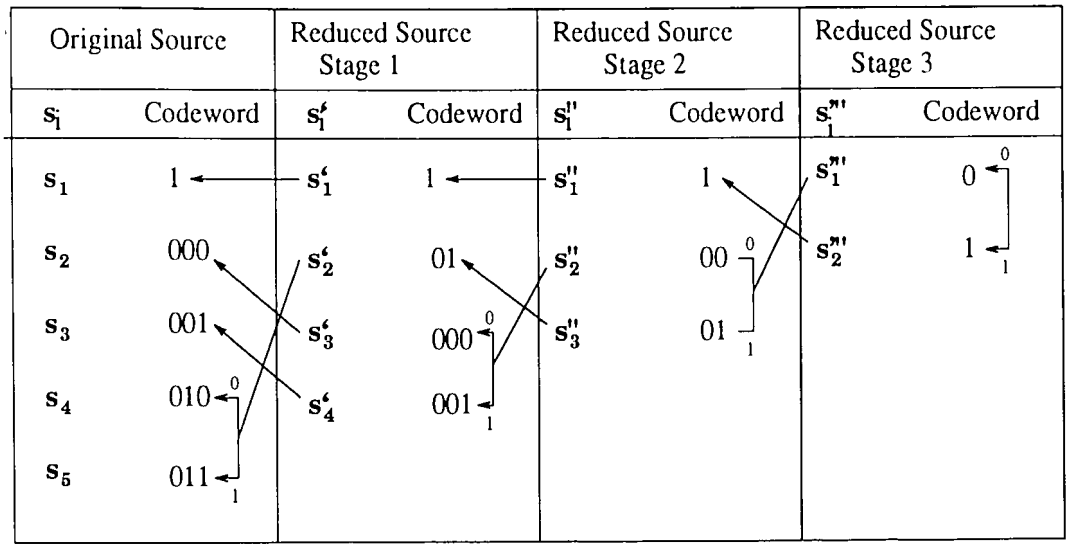

Figure 2.2: Huffman Algorithm Codeword Construction Process

in the encoding phase of Huffman Algorithm can be stated as follows:

Statistical modelling

1. Determine the probability of occurrence of every symbol in the symbol ensemble.

2. Arrange the symbols in order of decreasing probability.

Source Reduction Process

3. Combine the probabilities of the two least probable symbols.

4. Add the two smallest probabilites and assign the sum into a new symbol in the next stage. Rearrange the symbols in the next stage according to their probabilites in descending order.

5. Dedicate the sum as the parent symbol in the next stage and two symbols as children in the current stage. Discard the least probable symbol.

6. Repeat from step 3 for the next stage until the number of members in the symbol ensemble is reduced to two.

Codeword Construction Process

7. Assign one of each of the binary digits, i.e. 0 or 1 to the two symbols obtained in the last stage of the source reduction process.

8. Find the parent in the current stage and assign 0 and 1 to its children in the previous stage.

9. Assign the rest of the symbols in the previous stage the codes of the symbols in the current stage in descending order.

10. Repeat steps 9 and 10 until all of the original source symbols are coded.

Hence as seen from the above process, the two restrictions proposed on developing the Huffman algorithm is as follows:

- Every symbol will consist of an individual code.
- In the decoding end, once the start of the sequences is known , it will be possible to detect each code without the information about its beginning or end.

These properties of Huffman Coding allow the decoding of each symbol from a stream of digits. For example, 01, 102, 111, 202 are valid codes that satisfy the two properties, a sequence of these codes 1111022020101111 can be decoded as 111-102-202-01-01-111-102. However, if the codes are 11, 111, 102, 02, decoding can not be performed without a beginning and end information of each corresponding code, since in the above example, the sequence might have started with 11 or 111 and it is not possible to determine which code it is without a start and end information of the code [17].

Even if Huffman Coding is the optimum minimum-redundancy coding algorithm assigning variable length codes per symbol, it has also some limitations. In a discrete memoryless source or 0-order Markov model as observed in the above example, the ideal codeword length for Huffman coding is $log_2 p(s_i)$ where $p(s_i)$ is the probability

of the symbol s_i. Hence the ideal probabilities of symbols have to be negative powers of 2, i.e. 1/2, 1/4, 1/8,... If the probability of the symbols deviate highly from this condition, the assigned codewords will no longer be efficient. Also, if the statistics utilized to obtain the Huffman table deviate significantly from the statistics of the input data, the codewords no longer approach the entropy of the input data. For example, when digital halftoned images are compressed based on the CCITT international digital facsimile data compression standard, 50% data expansion occurs. Because the Huffman tables in the CCITT standard are obtained by the statistics from binary texts and documents, not from the statistics of digital halftones. Hence, to overcome this problem, the algorithm might obtain the statistics in real time, and utilize it for encoding of the symbols[17]. Thus an adaptive method can be utilized to improve the coding efficiency in certain cases. In Adaptive Huffman Coding which is analyzed in the next section, the Huffman table will be built in real time.

Chapter 3

Adaptive Huffman Coding

The statistical model for variable-length coding algorithms can be built in two ways, by statistical and adaptive methods. Huffman coding discussed in the previous section utilizes static techniques in building the model. It constructs a huffman table from a training image and utilizes this data for coding the rest of the images. To achieve optimum compression, it would be desirable to make the Huffman table for every image. However, this process would be costly and slow. Also if one desires to use higher order modelling, instead of the order-0 modelling used in the previous example, the amount of Huffman tables to be built increases drastically. Hence, to overcome the problem of static modelling, Adaptive Huffman Coding builds the Huffman table adaptively, i.e. in real time. Theoretically, this is the only difference between Huffman and Adaptive Huffman Coding algorithms. In Adaptive Huffman Coding, the Huffman table is to be updated after every symbol is read in. Practically, the Huffman table consists of a binary tree which is reconstructed upon every input symbol. Hence, the Huffman tree can be basically described as a binary tree with every node adjacent to its sibling and sharing the same parent. The nodes in the tree are also organized in terms of increasing weight.

The tree shown in figure 3.1 satisfies the properties of a Huffman tree stated above. The nodes are arranged in terms of increasing weight and every node is adjacent to its sibling. As seen in figure 3.1 the nodes are named A B C D and E.

Practical application, requires the use of a property of Huffman coding called

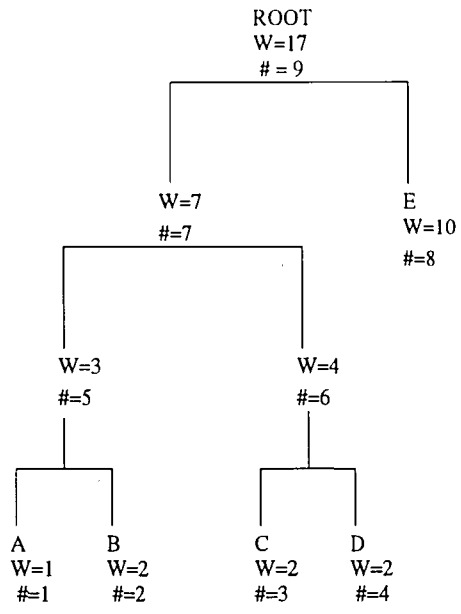

Figure 3.1: An example of a Huffman Tree

the "sibling property." It would be very slow to reconstruct the Huffman tree for each entry symbol. Thw Adaptive Huffman Algorithm makes use of the "sibling property" to update the Huffman tree faster. A Huffman tree shows the sibling property if the weight of every node can be arranged in increasing order and each node can be adjacent to its sibling [18, 15]. This Sibling property allows speedy update of the Huffman tree and thus faster compression of image data [15].

When a node is incremented, the tree is also updated. The update consists of two sections. In the first section, the weight of the corresponding node is incremented, then the weight of its parent is incremented. The incrementation operation continues till the root of the tree is reached. This process is shown in figure 3.2[15].

In the second stage, any nodes that violate the sibling property are arranged accordingly in the Huffman tree. For example as shown in figure 3.3, if node A is incremented twice, its weight becomes larger than other nodes on its row. To maintain the sibling property, it is swapped with node D, since node D is the node with the next lower weight. Only two nodes are swapped for each update of the Huffman Tree. After this update, the upper nodes are also incremented till the root node is reached. Sibling property violations are checked at each stage and the tree

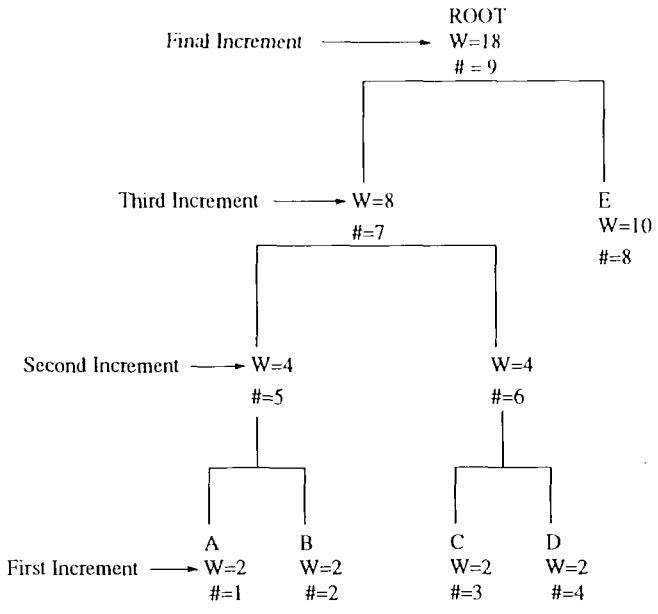

Figure 3.2: Incrementing the Huffman Tree

is arranged again so that it satisfies the sibling property. Swapping of the nodes move the symbols in the Huffman Tree and allow efficient codes to be assigned to them [15].

Figure 3.4 shows the tree after node A was incremented twice again. In this way B and D with the least probability are assigned more bits than C, and A is encoded with fewer bits than B, D and C. In this organization, code E which has the most weight and thus the highest probability is assigned the least number of bits. Hence, this process allows efficient assignment of bits after each update[15].

In the decoding end of the algorithm, the Huffman tree is updated after each symbol is read. The same update process as described above for the encoding end is utilized for the decoding.

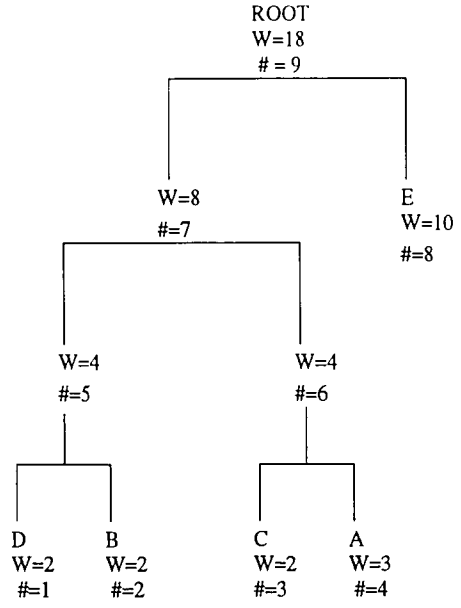

Figure 3.3: Swapping the nodes in the Huffman Tree

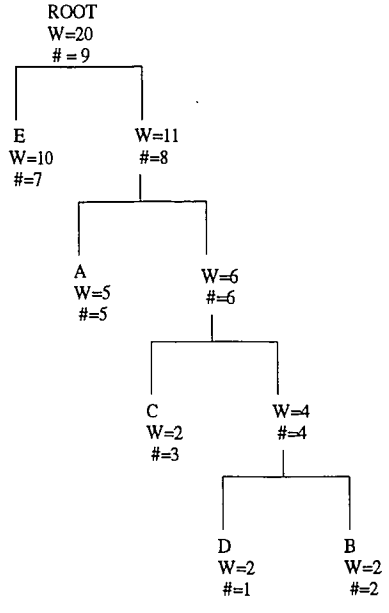

Figure 3.4: After several increments in the Huffman Tree

Chapter 4

LZW Coding

Lempel-Ziv-Welch coding is a dictionary-based coding algorithm. The originators of dictionary based coding is A. Lempel and J. Ziv with their papers in 1977 and 1978[19, 20] with LZ77 and LZ78 algorithms respectively. Terry Welch improved the LZ78 algorithm and called the new one LZW. He also proposed a practical application in data storage mediums[21]. His algorithm sparked attention and led to the development of new compression programs like compress program for UNIX platforms.

LZW method reads input characters or symbols one at a time and builds up phrases. It sends out the corresponding code when a match occurs between the input data and the phrases in the dictionary. The flow of the algorithm can best be explained by an example.

In the example shown in table 4.1, the LZW dictionary is assumed to contain all the ASCII codes. In the first pass, it checks wether " W" exists in the dictionary, since it is not in the dictionary, the algorithm assigns a code to the new phrase beyond the already utilized codes. The first code that is available for use is 256, which comes after ASCII codes. The ASCII code for the space character is output and the process continues with "WE". Since this phrase is also not in the dictionary, it's included in the dictionary. The process continues till a match between the input characters and the dictionary phrase occurs. For instance, when " WE" is reached, the code corresponding to previously matched phrase " W" is output along with the

Input string: " WED WE WEE WEB WET"

Characters Input	Code Output	New code value and associated string
" W"	" "	256 = " W"
"E"	'W'	257 = "WE"
"D"	'E'	258 = "ED"
" "	"D"	259 = "D "
"WE"	256	260 = " WE"
" "	'E'	261 = "E "
"WEE"	260	262 = " WEE"
" W"	261	263 = "E W"
"EB"	257	264 = "WEB"
" "	B	265 = "B"
"WET"	260	266 = " WET"
EOF	T	

Table 4.1: LZW decoding example

ASCII code for "E". " WE" is coded and added to the dictionary. The compression continues till all the input data is processed [15].

The decoding of the LZW algorithm is similar to the encoding. The characters are read in pairs. As shown in table 4.2, **OLD_CODE** is the first character and the **NEW_CODE** is the second character. As the characters are read in, the dictionary builds up similarly in the decoding end. The dictionary of the decoding end is a replica of the dictionary in the encoding side. Hence, LZW algorithm uses an adaptive technique to build and utilize the dictionary [15].

No matter how big the dictionary size is, it will fill up eventually. Hence, the dictionary should be manipulated so that it could contain the codes essential for good compression. If the same dictionary is maintained after it is full, it might not provide efficient tables for codes encountered later in the image. The UNIX compress program handles dictionary management by monitoring the compression ratio and deleting the dictionary and starting a new one once the compression ratio starts to deteriorate. A more simplistic approach is to flush the dictionary and start a new one when it is full[15].

Another issue in implementing a dictionary is that for smaller and almost random

Input Codes: "WED<256>E<260><261><257>B<260>T"

Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
' '	' '	" "		
'W'	' '	"W"	'W'	257=" W"
'E'	'W'	"E"	'E'	257="WE"
'D'	'E'	"D"	'D'	257="ED"
256	'D'	" W"	' '	259="D "
'E'	256	"E"	'E'	260=" WE"
260	'E'	" WE"	' '	261="E "
261	260	"E "	'E'	262=" WEE"
257	261	"WE"	'W'	263="E W"
'B'	257	"B"	'B'	264="WEB"
260	'B'	" WE"	' '	260="B "
'T'	260	"T"	'T'	260=" WET"

Table 4.2: LZW decoding example

files, small dictionary sizes perform better compression than the large dictionaries, since for smaller files the codes are found initially in the dictionary and thus the dictionary does not fill up completely. Hence, sending less bits for the codes in the dictionary will be more efficient for compression. Consequently for short input and almost random input sequences, small dictionary sizes compress better, while for long and least random sequences, big dictionary sizes compress better[22]. A more efficient implementation is to increase the dictionary code size as the dictionary fills up with phrases[15]. This is the method used in this implementation of the LZW algorithm. In this case, the dictionary code size can start with 9 bits and can go upto the desired size.

Chapter 5

Analysis of Lossless Compression Algorithms in Vector Transform and Vector Subband Domains

5.1 Description of the Compression Environment

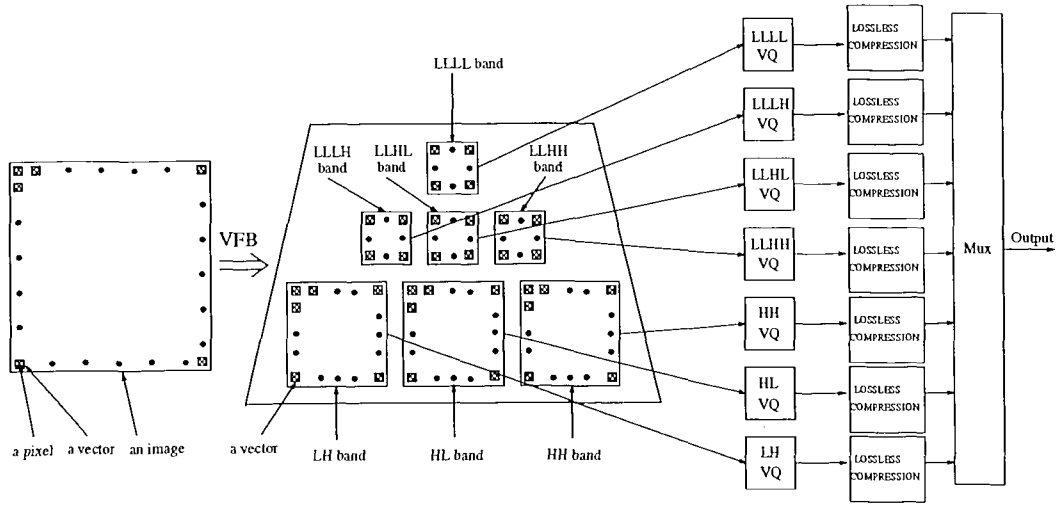

Figure 5.1: Block Diagram of the Vector Subband Coding Algorithm.

A simplified block diagram of vector subband coding algorithm can be visualized in figure 5.1. As seen in the figure, the image is decomposed into vector subbands using a vector filter bank and each subband is vector quantized [9]. In this diagram only seven subbands are shown. In our tests we will utilize 10 subbands for compression. Each one of subbands will be compressed through a lossless compression algorithm, Huffman, Adaptive Huffman or LZW in our case. The compressed data is sent through a presumably lossless communication channel, and decoded lossless in inverse manner and given to the decoding end of the vector subband or vector transform coding algorithm.

5.1.1 Source Characteristics

Basically, two images and a video sequence were utilized for the tests. The images are the well-known Lenna image and a training image which consisted of 16 different images. The video utilized is a 143 frame football video sequence.

The statistics taken on the images and the video sequence can be observed in figures 7.1 to 7.30 in the appendix. As mentioned previously, the image indices are obtained after vector subband transform of each image frame. For the video sequence, each index consists of a classification bit which shows the number of bits used in the index, and the index value. The classification bit in our case is 0 or 1. Classification bit 0 indicates an index value of 0, and 1 indicates an index value ranging from 0 to 4095, the maximum value attainable with 12 bits. For the images, there is no (0 0) index. The image indices range from 0 to 4095.

The figures 7.1 through 7.30 show the number of occurence sorted in descending order for the image indices. The first figures 7.1 through 7.10 are for the ten subbands of the football sequence. The rest of the figures 7.11 through 7.30 are histograms for the Lenna and training images respectively. For the video sequence, since the number of occurrence for the index (0 0) is much higher than the other indices and to adjust the scale so that the shape of the curve is clearly displayed, the zero index (0 0) is not shown on the graph, rather its number of occurence is indicated below the graph.

As observed from the curves, the occurence rate for the indices does not show a constant behavior, rather it drops for certain indices. This kind of behavior is essential for the entropy coding of data. Lossless compression algorithms utilized such as Huffman and Adaptive Huffman coding performs best if some indices occur more frequently than others, i.e. if the probability of occurence for some indices is more than others. Because, the algorithm can favor more frequent indices by assigning less bits and thus decrease the average number of bits sent per index. In LZW algorithm, the more the number of repeated patterns, the better the compression will be. Because basically, LZW algorithm achieves compresion by assigning fixed length codes to long sequence of repeated patterns.

As stated earlier, the lossless compression algorithms utilized in the tests are Huffman, Adaptive Huffman and Lempel-Ziv-Welch coding methods. In performing the tests, the results were obtained for each lossless algorithm seperately. The description of the algorithms and the discussion of the results will be presented next.

5.2 Huffman Coding

Huffman Coding Algorithm is developed based on the example about Huffman Coding in Chapter 2. The code is developed originally by the author of this thesis by following the steps involved in statistical modelling, source reduction and codeword construction sections for encoding. The Huffman table is developed in the encoding section, sent over the communication channel and utilized for decoding. The compressed data stream is also sent over the channel and decoded based on this Huffman table. The results of the tests by Huffman Coding will be shown next.

5.2.1 Discussion

Huffman Coding is applied to the two images and the video sequence. For the images, firstly, the Lenna image is utilized for taking the statistical information for building the Huffman table and it is compressed based on this Huffman table. Each

subband is encoded seperately.

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	7.649	7.688	1.560:1
1	7.699	7.719	1.555:1
2	7.047	7.094	1.692:1
3	7.343	7.375	1.627:1
4	8.676	8.695	1.380:1
5	7.613	7.648	1.569:1
6	8.191	8.211	1.461:1
7	8.775	8.803	1.363:1
8	8.920	7.951	1.509:1
9	8.416	8.443	1.421:1

Table 5.1: The Results of Huffman Coding for the Lenna Image

The results are shown in table 5.1 The compression ratios for the subbands range from 1.363:1 to 1.692:1. The average compression of the image for the total ten bands is **1.446:1**. As observed in the above table, the original entropy and the average number of bits used for the compressed file are very close. Theoretically, Huffman coding is the best *minimum-redundancy* coding algorithm. As also observed in the above results, this indicates that the Huffman algorithm compresses very close to the ideal entropy of the image. This result is mainly due to the fact that the same image is utilized for taking the statistics to build the Huffman table and the coding. The probability of occurence of the image indices represent very accurately the statistical model utilized for coding. Hence, this compression ratio performance is probably one of the best results we could get in compressing these files, since the results are very close to the entropy of the source.

In the second test, the training image is utilized for statistical modelling and building the Huffman table. The Lenna image is compressed based on this Huffman table.

The results are shown in table 5.2. The average compression of the Lenna image for the the ten bands is **1.275:1**. Compression ratios show improvement in the upper subbands due to the high correlation among the indices. However, the overall compression ratio shows a clear decrease from the previous Huffman coding results.

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	7.649	11.813	1.016:1
1	7.699	12.438	0.965:1
2	7.047	9.750	1.231:1
3	7.348	11.344	1.058:1
4	8.675	10.352	1.159:1
5	7.613	8.992	1.334:1
6	8.191	9.711	1.236:1
7	8.775	9.565	1.255:1
8	7.920	8.795	1.364:1
9	8.416	9.199	1.304:1

Table 5.2: The Results of Huffman Coding for the Lenna Image by training with the training image

This is mainly because of the fact that a different image is being utilized for training and thus the statistics obtained from the training image do not accurately represent the probabilities of the indices for the Lenna image. As observed from the above results, Huffman coding can at most compress as good as the entropy of the training image used for statistical modelling. If the statistical information of the training image is closer to the Lenna image, better compression will occur. The entropy of the Lenna image is much lower than the average bits utilized in transmission for the Lenna image, as opposed to the results shown in the previous table where the two values are very close. Hence Huffman compression performance is limited by the entropy of the training image utilized for taking the statistics.

Huffman coding is also applied to the 143 frame football video sequence. As mentioned previously, there is an extra index (0 0) in the video sequence. Firstly, this index is assigned a image index value of 4096 and treated like other indices in the Huffman coding. The video sequence is divided into 10 subbands where each subband contains from the 143 frames. Each one of the ten subbands is encoded seperately.

The results are shown in the table 5.3. The average compression ratio for the ten subbands is **1.085:1**. Even if the results are very close to the entropy of the video sequence, this test does not show much improvement in compression ratio. Since,

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	9.029	9.051	1.081:1
1	9.072	9.094	1.083:1
2	9.073	9.094	1.081:1
3	9.002	9.025	1.093:1
4	8.990	9.014	1.090:1
5	9.050	9.072	1.085:1
6	8.827	8.860	1.116:1
7	8.826	8.858	1.117:1
8	8.790	8.821	1.118:1
9	8.792	8.823	1.115:1

Table 5.3: Huffman Coding of the Video Sequence. Index (0 0) is included in the Huffman table.

when the algorithm is investigated, it is observed that Huffman algorithm assigns two bits to index (0 0). In the original file, this index is represented with only one bit, and the rest of the indices are represented by a 1-bit classification bit and 12 bit image index value.

In the second test of the video sequence, the Huffman algorithm is modified so that index (0 0) is assigned only one bit "0" and this index is excluded out of the Huffman table. The rest of the indices are Huffman coded . However, in order to distinguish between the index (0 0) and other indices, a 1-bit classification bit "1" is assigned prior to non-zero indices.

The results can be visualized in 5.4. The average compression ratio is **1.066:1**. The compression ratio for this test decreases instead of improvement. This is because of one bit classification bit added to distinguish the non-zero image indices. Since there seems to be more non-zero indices than the (0 0) index, one bit adds more overhead and thus reduces the compression ratio.

Subband Number	Minimum Compressed File Size	Compressed File Size	Compression Ratio
0	202377	202841	1.062:1
1	203515	203979	0.063:1
2	203466	203954	1.061:1
3	50507	50627	1.073:1
4	50411	50524	1.071:1
5	50753	50873	1.066:1
6	12391	12419	1.096:1
7	12389	12416	1.097:1
8	12334	12362	1.099:1
9	12333	12361	1.088:1

Table 5.4: Huffman Coding of the video sequence. Index (0 0) is excluded from the Huffman Table.

5.3 Adaptive Huffman Coding

For Adaptive Huffman Coding, the algorithm from Mark Nelson's "The Data Compression Book" [15] is utilized. The program is designed originally for text compression to process ASCII characters. Since a different image source is utilized in this case, the program has to be modified. The program is altered to read 12-bit data (from 0 to 4095) and the control characters such as those showing the beginning and end of the data bit stream. The major difference between the Huffman and Adaptive Huffman algorithms is that Adaptive Huffman Coding builds the dictionary adaptively *during* encoding and decoding phases, while Huffman Coding builds the dictionary statistically *prior* to encoding.

5.3.1 Discussion

For Adaptive Huffman Coding, since the building of the Huffman table or tree is in real time, there is no prior statistical information that needs to stored. Hence, the training image is not needed for building the Huffman table. Instead, the Lenna image is compressed based on the statistical information of its own image indices.

Adaptive Huffman Coding algorithm was applied to the Lenna indices at each subband of vector subband coding.

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	7.649	16.625	0.722:1
1	7.699	16.875	0.711:1
2	7.047	14.000	0.857:1
3	7.348	15.281	0.785:1
4	8.675	14.516	0.827:1
5	7.613	11.750	1.021:1
6	8.191	13.602	0.882:1
7	8.775	11.586	1.036:1
8	7.920	10.102	1.188:1
9	8.416	11.006	1.090:1

Table 5.5: The Results of Adaptive Huffman Coding For the Lenna Image

The results are shown in table 5.5. The average compression for the image is **1.03:1**. Adaptive Huffman algorithm can not perform as well as Huffman algorithm. Especially for lower subbands, the algorithm expands the binary index files instead of compressing them. I think this is due to the fact that Adaptive Huffman algorithm can not build enough statistical information to code the binary files efficiently. Especially, for low order bands, the order that each symbol is being read from the file affects the performance because the order of the input data builds the statistical Huffman tables and thus it is essential in assigning the right code for the compression. Hence, overall, the compression ratio is very small and there is hardly any compression.

For the video sequence, similar to the Huffman coding, initially index (0 0) is treated as a symbol and it is encoded in the algorithm.

The results are shown in the table 5.6. The overall compression ratio for the ten bands is **1.023:1**. Thus, Adaptive Huffman compression is not as good as Huffman compression and compression ratio is very low. Adaptive Huffman coding can not perform as good as Huffman coding, since it builds the dictionary in real time while it is encoding the image indices. As mentioned earlier, initial statistical information

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	9.029	9.291	1.053:1
1	9.072	9.338	1.054:1
2	9.073	9.340	1.052:1
3	9.002	10.057	0.981:1
4	8.990	10.062	0.977:1
5	9.050	10.131	0.972:1
6	8.827	11.699	0.846:1
7	8.826	11.628	0.851:1
8	8.790	11.541	0.855:1
9	8.792	11.59	0.849:1

Table 5.6: Adaptive Huffman Coding of the Video Sequence. Index (0 0) is included in the Huffman table.

is not enough to assign the right codes to the indices. However, major advantage of Adaptive Huffman Coding is that building of the dictionary is not needed prior to encoding, and thus the dictionary does not have to be sent to the other side before encoding is started.

In the second test with the video sequence, similar to Huffman Coding, index (0 0) is coded with a fixed one bit "0" and the rest of the indices are encoded regularly. To distinguish the non-zero indices from index (0 0), a one bit "1" has to be sent prior to non-zero indices.

The results are shown in table 5.7. For this case, similar to the tests in Huffman coding for the video sequence, the compression ratio does not improve, but decreases. The overall compression ratio for the ten subbands is **1.002:1**. Hence, we could say that Adaptive Huffman algorithm has almost no effect in compressing this video sequence.

5.4 LZW Coding

For LZW coding, the program from Mark Nelson's "The Data Compression Book" [15]is modified based on the input data. Similar to the Adaptive Huffman Coding, the algorithm is modified to compress the image indices ranging from 0 to 4096.

Subband Number	Minimum Compressed File Size	Compressed File Size	Compression Ratio
0	202377	209197	1.030:1
1	203515	210386	1.031:1
2	203466	210307	1.029:1
3	50507	56390	0.963:1
4	50411	56354	0.960:1
5	50753	56809	0.954:1
6	12391	16281	0.836:1
7	12389	16253	0.938:1
8	12334	16158	0.840:1
9	12333	16202	0.836:1

Table 5.7: Adaptive Huffman Coding of the video sequence. Index (0 0) is excluded from the Huffman Table

Initially, the dictionary is assumed to contain all the individual numbers from 0 to 4096.

The program code also gives the ability to modify the number of bits used to build the dictionary. In our implementation we started with 12 bits and varied up to 15 bits. Varying the dictionary bit size rather than keeping it fixed seems to improve the compression performance.

5.4.1 Discussion

The LZW algorithm is applied to the indices obtained after vector subband coding of the Lenna image, each subband is compressed seperately with LZW algorithm. Since the statistical information is gathered in real time like Adaptive Huffman coding. The dictionary is built on the fly on both encoding and decoding ends of the compression algorithm. Hence, there is no need for prior building of the dictionary based on the training image.

As seen from table 5.8, the performance of the LZW algorithm is poorer than the Huffman algorithm, but very close to the Adaptive Huffman compression. The average compression for the image is **1.02:1**. LZW coding is an adaptive compression technique like Adaptive Huffman Coding. Hence the input data information in the

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	7.649	13.0625	0.919:1
1	7.699	13.000	0.923:1
2	7.047	12.813	0.937:1
3	7.348	13.000	0.923:1
4	8.675	12.836	0.935:1
5	7.613	11.984	1.001:1
6	8.191	12.280	0.977:1
7	8.775	11.824	1.015:1
8	7.920	11.039	1.087:1
9	8.416	11.680	1.027:1

Table 5.8: The Results of LZW Coding for the Lenna Image

lower bands is not enough to accumulate accurate statistics. Hence, for lower bands, expansion rather than compression occurs. In the higher bands better performance is observed, since more data is collected to build accurate statistical models and the indices are highly correlated.

Also, this poor performance of the LZW algorithm is due to the fact that it depends on the probability of the occurence of sequence of symbols rather than the occurence of the single symbol itself. Huffman algorithm makes better use of the single symbol probability of occurence. LZW algorithm performs better, if the probability of the occurrence for the repeated index sequences is high.

LZW and Adaptive Huffman algorithms have very close compression ratios, since they both use adaptive techniques for statistical analysis. They both improve their compression performance as the input data increases.

Similar to previous coding algorithms, the football video sequence is compressed in two ways for LZW coding. Firstly, extra index (0 0) is treated like other non-zero indices and included as an index in compression.

The results indicate no compression but expansion. The overall ratio is **0.915:1**. When index (0 0) is treated seperately from other non-zero indices and a fixed one bit "0" is sent as a code, the compression ratio does not improve but decreases.

The results get poorer for this case which indicates an overall expansion of

Subband Number	Ideal Entropy	Average Index Length	Compression Ratio
0	9.029	10.675	0.916:1
1	9.072	10.730	0.918:1
2	9.073	10.726	0.916:1
3	9.002	10.824	0.911:1
4	8.990	10.769	0.913:1
5	9.050	10.796	0.912:1
6	8.827	10.999	0.899:1
7	8.826	11.007	0.899:1
8	8.790	10.978	0.899:1
9	8.792	10.978	0.896:1

Table 5.9: LZW Coding of the Video Sequence. Index (0 0) is included in LZW coding.

0.889:1. Because when sequence of (0 0) indices occur in the middle of the non-zero sequences, the sequence of non-zero indices built in the dictionary have to be cut short, leading to shorter non-zero sequences and thus decrease in compression. Hence, LZW algorithm can not compress for the video sequence and it does not seem well-suited for the video compression.

The summary of the compression performance of the three algorithms can be visualized in the tables 5.11 and 5.12. As observed in the tables for the video sequence, Huffman coding seems to be the best compression method among the three algorithms in terms of compression ratio. It gives results very close to the entropy of the source. Even if the compression for Huffman coding is not high, this results indicate a good compression ratio that can be achieved for lossless coding. For Adaptive Huffman coding, the results are worse than Huffman algorithm compression. The compression ratio is very low. LZW coding expands the files rather than compression and does not seem appropriate for the image data utilized.

Subband Number	Minimum Compressed File Size	Compressed File Size	Compression Ratio
0	202377	242836	0.887:1
1	203515	244236	0.888:1
2	203466	244090	0.887:1
3	50507	61454	0.884:1
4	50411	61129	0.885:1
5	50753	61392	0.883:1
6	12391	14722	0.925:1
7	12389	14707	0.926:1
8	12334	14701	0.924:1
9	12333	14539	0.931:1

Table 5.10: LZW Coding of the video sequence. Index (0 0) is excluded from LZW coding.

Coding Method	Compression Ratio
Huffman Coding	1.085:1
Adaptive Huffman Coding	1.023:1
LZW Coding	0.915:1

Table 5.11: Summary of coding results of the video sequence with index (0 0) treated as a regular image index.

Coding Method	Compression Ratio
Huffman Coding	1.066:1
Adaptive Huffman Coding	1.002:1
LZW Coding	0.889:1

Table 5.12: Summary of coding results of the video sequence with index (0 0) treated seperately from other image indices.

Chapter 6

Conclusions

In this study, the three lossless compression algorithms are analyzed and compared in terms of their compression performance. The algorithms are Huffman coding, Adaptive Huffman coding and Lempel-Ziv-Welch coding algorithms. Two images, the Lenna and the training image and a football video sequence are utilized as input source for the compression algorithms.

After the tests, it's observed that Huffman coding performs better than Adaptive Huffman and LZW coding algorithms in terms of compression ratio for the images and the video sequence. If the same image is utilized for training and coding, the Huffman algorithm gives results very close to its entropy. If a different image is utilized for training in Huffman coding, the compression ratio decreases but it is still better than the other two algorithms.

Adaptive Huffman coding has very low compression ratio for the Lenna image, since it builds the Huffman table in real time and especially for the lower bands, there are not enough indices to represent accurately the probability of occurrence for the indices.

Similarly, LZW algorithm has also very low compression for the Lenna image. For the Lenna image, LZW algorithm's compression ratio is very close to Adaptive Huffman's compression ratio. This is because LZW algorithm like Adaptive Huffman coding builds the dictionary adaptively in real time and there is a lack of statistical data to build the dictionary accurately.

For the video sequence, Huffman coding gives again the best performance among the three algorithms. For this case, Adaptive Huffman algorithm comes next with a lower compression ratio while LZW algorithm expands the image files instead of compression. Huffman compression results are very close to the entropy of the source and even if the compression ratio is low, it achieves a good compression performance due to the entropy limitations. The LZW algorithm does not seem appropriate for compression of this video sequence.

Hence in order to achieve better compression some future work is suggested. In Huffman coding, the least probable indices might be coded with an escape sequence and a fixed length code, rather than including them in the statistical analysis for the Huffman table. The long sequence of (0 0) indices can be manipulated differently from the non-zero indices. A further study can be the run-length encoding (RLE) of these indices along with entropy or LZW coding.

Chapter 7

Appendix

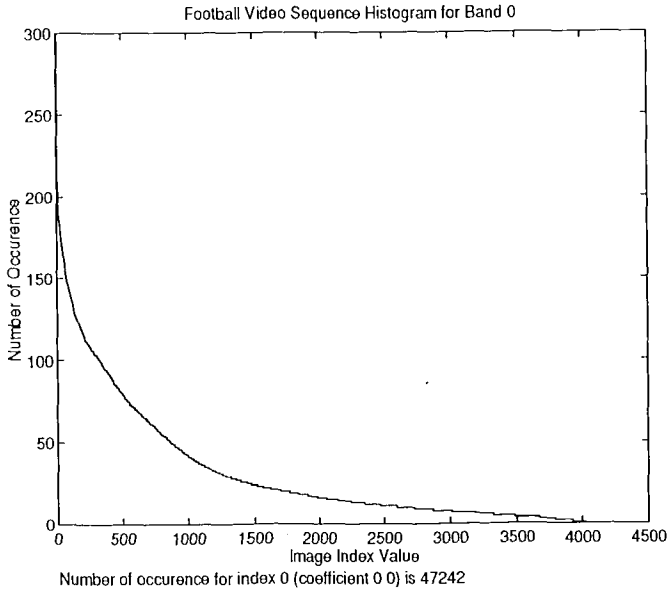

Figure 7.1: The frequency of occurence in descending order for the image indices in band 0 of the football video sequence.

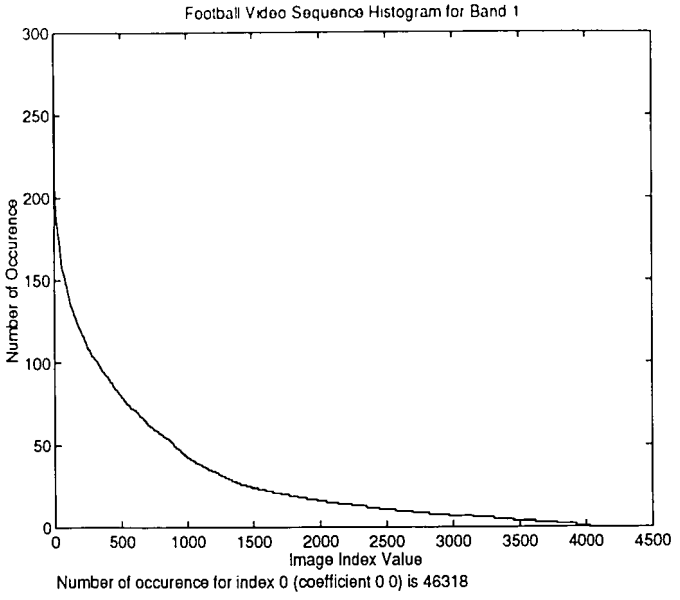

Figure 7.2: The frequency of occurence in descending order for the image indices in band 1 of the football video sequence.

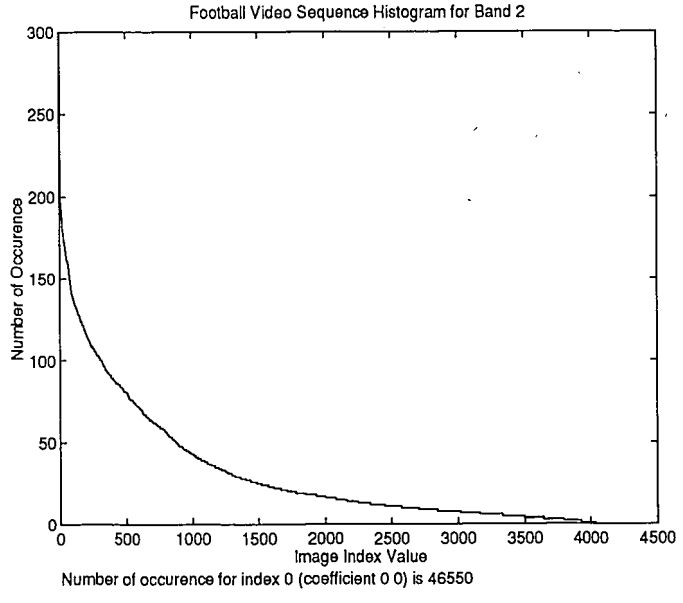

Figure 7.3: The frequency of occurence in descending order for the image indices in band 2 of the football video sequence.

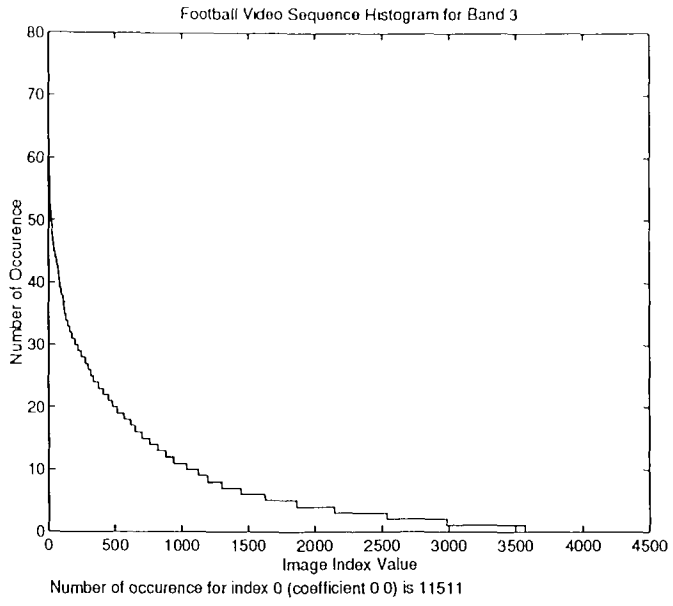

Figure 7.4: The frequency of occurence in descending order for the image indices in band 3 of the football video sequence.

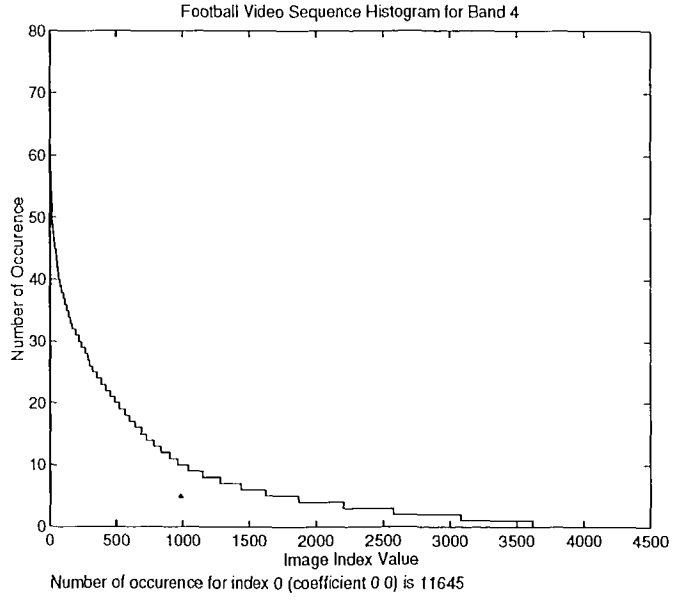

Figure 7.5: The frequency of occurence in descending order for the image indices in band 4 of the football video sequence.

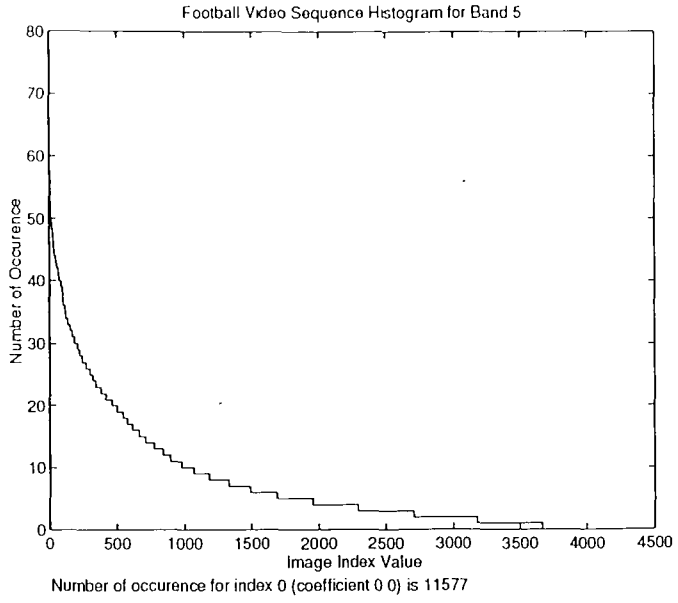

Figure 7.6: The frequency of occurence in descending order for the image indices in band 5 of the football video sequence.

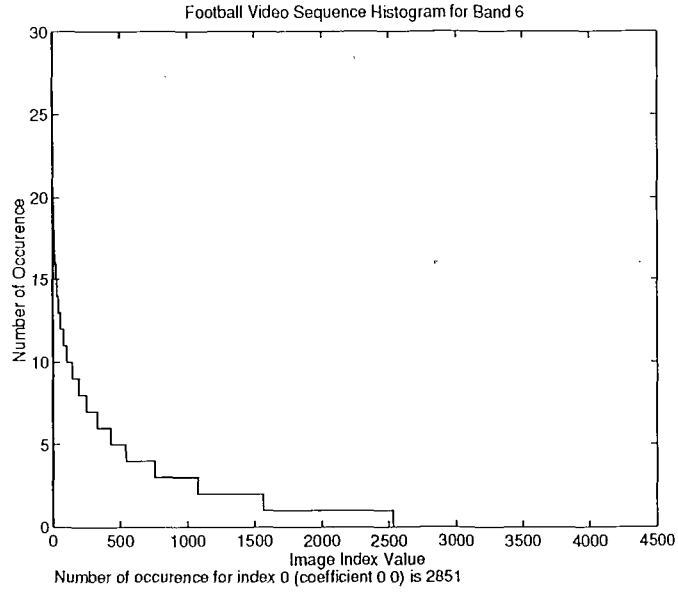

Figure 7.7: The frequency of occurence in descending order for the image indices in band 6 of the football video sequence.

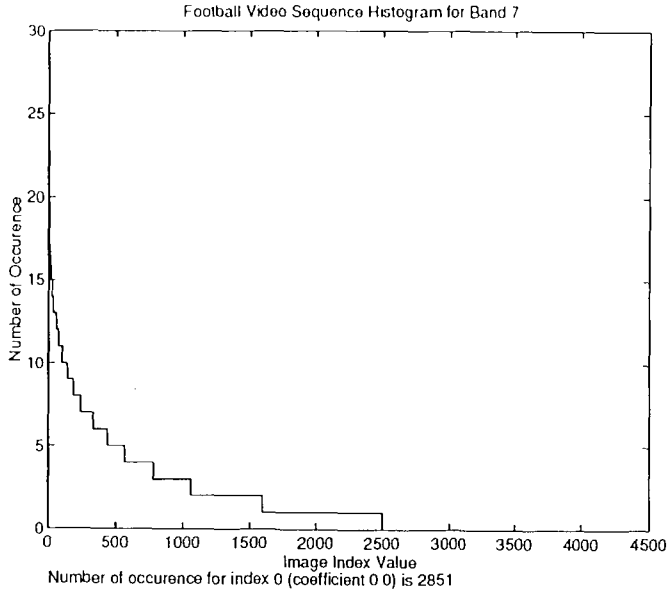

Figure 7.8: The frequency of occurence in descending order for the image indices in band 7 of the football video sequence.

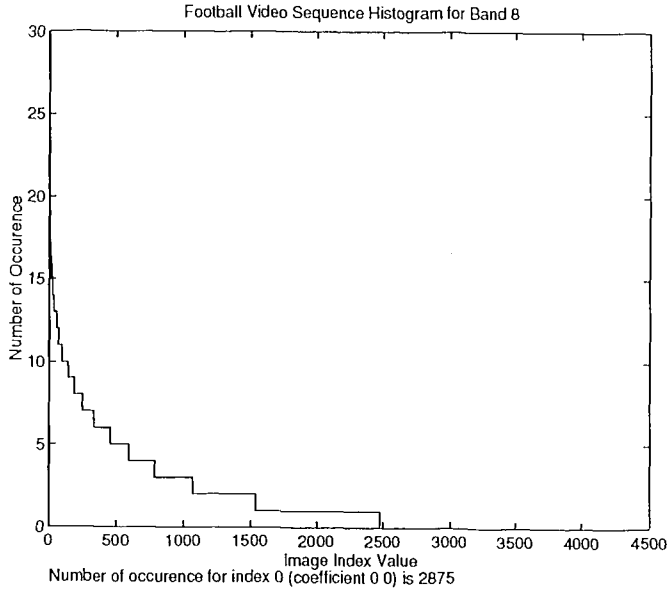

Figure 7.9: The frequency of occurence in descending order for the image indices in band 8 of the football video sequence.

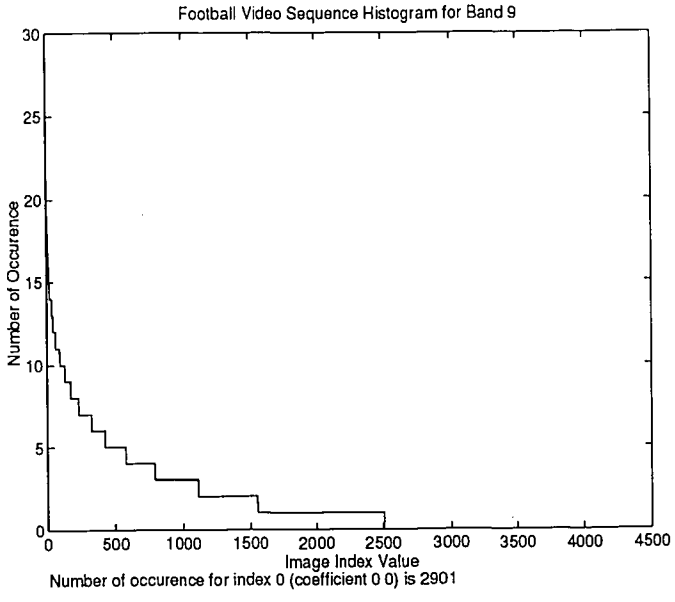

Figure 7.10: The frequency of occurence in descending order for the image indices in band 9 of the football video sequence.

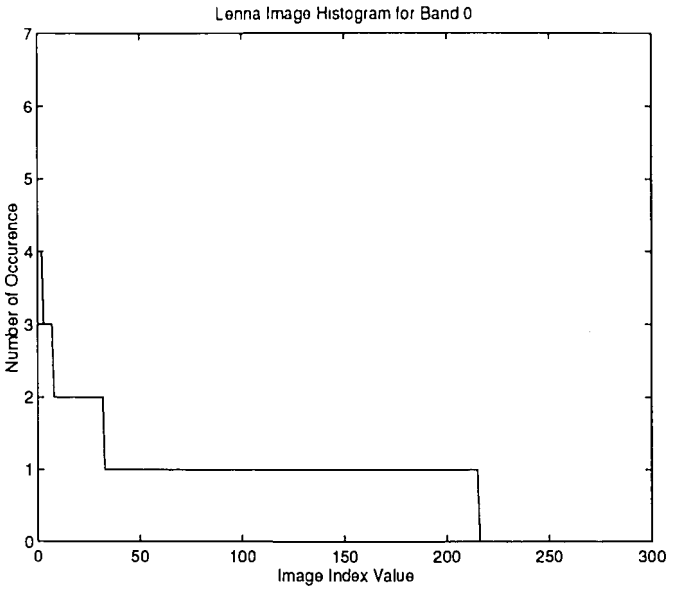

Figure 7.11: The frequency of occurence of the indices in descending order for band 0 of the Lenna image.

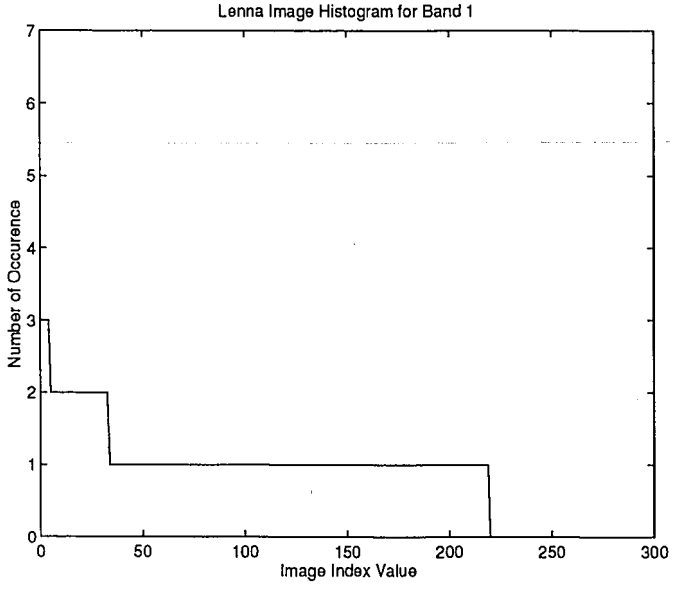

Figure 7.12: The frequency of occurence of the indices in descending order for band 1 of the Lenna image.

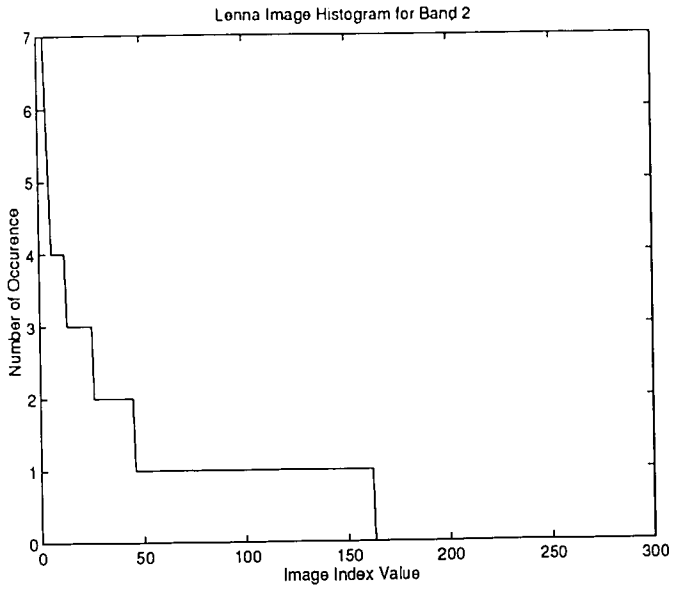

Figure 7.13: The frequency of occurence of the indices in descending order for band 2 of the Lenna image.

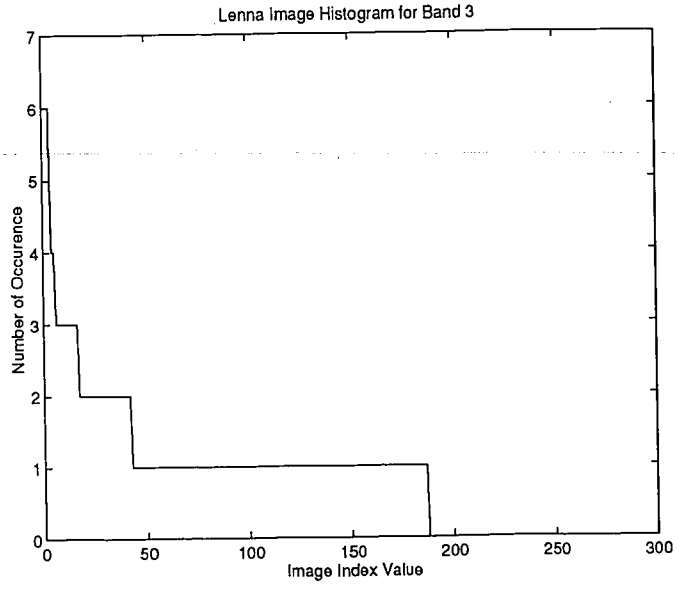

Figure 7.14: The frequency of occurence of the indices in descending order for band 3 of the Lenna image.

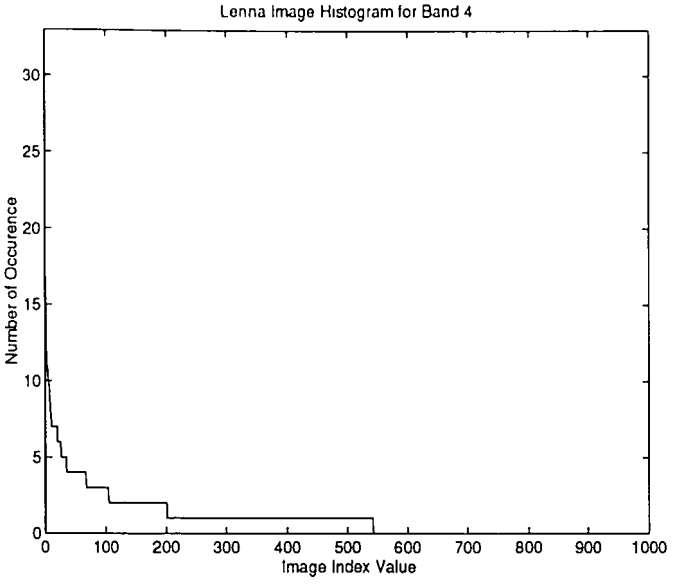

Figure 7.15: The frequency of occurence of the indices in descending order for band 4 of the Lenna image.

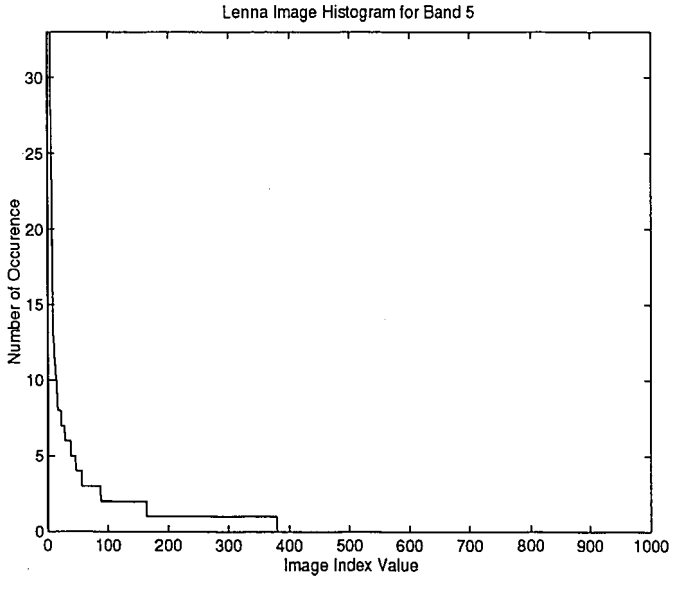

Figure 7.16: The frequency of occurence of the indices in descending order for band 5 of the Lenna image.

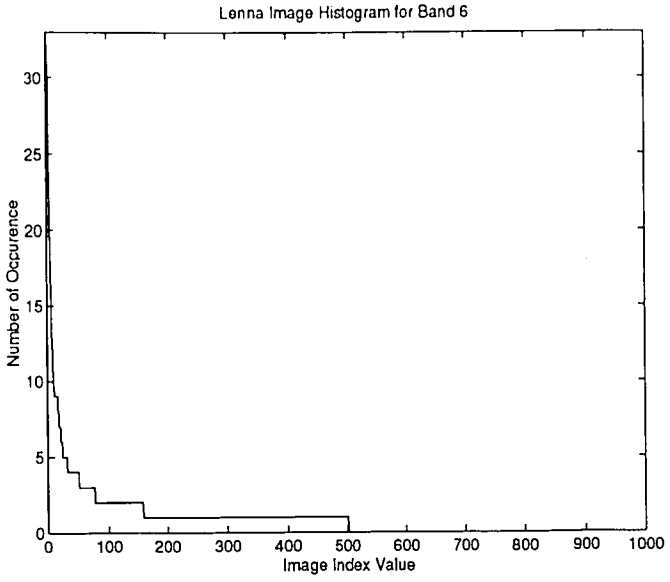

Figure 7.17: The frequency of occurence of the indices in descending order for band 6 of the Lenna image.

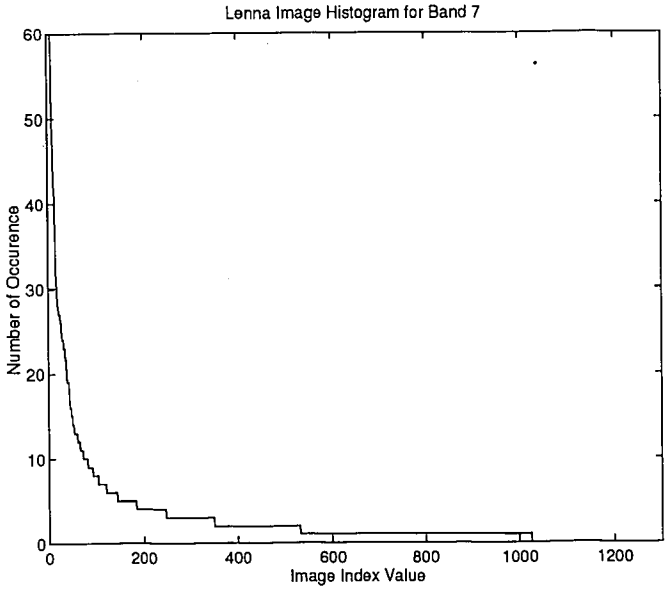

Figure 7.18: The frequency of occurence of the indices in descending order for band 7 of the Lenna image.

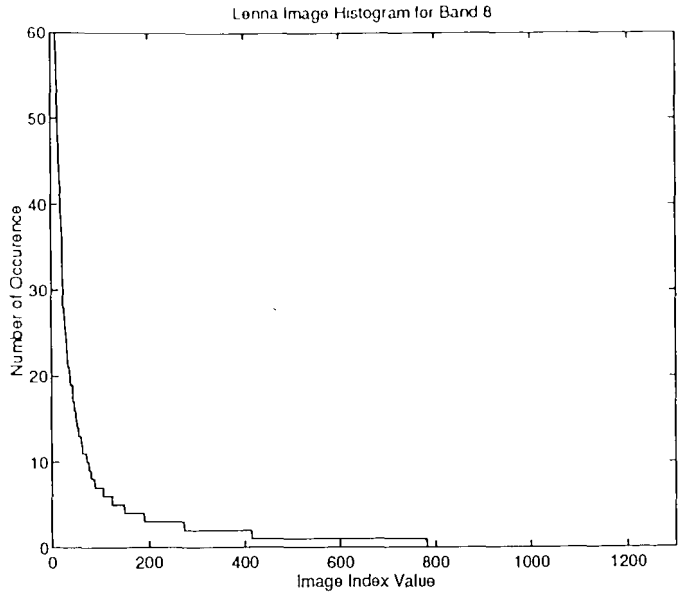

Figure 7.19: The frequency of occurence of the indices in descending order for band 8 of the Lenna image.

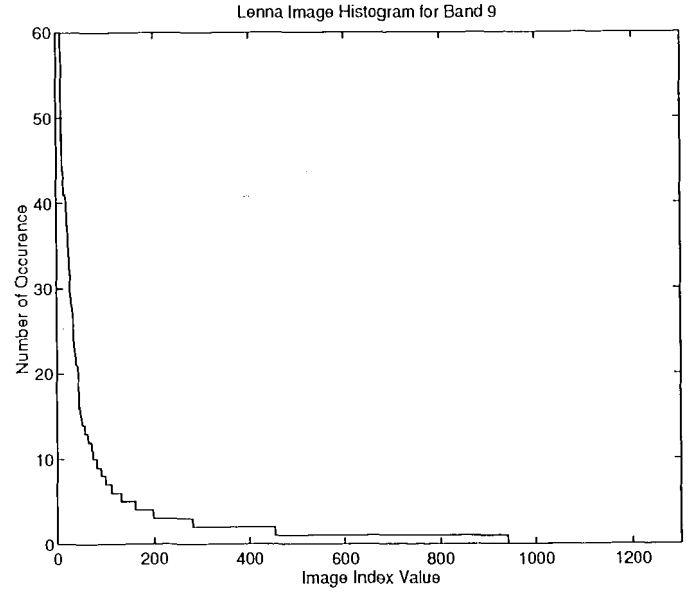

Figure 7.20: The frequency of occurence of the indices in descending order for band 9 of the Lenna image.

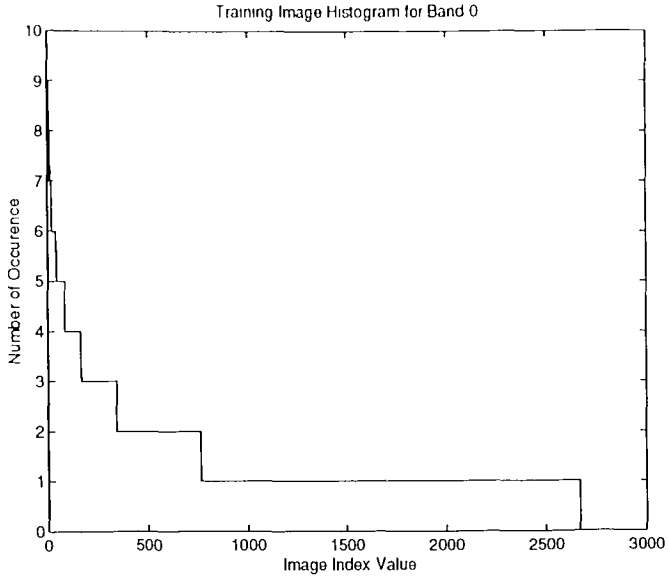

Figure 7.21: The frequency of occurence of the indices in descending order for band 0 of the training image.

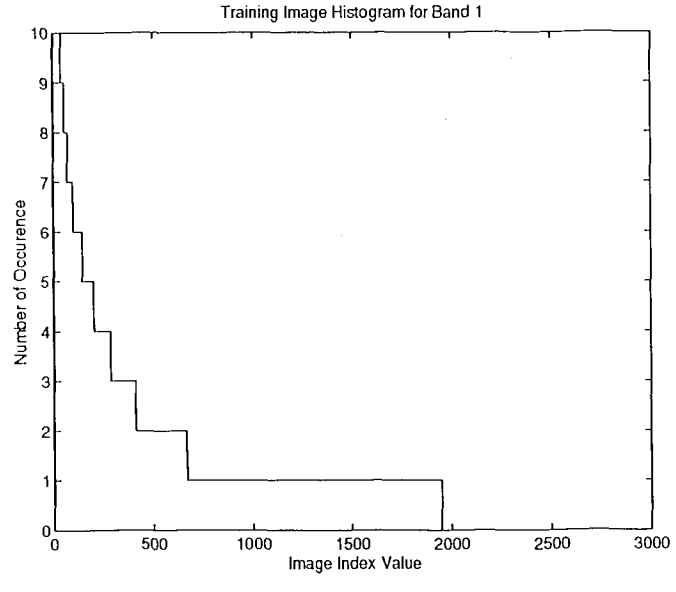

Figure 7.22: The frequency of occurence of the indices in descending order for band 1 of the Lenna image.

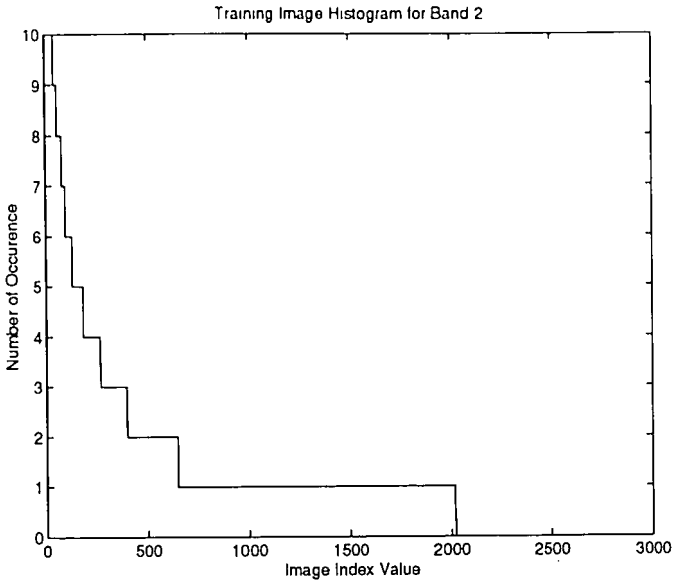

Figure 7.23: The frequency of occurence of the indices in descending order for band 2 of the Lenna image.

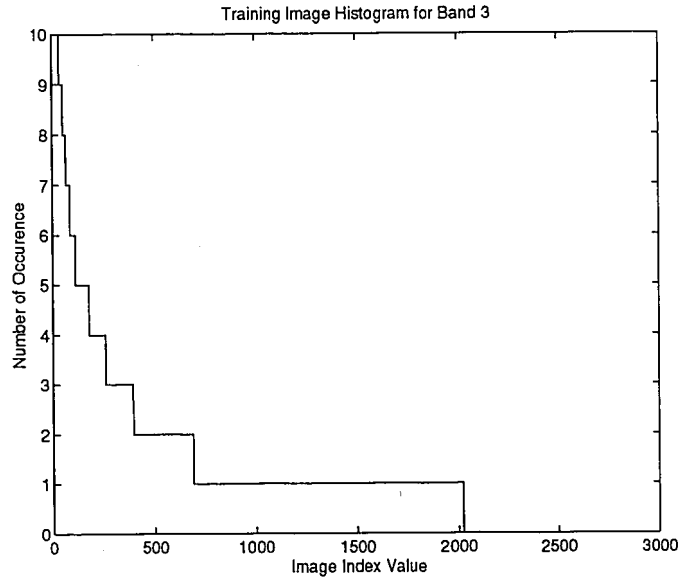

Figure 7.24: The frequency of occurence of the indices in descending order for band 3 of the Lenna image.

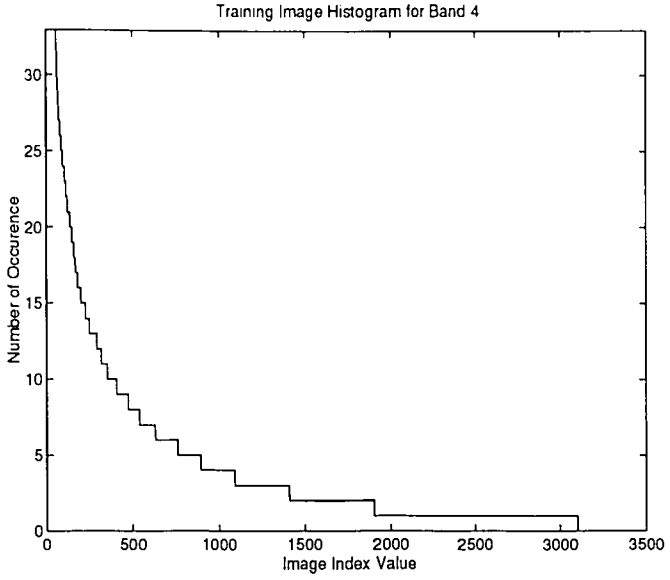

Figure 7.25: The frequency of occurence of the indices in descending order for band 4 of the Lenna image.

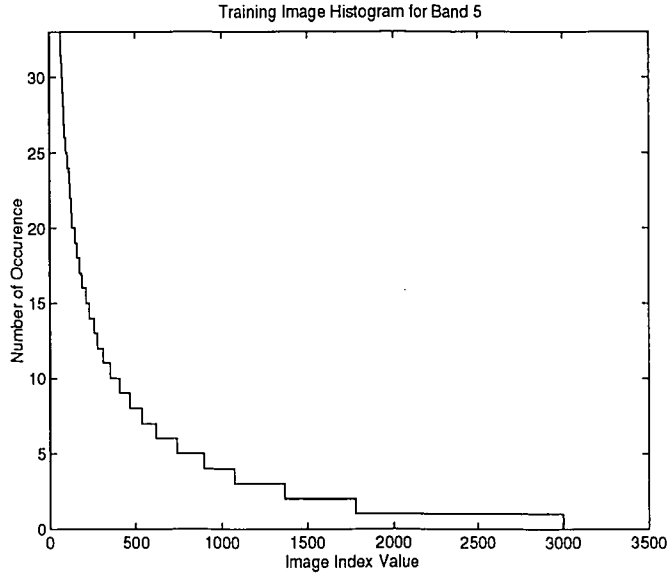

Figure 7.26: The frequency of occurence of the indices in descending order for band 5 of the Lenna image.

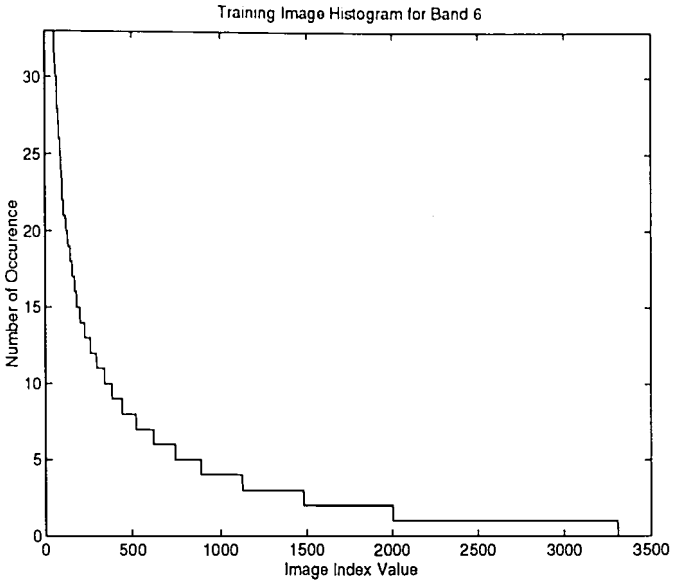

Figure 7.27: The frequency of occurence of the indices in descending order for band 6 of the Lenna image.

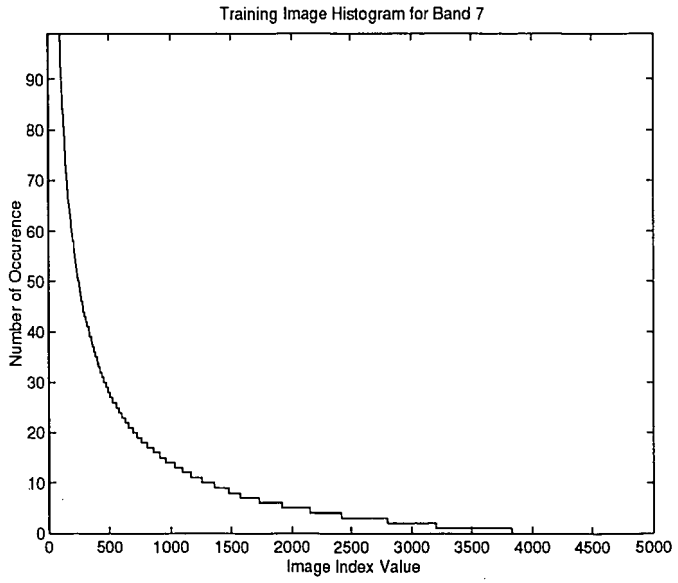

Figure 7.28: The frequency of occurence of the indices in descending order for band 7 of the Lenna image.

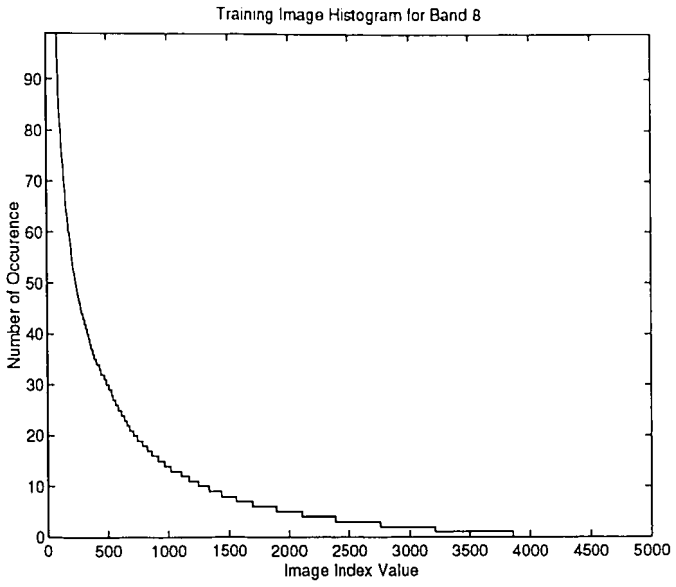

Figure 7.29: The frequency of occurence of the indices in descending order for band 8 of the Lenna image.

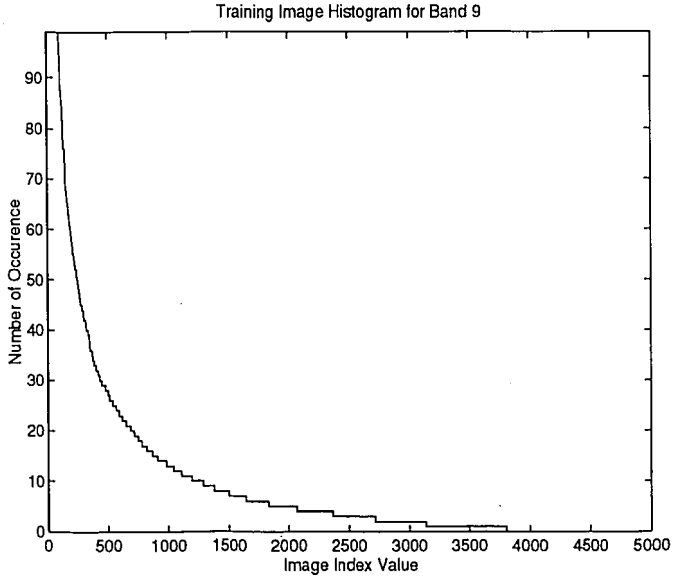

Figure 7.30: The frequency of occurence of the indices in descending order for band 9 of the Lenna image.

Bibliography

[1] R.C. Gonzalez, *Digital Image Processing*, Addison Wesley, 1992.

[2] C.E. Shannon, "The Mathematical Theory of Communication," Urbana IL:Univ. Illinois Press, 1949.

[3] A.N. Netrevali, J.O. Limb, "Picture Coding: A Review," Proceedings of the IEEE, Vol. 68, No. 3, March 1980, pp. 366-403.

[4] Weiping Li, Y.Q. Zhang, "Vector-based signal processing and quantization for image and video compression", Proceedings of the IEEE, Vol. 83, No. 4, February 1995, pp. 317 - 335.

[5] A.K. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ:Prentice Hall, 1989.

[6] M. Rabbani, P.W. Jones, *Digital Image Compression Techniques*, Bellingham,WA:SPIE Optical Engineering Press, 1991.

[7] M. Rabbani, B.J. Thompson, *Selected Papers on Image Coding and Compression*, Bellingham WA:SPIE Optical Engineering Press, 1992.

[8] Weiping Li, "Vector Transform and Image Coding", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 1, No. 4, December 1991, pp. 297 - 307.

[9] John Peter Wus, *Vector Subband Coding of Images*, Master Thesis, Lehigh University, December 1994.

[10] Ahmed Y. Banafa, *A Comparative Study of Image Compression Techniques within a Noisy Channel Environment* , Master Thesis, Lehigh University, May 1993.

[11] Gregory K. Wallace, "The JPEG Still Picture Compression Standard", Communications of the ACM, Vol. 34, No. 4, April 1991, pp. 30 - 44.

[12] Didier Le Gall, "MPEG: a video compression standard for multimedia applications", Communications of the ACM, Vol. 34, 1991, pp. 46 - 58.

[13] T. M. Cover, J.A. Thomas, *Elements of Information Theory*, John Wiley&Sons, Inc. 1991.

[14] T. Bell, I.H. Witten, J.G. Cleary, "Modeling for Text Compression," ACM Computing Surveys, Vol. 21, No. 4, December 1989, pp. 557-591.

[15] Mark Nelson, *The Data Compression Book*, San Mateo, CA:M&T Books, 1992.

[16] Tirso Alonso, *Digital Image Compression*, Master Thesis, Lehigh University, May 1992.

[17] David A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes,'*Proceeedings of the IRE*, Vol. 40(10), September 1952, pp. 1098-1101.

[18] D.A. Lelewer, D.S. Hirschberg, "Data Compression," *ACM Computing Surveys*, Vol. 19, No. 3, September 1987, pp. 261-296.

[19] J. Ziv, A. Lampel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory* 23, 3(May) 1977, pp. 337-343.

[20] J. Ziv, A. Lampel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory* 24, 5(Sept.) 1978, pp. 530-536.

[21] T. Welch, "A Technique for High-Performance Data Compression," *Computer*, June 1984, pp. 8-19.

[22] C.G.Boncelet Jr., J.R. Cobbs, A.R. Moser, "Error Free Compression of Medical X-Ray Images," *Visual Communications and Image Processing, Proc. SPIE*, Vol. 1001, 1988, pp. 269-276.

[23] James A. Storer, *Image and Text Compression*, Norwell, Massachusetts:Kluwer Academic Publishers, 1992.

Appendix A

Biography

Asaf M. Sofu was born in Ankara, Turkey in 1970. He attended Drexel University, Philadelphia PA in September, 1988. During his undergraduate studies, he held three cooperative education assignments at different locations in industry, namely, Smithkline & Beecham R&D Labs, Siemens Research Corporation, and Unisys Corporation. He was also an undergraduate research assistant in Signal Processing. He was awarded the Arthur Von Neuman scholarship and graduated Cum Laude with a B.S. degree in Electrical and Computer Engineering in 1993. In the Fall of 1993, he joined Lehigh University for M.S. degree in Electrical Engineering.

END

OF

TITLE