UNIT-5

Wavelets and Multi-resolution Processing

Image Compression

Introduction

In recent years, there have been significant advancements in algorithms and architectures for the processing of image, video, and audio signals. These advancements have proceeded along several directions. On the algorithmic front, new techniques have led to the development of robust methods to reduce the size of the image, video, or audio data. Such methods are extremely vital in many applications that manipulate and store digital data. Informally, we refer to the process of size reduction as a compression process. We will define this process in a more formal way later. On the architecture front, it is now feasible to put sophisticated compression processes on a relatively low-cost single chip; this has spurred a great deal of activity in developing multimedia systems for the large consumer market.

One of the exciting prospects of such advancements is that multimedia information comprising image, video, and audio has the potential to become just another data type. This usually implies that multimedia information will be digitally encoded so that it can be manipulated, stored, and transmitted along with other digital data types. For such data usage to be pervasive, it is essential that the data encoding is standard across different platforms and applications. This will foster widespread development of applications and will also promote interoperability among systems from different vendors. Furthermore, standardisation can lead to the development of cost-effective implementations, which in turn will promote the widespread use of multimedia information. This is the primary motivation behind the emergence of image and video compression standards.

Compression is a process intended to yield a compact digital representation of a signal. In the literature, the terms *source coding, data compression, bandwidth compression,* and *signal compression* are all used to refer to the process of compression. In the cases where the signal is defined as an image, a video stream, or an audio signal, the generic problem of compression is to minimise the bit rate of their digital representation. There are many applications that benefit when image, video, and audio signals are available in compressed form. **Without compression, most of these applications would not be feasible!**

Example 1: Let us consider **facsimile image transmission**. In most facsimile machines, the document is scanned and digitised. Typically, an 8.5x11 inches page is scanned at 200 dpi;

thus, resulting in 3.74 Mbits. Transmitting this data over a low-cost 14.4 kbits/s modem would require 5.62 minutes. With compression, the transmission time can be reduced to 17 seconds. This results in substantial savings in transmission costs.

Example 2: Let us consider a video-based CD-ROM application. **Full-motion video**, at 30 fps and a 720 x 480 resolution, generates data at 20.736 Mbytes/s. At this rate, only 31 seconds of video can be stored on a 650 MByte CD-ROM. Compression technology can increase the storage capacity to 74 minutes, for VHS-grade video quality.

Image, video, and audio signals are amenable to compression due to the factors below. There is considerable statistical redundancy in the signal.

1. Within a single image or a single video frame, there exists significant correlation among neighbour samples. This correlation is referred to as *spatial correlation.*
2. For data acquired from multiple sensors (such as satellite images), there exists significant correlation amongst samples from these sensors. This correlation is referred to as *spectral correlation.*
3. For temporal data (such as video), there is significant correlation amongst samples in different segments of time. This is referred to as *temporal correlation.*

The term data compression refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between data and information. They are not synonymous. In fact, data are the means by which information is conveyed. Various amounts of data may be used to represent the same amount of information. Such might be the case, for example, if a long-winded individual and someone who is short and to the point were to relate the same story. Here, the information of interest is the story; words are the data used to relate the information. If the two individuals use a different number of words to tell the same basic story, two different versions of the story are created, and at least one includes nonessential data. That is, it contains data (or words) that either provide no relevant information or simply restate that which is already known. It is thus said to contain data redundancy.

Data redundancy is a central issue in digital image compression. It is not an abstract concept but a mathematically quantifiable entity. If n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information, the relative

data redundancy R_D of the first data set (the one characterized by n_1) can be defined as

$$R_D = 1 - \frac{1}{C_R}$$

where C_R , commonly called the compression ratio, is

$$C_R = \frac{n_1}{n_2}.$$

For the case $n_2 = n_1$, $C_R = 1$ and $R_D = 0$, indicating that (relative to the second data set) the first representation of the information contains no redundant data. When $n_2 << n_1$, $C_R \to \infty$ and $R_D \to 1$, implying significant compression and highly redundant data. Finally, when $n_2 >> n_1$, $C_R \to 0$ and $R_D \to \infty$, indicating that the second data set contains much more data than the original representation. This, of course, is the normally undesirable case of data expansion. In general, C_R and R_D lie in the open intervals $(0,\infty)$ and $(-\infty, 1)$, respectively. A practical compression ratio, such as 10 (or 10:1), means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second or compressed data set. The corresponding redundancy of 0.9 implies that 90% of the data in the first data set is redundant.

In digital image compression, three basic data redundancies can be identified and exploited: **coding redundancy**, **interpixel redundancy**, and **psychovisual redundancy**. Data compression is achieved when one or more of these redundancies are reduced or eliminated.

Coding Redundancy:

In this, we utilize formulation to show how the gray-level histogram of an image also can provide a great deal of insight into the construction of codes to reduce the amount of data used to represent it.

Let us assume, once again, that a discrete random variable r_k in the interval [0, 1] represents the gray levels of an image and that each r_k occurs with probability $p_r(r_k)$.

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L-1$$

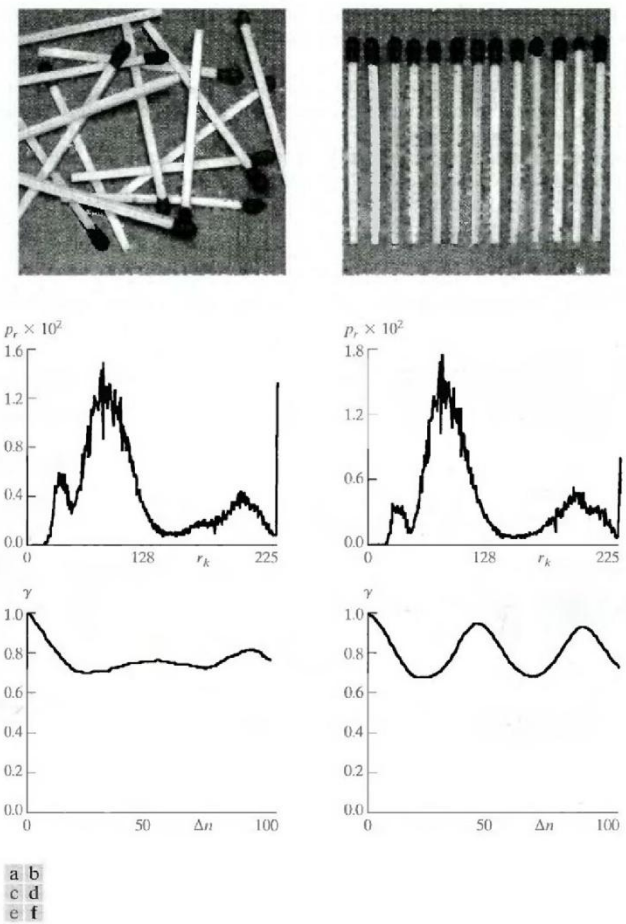
where L is the number of gray levels, n_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k).$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an M X N image is MNL_{avg}.

Interpixel Redundancy:

Consider the images shown in Figs. 1.1(a) and (b). As Figs. 1.1(c) and (d) show, these images have virtually identical histograms. Note also that both histograms are trimodal, indicating the presence of three dominant ranges of gray-level values. Because the gray levels in these images are not equally probable, variable-length coding can be used to reduce the coding redundancy that would result from a straight or natural binary encoding of their pixels. The coding process, however, would not alter the level of correlation between the pixels within the images. In other words, the codes used to represent the gray levels of each image have nothing to do with the correlation between pixels. These correlations result from the structural or geometric relationships between the objects in the image.



a b
c d
e f

Fig.1.1 Two images and their gray-level histograms and normalized autocorrelation coefficients along one line.

Figures 1.1(e) and (f) show the respective autocorrelation coefficients computed along one line of each image.

$$\gamma(\Delta n) = \frac{A(\Delta n)}{A(0)}$$

where

$$A(\Delta n) = \frac{1}{N - \Delta n} \sum_{y=0}^{N-1-\Delta n} f(x, y) f(x, y + \Delta n).$$

The scaling factor in Eq. above accounts for the varying number of sum terms that arise for each integer value of n. Of course, n must be strictly less than N, the number of pixels on a line. The variable x is the coordinate of the line used in the computation. Note the dramatic difference between the shape of the functions shown in Figs. 1.1(e) and (f). Their shapes can be qualitatively related to the structure in the images in Figs. 1.1(a) and (b).This relationship is particularly noticeable in Fig. 1.1 (f), where the high correlation between pixels separated by 45 and 90 samples can be directly related to the spacing between the vertically oriented matches of Fig. 1.1(b). In addition, the adjacent pixels of both images are highly correlated. When n is 1, γ is 0.9922 and 0.9928 for the images of Figs. 1.1 (a) and (b), respectively. These values are typical of most properly sampled television images.

These illustrations reflect another important form of data redundancy—one directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonably predicted from the value of its neighbors, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of the values of its neighbors. A variety of names, including spatial redundancy, geometric redundancy, and interframe redundancy, have been coined to refer to these interpixel dependencies. We use the term interpixel redundancy to encompass them all.

In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient (but usually "nonvisual") format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type (that is, those that remove interpixel redundancy) are referred to as mappings. They are called reversible

mappings if the original image elements can be reconstructed from the transformed data set.

Psychovisual Redundancy:

The brightness of a region, as perceived by the eye, depends on factors other than simply the light reflected by the region. For example, intensity variations (Mach bands) can be perceived in an area of constant intensity. Such phenomena result from the fact that the eye does not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be psychovisually redundant. It can be eliminated without significantly impairing the quality of image perception.

That psychovisual redundancies exist should not come as a surprise, because human perception of the information in an image normally does not involve quantitative analysis of every pixel value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process. Psychovisual redundancy is fundamentally different from the redundancies discussed earlier. Unlike coding and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisually redundant data results in a loss of quantitative information, it is commonly referred to as quantization.

This terminology is consistent with normal usage of the word, which generally means the mapping of a broad range of input values to a limited number of output values. As it is an irreversible operation (visual information is lost), quantization results in lossy data compression.

Fidelity Criterion.

The removal of psycho visually redundant data results in a loss of real or quantitative visual information. Because information of interest may be lost, a repeatable or reproducible means of quantifying the nature and extent of information loss is highly desirable. Two general classes of criteria are used as the basis for such an assessment:

A) Objective fidelity criteria and

B) Subjective fidelity criteria.

When the level of information loss can be expressed as a function of the original or

input image and the compressed and subsequently decompressed output image, it is said to be based on an objective fidelity criterion. A good example is the root-mean-square (rms) error between an input and output image. Let f(x, y) represent an input image and let f(x, y) denote an estimate or approximation of f(x, y) that results from compressing and subsequently decompressing the input. For any value of x and y, the error e(x, y) between f (x, y) and f^ (x, y) can be defined as

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

so that the total error between the two images is

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]$$

where the images are of size M X N. The root-mean-square error, e_{rms}, between f(x, y) and f^(x, y) then is the square root of the squared error averaged over the M X N array, or

$$e_{rms} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]^2\right]^{1/2}$$

A closely related objective fidelity criterion is the mean-square signal-to-noise ratio of the compressed-decompressed image. If f^ (x, y) is considered to be the sum of the original image f(x, y) and a noise signal e(x, y), the mean-square signal-to-noise ratio of the output image, denoted SNR_{rms}, is

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x, y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]^2}.$$

The rms value of the signal-to-noise ratio, denoted SNR_{rms}, is obtained by taking the square root of Eq. above.

Although objective fidelity criteria offer a simple and convenient mechanism for evaluating information loss, most decompressed images ultimately are viewed by humans. Consequently, measuring image quality by the subjective evaluations of a human observer often is more appropriate. This can be accomplished by showing a "typical" decompressed image to an appropriate cross section of viewers and averaging their evaluations. The evaluations may be made using an absolute rating scale or by means of side-by-side comparisons of f(x, y) and f^(x, y).

Image Compression models

A compression system consists of two distinct structural blocks: an encoder and a decoder. An input image f(x, y) is fed into the encoder, which creates a set of symbols from the input data. After transmission over the channel, the encoded representation is fed to the decoder, where a reconstructed output image f^(x, y) is generated. In general, f^(x, y) may or may not be an exact replica of f(x, y). If it is, the system is error free or information preserving; if not, some level of distortion is present in the reconstructed image. Both the encoder and decoder shown in Fig. 3.1 consist of two relatively independent functions or subblocks. The encoder is made up of a source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. As would be expected, the decoder includes a channel decoder followed by a source decoder. If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

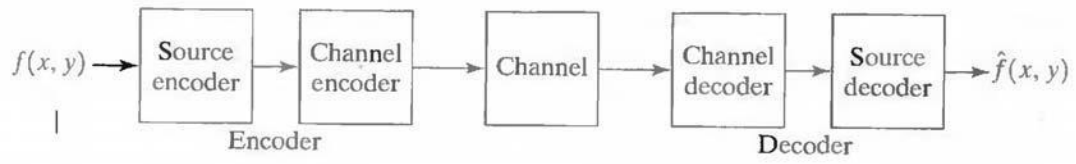
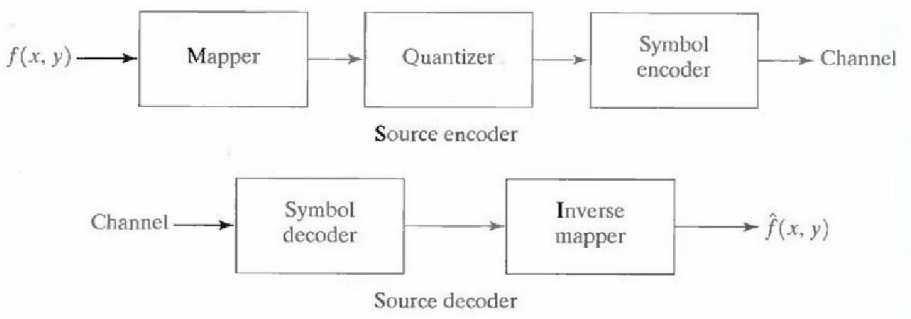


Fig.3.1 A general compression system model

The Source Encoder and Decoder:

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application and associated fidelity requirements dictate the best encoding approach to use in any given situation. Normally, the approach can be modeled by a series of three independent operations.



a
b

Digit

Fig.3.2 (a) Source encoder and (b) source decoder model

As Fig. 3.2 (a) shows, each operation is designed to reduce one of the three redundancies. Figure 3.2 (b) depicts the corresponding source decoder. In the first stage of the source encoding process, the mapper transforms the input data into a (usually nonvisual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.

Run-length coding is an example of a mapping that directly results in data compression in this initial stage of the overall source encoding process. The representation of an image by a set of transform coefficients is an example of the opposite case. Here, the mapper transforms the image into an array of coefficients, making its interpixel redundancies more accessible for compression in later stages of the encoding process.

The second stage, or quantizer block in Fig. 3.2 (a), reduces the accuracy of the mapper's output in accordance with some preestablished fidelity criterion. This stage reduces the psychovisual redundancies of the input image. This operation is irreversible. Thus it must be omitted when error-free compression is desired.

In the third and final stage of the source encoding process, the symbol coder creates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding process. In most cases, a variable-length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most frequently occurring output values and thus reduces coding redundancy. The operation, of course, is reversible. Upon completion of the symbol coding step, the input image has been processed to remove each of the three redundancies.

Figure 3.2(a) shows the source encoding process as three successive operations, but all three operations are not necessarily included in every compression system. Recall, for example, that the quantizer must be omitted when error-free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in Fig. 3.2(a). In the predictive compression systems, for instance, the mapper and quantizer are often represented by a single block, which simultaneously performs both operations.

The source decoder shown in Fig. 3.2(b) contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in Fig. 3.2(b).

The Channel Encoder and Decoder:

The channel encoder and decoder play an important role in the overall encoding-decoding process when the channel of Fig. 3.1 is noisy or prone to error. They are designed to reduce the impact of channel noise by inserting a controlled form of redundancy into the source encoded data. As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of this "controlled redundancy." One of the most useful channel encoding techniques was devised by R. W. Hamming (Hamming [1950]). It is based on appending enough bits to the data being encoded to ensure that some minimum number of bits must change between valid code words. Hamming showed, for example, that if 3 bits of redundancy are added to a 4-bit word, so that the distance between any two valid code words is 3, all single-bit errors can be detected and corrected. (By appending additional bits of redundancy, multiple-bit errors can be detected and corrected.) The 7-bit Hamming (7, 4) code word $h_1, h_2, h_3 \ldots, h_6, h_7$ associated with a 4-bit binary number $b_3b_2b_1b_0$ is

$$
\begin{aligned}
h_1 &= b_3 \oplus b_2 \oplus b_0 \qquad & h_3 &= b_3 \\
h_2 &= b_3 \oplus b_1 \oplus b_0 \qquad & h_5 &= b_2 \\
h_4 &= b_2 \oplus b_1 \oplus b_0 \qquad & h_6 &= b_1 \\
& & h_7 &= b_0
\end{aligned}
$$

where denotes the exclusive OR operation. Note that bits h_1, h_2, and h_4 are even-parity bits for the bit fields $b_3\ b_2\ b_0$, $b_3b_1b_0$, and $b_2b_1b_0$, respectively. (Recall that a string of binary bits has even parity if the number of bits with a value of 1 is even.) To decode a Hamming encoded result, the channel decoder must check the encoded value for odd parity over the bit fields in which even parity was previously established. A single-bit error is indicated by a nonzero parity word $c_4c_2c_1$, where

$$
\begin{aligned}
c_1 &= h_1 \oplus h_3 \oplus h_5 \oplus h_7 \\
c_2 &= h_2 \oplus h_3 \oplus h_6 \oplus h_7 \\
c_4 &= h_4 \oplus h_5 \oplus h_6 \oplus h_7.
\end{aligned}
$$

If a nonzero value is found, the decoder simply complements the code word bit position indicated by the parity word. The decoded binary value is then extracted from the corrected code word as $h_3h_5h_6h_7$.

Variable-Length Coding:

The simplest approach to error-free image compression is to reduce only coding redundancy. Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. It can be eliminated by coding the gray levels. To do so requires construction of a variable-length code that assigns the shortest possible code words to the most probable gray levels. Here, we examine several optimal and near optimal techniques for constructing such a code. These techniques are formulated in the language of information theory. In practice, the source symbols may be either the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run lengths, and so on).

Huffman coding:

The most popular technique for removing coding redundancy is due to Huffman (Huffman [1952]). When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol. In terms of the noiseless coding theorem, the resulting code is optimal for a fixed value of n, subject to the constraint that the source symbols be coded one at a time.

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 4.1 illustrates this process for binary coding (K-ary Huffman codes can also be constructed). At the far left, a hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities, 0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the

probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As Fig. 4.2 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order

of the 0 and 1 would work just as well). As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily

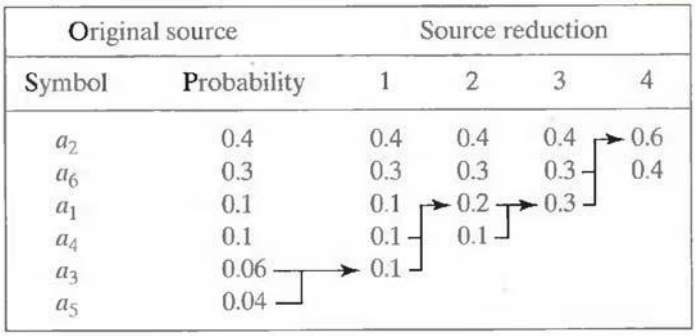


Fig.4.1 Huffman source reductions.

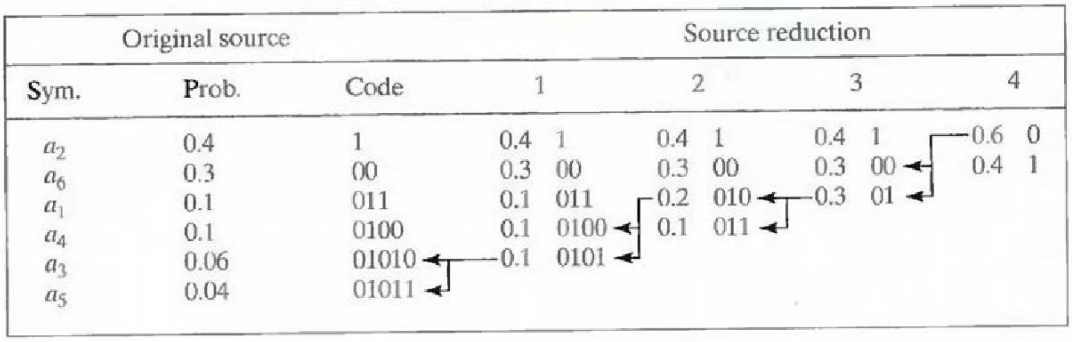


Fig.4.2 Huffman code assignment procedure.

appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far left in Fig. 4.2. The average length of this code is

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/symbol}$$

and the entropy of the source is 2.14 bits/symbol. The resulting Huffman code efficiency is 0.973.

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple lookup table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous,

because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of Fig. 4.2, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a_3 .The next valid code is 011, which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3a_1a_2a_2a_6$.

Arithmetic coding:

Unlike the variable-length codes described previously, arithmetic coding generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias, a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound established by the noiseless coding theorem.

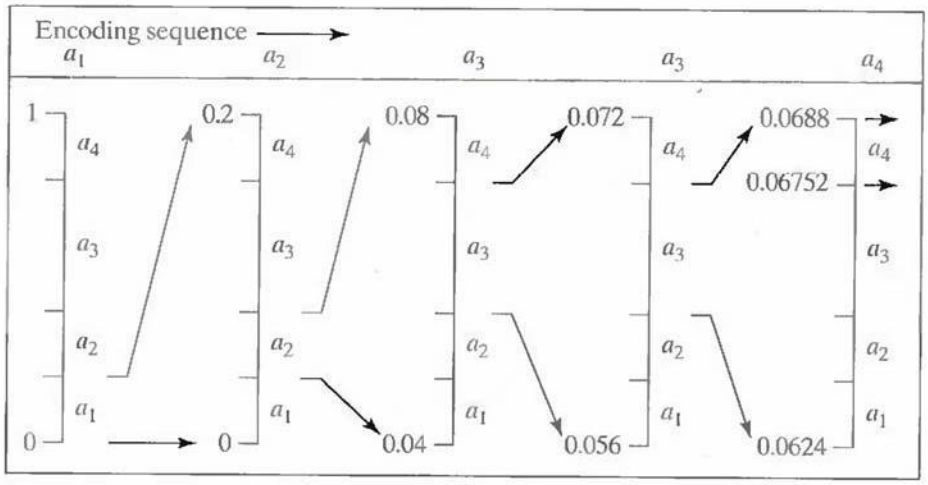


Fig.5.1 Arithmetic coding procedure

Figure 5.1 illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, $a_1a_2a_3a_3a_4$, from a four-symbol source is coded. At the start of the

coding process, the message is assumed to occupy the entire half-open interval [0, 1). As Table 5.2 shows, this interval is initially subdivided into four regions based on the probabilities of each source symbol. Symbol ax, for example, is associated with subinterval [0, 0.2). Because it is the first symbol of the message being coded, the message interval is initially narrowed to [0, 0.2). Thus in Fig. 5.1 [0, 0.2) is expanded to the full height of the figure and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original source symbol probabilities and the process continues with the next message symbol.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

Table 5.1 Arithmetic coding example

In this manner, symbol a_2 narrows the subinterval to [0.04, 0.08), a_3 further narrows it to [0.056, 0.072), and so on. The final message symbol, which must be reserved as a special end-of- message indicator, narrows the range to [0.06752, 0.0688). Of course, any number within this subinterval—for example, 0.068—can be used to represent the message.

In the arithmetically coded message of Fig. 5.1, three decimal digits are used to represent the five-symbol message. This translates into 3/5 or 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which is 0.58 decimal digits or 10-ary units/symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by the noiseless coding theorem.

In practice, two factors cause coding performance to fall short of the bound: (1) the addition of the end-of-message indicator that is needed to separate one message from another; and (2) the use of finite precision arithmetic. Practical implementations of arithmetic coding address the latter problem by introducing a scaling strategy and a rounding strategy (Langdon and Rissanen [1981]). The scaling strategy renormalizes each subinterval to the [0, 1) range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

LZW Coding:

The technique, called Lempel-Ziv-Welch (LZW) coding, assigns fixed-length code words to variable length sequences of source symbols but requires no a priori knowledge of

the probability of occurrence of the symbols to be encoded. LZW compression has been integrated into a variety of mainstream imaging file formats, including the graphic interchange format (GIF), tagged image file format (TIFF), and the portable document format (PDF). LZW coding is conceptually very simple (Welch [1984]). At the onset of the coding process, a codebook or "dictionary" containing the source symbols to be coded is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to the gray values 0, 1, 2..., and 255. As the encoder sequentially examines the image's pixels, gray-level sequences that are not in the dictionary are placed in algorithmically determined (e.g., the next unused) locations. If the first two pixels of the image are white, for instance, sequence "255-255" might be assigned to location 256, the address following the locations reserved for gray levels 0 through 255. The next time that two consecutive white pixels are encountered, code word 256, the address of the location containing sequence 255-255, is used to represent them. If a 9-bit, 512-word dictionary is employed in the coding process, the original (8 + 8) bits that were used to represent the two pixels are replaced by a single 9-bit code word. Cleary, the size of the dictionary is an important system parameter. If it is too small, the detection of matching gray-level sequences will be less likely; if it is too large, the size of the code words will adversely affect compression performance.

Consider the following 4 x 4, 8-bit image of a vertical edge:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Table 6.1 details the steps involved in coding its 16 pixels. A 512-word dictionary with the following starting content is assumed:

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Locations 256 through 511 are initially unused. The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner. Each successive gray-level value is

concatenated with a variable—column 1 of Table 6.1 —called the "currently recognized sequence." As can be seen, this variable is initially null or empty. The dictionary is searched for each concatenated sequence and if found, as was the case in the first row of the table, is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. This was done in column 1 of row 2.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Table 6.1 LZW coding example

No output codes are generated, nor is the dictionary altered. If the concatenated sequence is not found, however, the address of the currently recognized sequence is output as the next encoded value, the concatenated but unrecognized sequence is added to the dictionary, and the currently recognized sequence is initialized to the current pixel value. This occurred in row 2 of the table. The last two columns detail the gray-level sequences that are added to the dictionary when scanning the entire 4 x 4 image. Nine additional code words are defined. At the conclusion of coding, the dictionary contains 265 code words and the LZW algorithm has successfully identified several repeating gray-level sequences—leveraging them to reduce the original 128-bit image lo 90 bits (i.e., 10 9-bit codes). The encoded output is obtained by reading the third column from top to bottom. The resulting compression ratio is 1.42:1.

A unique feature of the LZW coding just demonstrated is that the coding dictionary or code book is created while the data are being encoded. Remarkably, an LZW decoder builds

an identical decompression dictionary as it decodes simultaneously the encoded data stream. . Although not needed in this example, most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is to monitor compression performance and flush the dictionary when it becomes poor or unacceptable. Alternately, the least used dictionary entries can be tracked and replaced when necessary.

Bit-Plane Coding:

An effective technique for reducing an image's interpixel redundancies is to process the image's bit planes individually. The technique, called bit-plane coding, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several well-known binary compression methods.

Bit-plane decomposition:

The gray levels of an m-bit gray-scale image can be represented in the form of the base 2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0.$$

Based on this property, a simple method of decomposing the image into a collection of binary images is to separate the m coefficients of the polynomial into m 1-bit bit planes. The zeroth-order bit plane is generated by collecting the a_0 bits of each pixel, while the (m - 1) st-order bit plane contains the a_{m-1}, bits or coefficients. In general, each bit plane is numbered from 0 to m-1 and is constructed by setting its pixels equal to the values of the appropriate bits or polynomial coefficients from each pixel in the original image. The inherent disadvantage of this approach is that small changes in gray level can have a significant impact on the complexity of the bit planes. If a pixel of intensity 127 (01111111) is adjacent to a pixel of intensity 128 (10000000), for instance, every bit plane will contain a corresponding 0 to 1 (or 1 to 0) transition. For example, as the most significant bits of the two binary codes for 127 and 128 are different, bit plane 7 will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point.

An alternative decomposition approach (which reduces the effect of small gray-level variations) is to first represent the image by an m-bit Gray code. The m-bit Gray code $g_{m-1}...g_2g_1g_0$ that corresponds to the polynomial in Eq. above can be computed from

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$
$$g_{m-1} = a_{m-1}.$$

Here, denotes the exclusive OR operation. This code has the unique property that successive code words differ in only one bit position. Thus, small changes in gray level are less likely to affect all m bit planes. For instance, when gray levels 127 and 128 are adjacent, only the 7th bit plane will contain a 0 to 1 transition, because the Gray codes that correspond to 127 and 128 are 11000000 and 01000000, respectively

Lossless Predictive Coding:

The error-free compression approach does not require decomposition of an image into a collection of bit planes. The approach, commonly referred to as lossless predictive coding, is based on eliminating the interpixel redundancies of closely spaced pixels by extracting and coding only the new information in each pixel. The new information of a pixel is defined as the difference between the actual and predicted value of that pixel.

Figure 8.1 shows the basic components of a lossless predictive coding system. The system consists of an encoder and a decoder, each containing an identical predictor. As each successive pixel of the input image, denoted f_n, is introduced to the encoder, the predictor generates the anticipated value of that pixel based on some number of past inputs. The output of the predictor is then rounded to the nearest integer, denoted $\hat{f}_n$ and used to form the difference or prediction error which is coded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream.

$$e_n = f_n - \hat{f}_n,$$

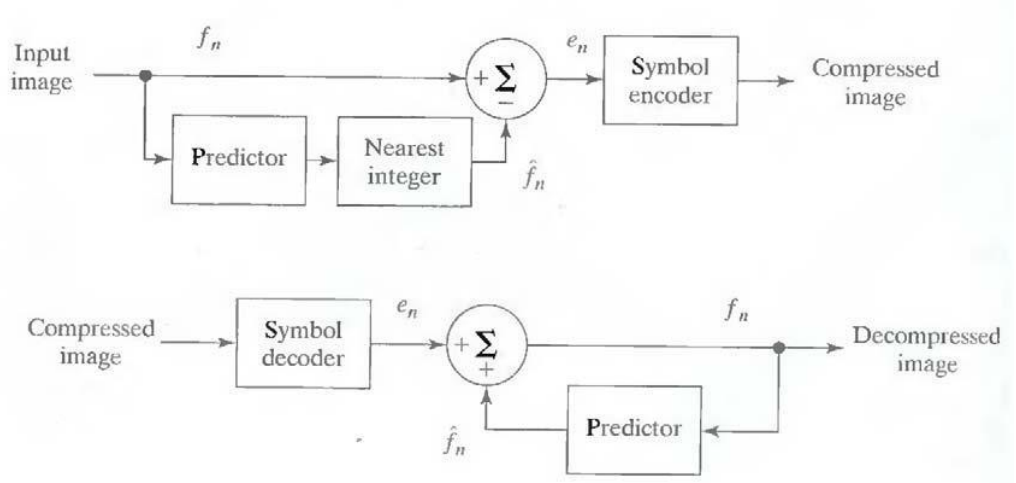


Fig.8.1 A lossless predictive coding model: (a) encoder; (b) decoder

The decoder of Fig. 8.1 (b) reconstructs en from the received variable-length code words and performs the inverse operation

$$f_n = e_n + \hat{f}_n.$$

Various local, global, and adaptive methods can be used to generate f^$_n$. In most cases, however, the prediction is formed by a linear combination of m previous pixels. That is,

$$\hat{f}_n = \text{round}\left[\sum_{i=1}^{m} \alpha_i f_{n-i}\right]$$

where m is the order of the linear predictor, round is a function used to denote the rounding or nearest integer operation, and the α_i, for i = 1,2,..., m are prediction coefficients. In raster scan applications, the subscript n indexes the predictor outputs in accordance with their time of occurrence. That is, f_n, f^$_n$ and e_n in Eqns. above could be replaced with the more explicit notation f (t), f^(t), and e (t), where t represents time. In other cases, n is used as an index on the spatial coordinates and/or frame number (in a time sequence of images) of an image. In 1-D linear predictive coding, for example, Eq. above can be written as

$$\hat{f}_n(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right]$$

where each subscripted variable is now expressed explicitly as a function of spatial coordinates x and y. The Eq. indicates that the 1-D linear prediction f(x, y) is a function of the previous pixels on the current line alone. In 2-D predictive coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image. In the 3-D case, it is based on these pixels and the previous pixels of preceding frames. Equation above cannot be evaluated for the first m pixels of each line, so these pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. A similar comment applies to the higher-dimensional cases

Lossy Predictive Coding:

In this type of coding, we add a quantizer to the lossless predictive model and examine the resulting trade-off between reconstruction accuracy and compression performance. As Fig.9 shows, the quantizer, which absorbs the nearest integer function of the error-free encoder, is inserted between the symbol encoder and the point at which the prediction error is formed. It maps the prediction error into a limited range of outputs, denoted e^$_n$ which establish the amount of compression and distortion associated with lossy

predictive coding.

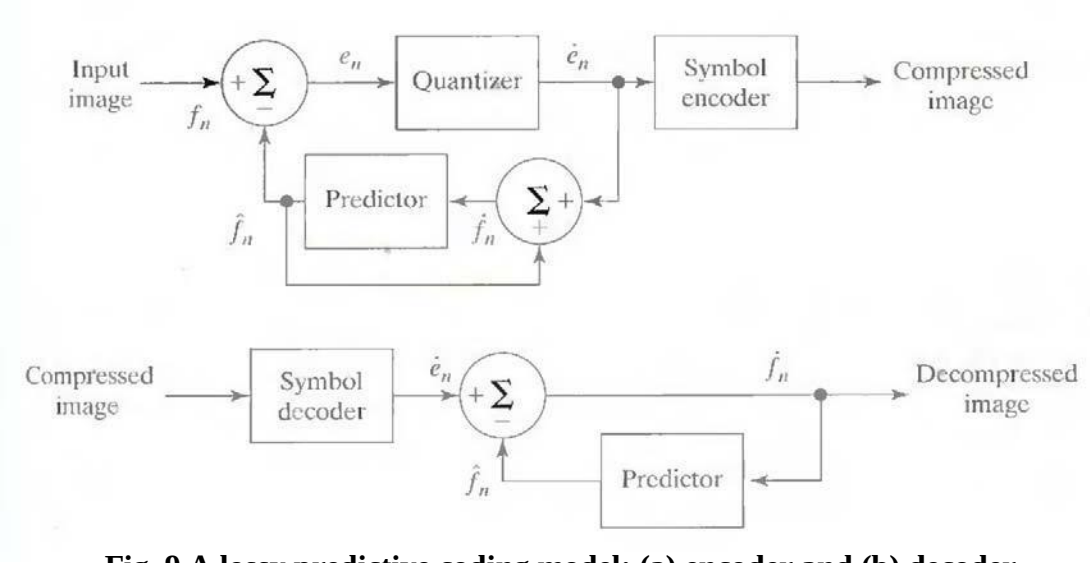


Fig. 9 A lossy predictive coding model: (a) encoder and (b) decoder

In order to accommodate the insertion of the quantization step, the error-free encoder of figure must be altered so that the predictions generated by the encoder and decoder are equivalent. As Fig.9 (a) shows, this is accomplished by placing the lossy encoder's predictor within a feedback loop, where its input, denoted $\dot{f}_n$, is generated as a function of past predictions and the corresponding quantized errors. That is,

$$\dot{f}_n = \dot{e}_n + \hat{f}_n$$

This closed loop configuration prevents error buildup at the decoder's output. Note from Fig. 9(b) that the output of the decoder also is given by the above Eqn.

Optimal predictors:

The optimal predictor used in most predictive coding applications minimizes the encoder's mean-square prediction error

$$E\{e_n^2\} = E\{[f_n - \hat{f}_n]^2\}$$

subject to the constraint that

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \approx e_n + \hat{f}_n = f_n$$

and

$$\hat{f}_n = \sum_{i=1}^{m} \alpha_i f_{n-i}.$$

That is, the optimization criterion is chosen to minimize the mean-square prediction

error, the quantization error is assumed to be negligible ($e^{\cdot}_n \approx e_n$), and the prediction is constrained to a linear combination of m previous pixels.1 These restrictions are not essential, but they simplify the analysis considerably and, at the same time, decrease the computational complexity of the predictor. The resulting predictive coding approach is referred to as differential pulse code modulation (DPCM).

Transform Coding:

All the predictive coding techniques operate directly on the pixels of an image and thus are spatial domain methods. In this coding, we consider compression techniques that are based on modifying the transform of an image. In transform coding, a reversible, linear transform (such as the Fourier transform) is used to map the image into a set of transform coefficients, which are then quantized and coded. For most natural images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion. A variety of transformations, including the discrete Fourier transform (DFT), can be used to transform the image data.

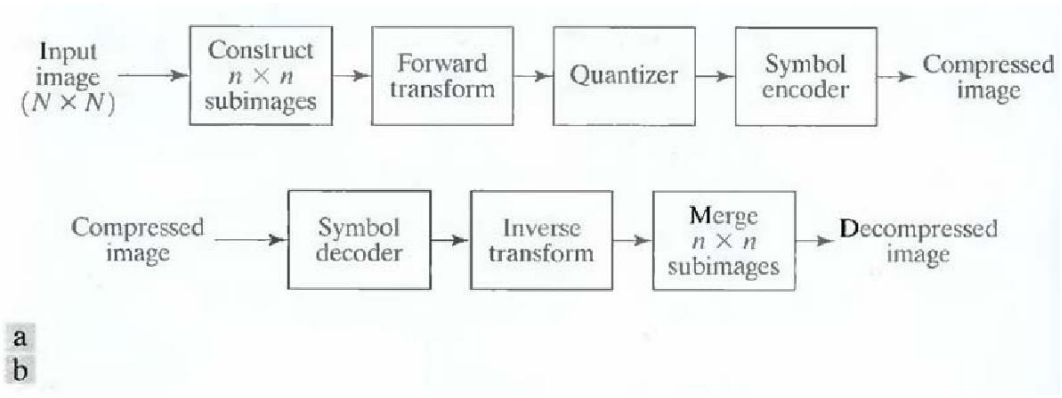


Fig. 10 A transform coding system: (a) encoder; (b) decoder.

Figure 10 shows a typical transform coding system. The decoder implements the inverse sequence of steps (with the exception of the quantization function) of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. An N X N input image first is subdivided into subimages of size n X n, which are then transformed to generate $(N/n)^2$ subimage transform arrays, each of size n X n. The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least information. These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by

coding (normally using a variable-length code) the quantized coefficients. Any or all of the transform encoding steps can be adapted to local image content, called adaptive transform coding, or fixed for all subimages, called nonadaptive transform coding.

Wavelet Coding:

The wavelet coding is based on the idea that the coefficients of a transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves. If the transform's basis functions—in this case wavelets—pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.

Figure 11 shows a typical wavelet coding system. To encode a 2^J X 2^J image, an analyzing wavelet, Ψ, and minimum decomposition level, J - P, are selected and used to compute the image's discrete wavelet transform. If the wavelet has a complimentary scaling function φ, the fast wavelet transform can be used. In either case, the computed transform converts a large portion of the original image to horizontal, vertical, and diagonal decomposition coefficients with zero mean and Laplacian-like distributions.

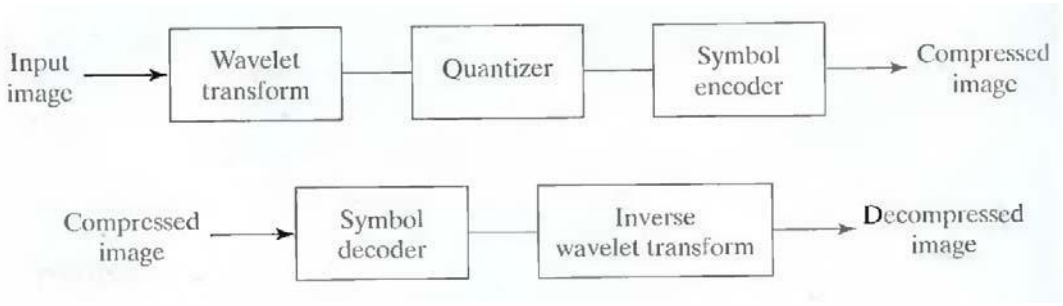


Fig.11 A wavelet coding system: (a) encoder; (b) decoder.

Since many of the computed coefficients carry little visual information, they can be quantized and coded to minimize intercoefficient and coding redundancy. Moreover, the quantization can be adapted to exploit any positional correlation across the P decomposition levels. One or more of the lossless coding methods, including run-length, Huffman, arithmetic, and bit-plane coding, can be incorporated into the final symbol coding step. Decoding is accomplished by inverting the encoding operations—with the exception of quantization, which cannot be reversed exactly.

The principal difference between the wavelet-based system and the transform coding system is the omission of the transform coder's subimage processing stages.

Because wavelet transforms are both computationally efficient and inherently local

(i.e., their basis functions are limited in duration), subdivision of the original image is unnecessary

PREVIOUS QUESTIONS

1. Draw the functional block diagram of image compression system and explain the purpose of each block.
2. Explain the need for image compression. How run length encoding approach is used for compression? Is it lossy? Justify.
3. Describe about wavelet packets.
4. Write short notes on: i) Arithmetic coding. ii) Vector quantization. iii) JPEG standards.
5. Explain about the Fast Wavelet Transform.
6. Explain two-band sub-band coding and decoding system.
7. What are the various requirements for multi-resolution analysis? Explain.
8. What is block transform coding? Explain.
9. With an example, explain Huffman coding.
10. Write about Haar Wavelet transform.
11. What is meant by redundancy in image? Explain its role in image processing.
12. Compute the Haar transform of the 2 x 2 image F= $\begin{bmatrix} 3 & -1 \\ 6 & 2 \end{bmatrix}$