Module 5

IMAGE COMPRESSION

Image compression is the art and science of reducing amount of data required to represent an image. Image compression is used in many applications like televideo conferencing, remote sensing, document and medical imaging, and facsimile transmission (FAX).

Fundamentals:

The data compression refers to the process of reducing the amount of data required to represent a given quantity of information. Here data and information are not the same, data are the mean by which information is conveyed and various amounts of data can be used to represent the same amount of information. The data can have irrelevant or repeated information are called redundant data.

If we let b and b' denote the number of bits in two representations of the same information, the relative data redundancy R of the representation with b bits is

$$R = 1 - 1/C$$

Where C, commonly called the compression ratio, is defined as

$$C = b/b'$$

The principle types of data redundancy that can be identified as:

1. Coding redundancy: A code is a system of symbols used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of code symbols, called a code word. The number of symbols in each code word is its

length. The 8-bit codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.

2. Spatial and temporal redundancy: Because the pixels of most 2-D intensity arrays are correlated specially,(i.e. each pixel is similar to or dependent on neighboring pixels Information is unnecessarily replicated in the representations of the correlated pixels. In a video sequence, temporally correlated pixels also duplicate information
3. Irrelevant information: Most 2-D intensity arrays contain information that is ignored by human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used.

Coding Redundancy

Let us assume, that a discrete random variable *rk* in the interval [0,L-1] represent the gray level of an M x N image and that each r_k occurs with probabilities $p_r(r_k)$:

$$\mathbf{P_r (r_k) = n_k / MN \quad k=0,1,2,3\ldots L\text{-}1.}$$

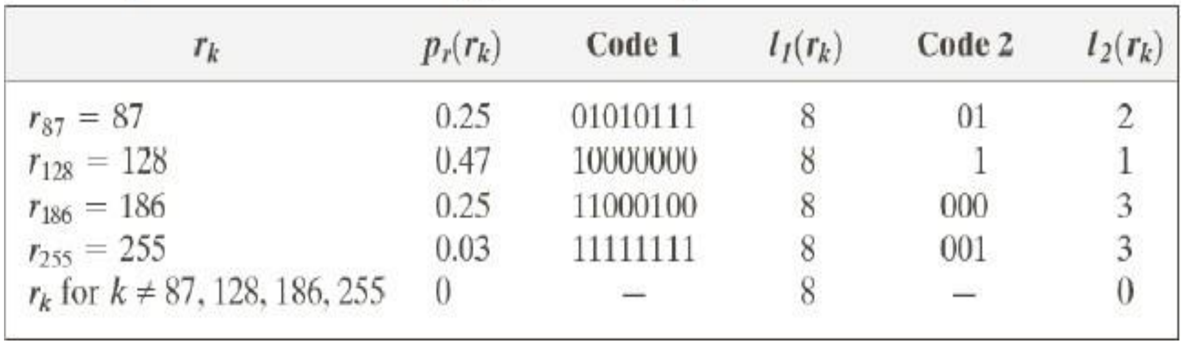
Where L is the number of intensity values, and n_k is the number of times that the k^{th} intensity appears in the image. If the number of bits used to represent each value of *rk* is *l(rk)*, then the average number of bits required to represent each pixel:

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

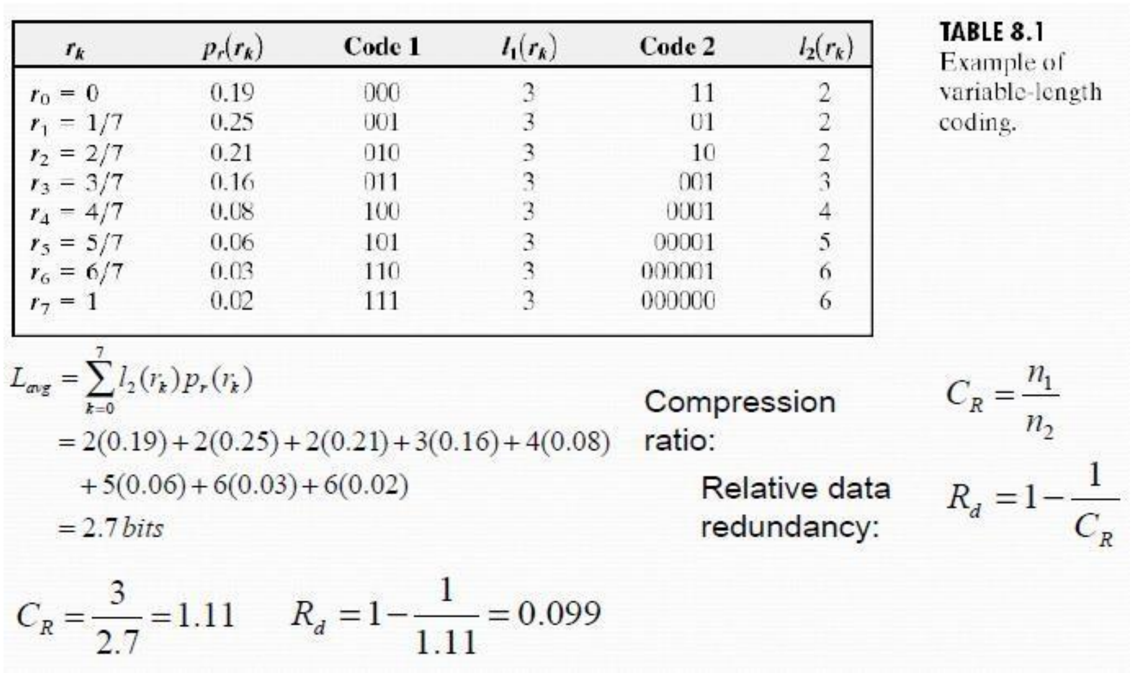
The total number bits required to code an *MxN* image:

$$M.N.L_{avg}$$

Examples of variable length encoding



r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	–	8	–	0



r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_0 = 0$	0.19	000	3	11	2
$r_1 = 1/7$	0.25	001	3	01	2
$r_2 = 2/7$	0.21	010	3	10	2
$r_3 = 3/7$	0.16	011	3	001	3
$r_4 = 4/7$	0.08	100	3	0001	4
$r_5 = 5/7$	0.06	101	3	00001	5
$r_6 = 6/7$	0.03	110	3	000001	6
$r_7 = 1$	0.02	111	3	000000	6

TABLE 8.1 Example of variable-length coding.

$$L_{avg} = \sum_{k=0}^{7} l_2(r_k)p_r(r_k)$$
$$= 2(0.19)+2(0.25)+2(0.21)+3(0.16)+4(0.08)$$
$$+5(0.06)+6(0.03)+6(0.02)$$
$$= 2.7\,bits$$

Compression ratio: $C_R = \frac{n_1}{n_2}$

Relative data redundancy: $R_d = 1 - \frac{1}{C_R}$

$$C_R = \frac{3}{2.7} = 1.11 \qquad R_d = 1 - \frac{1}{1.11} = 0.099$$

Interpixel redundancy

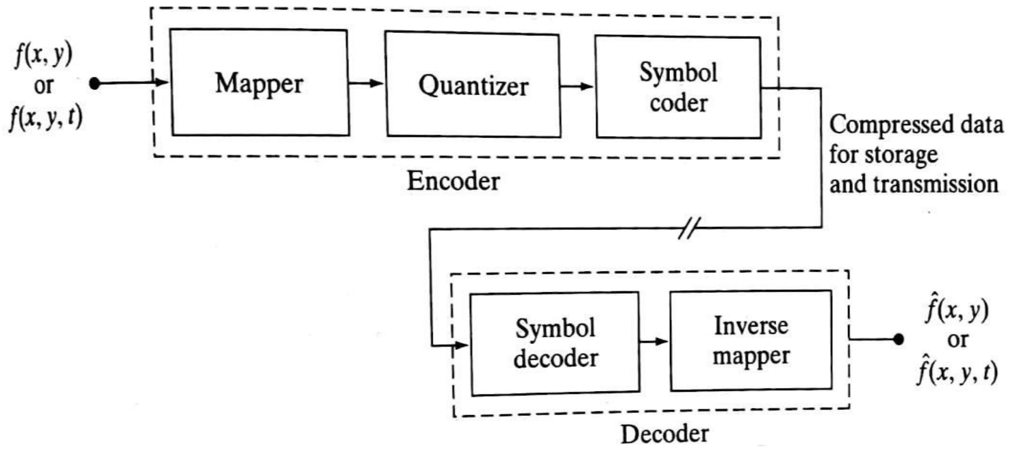
Interpixel redundancy is defined as failure to identify and utilize data relationships

If a pixel value can be reasonably predicted from its neighboring (or preceeding / following) pixels the image is said to contain interpixel redundancy. Interpixel redundancy depends on the resolution of the image The higher the (spatial) resolution of an image, the more probable it is that two neighboring pixels will depict the same object. The higher the frame rate in a video stream, the more probable it is that the corresponding pixel in the following frame will depict the same object these types of predictions are made more difficult by the presence of noise

Image Compression Model

The following figure shows an image compression system is composed of two distinct functional components: an encoder and decoder. The encoder performs compression and decoder performs the complementary operation of decompression. Both operations can be performed in software.

Input image is fed into the encoder, which creates a compressed representation of the input. When the compressed representation is presented to its complementary decoder, a reconstructed image is generated.



$f(x, y)$ or $f(x, y, t)$
Mapper
Quantizer
Symbol coder
Encoder
Compressed data for storage and transmission
Symbol decoder
Inverse mapper
Decoder
$\hat{f}(x, y)$ or $\hat{f}(x, y, t)$

FIGURE 8.5 Functional block diagram of a general image compression system.

- **Mapper** - transforms the image to a (non-visual) format designed to reduce interpixel redundancies. The operation generally is reversible and may or may not reduce directly the amount of data required to represent the image. In video applications, the mapper uses previous video frames to facilitate the removal of temporal redundancy.
- **Quantizer** - reduces psychovisual redundancies by quantizing data deemed less important for visual interpretation (omitted for lossless compression)
- **Symbol encoder** - codes the data efficiently (typically using some form of variable-length coding scheme) and aims to reduce coding redundancies. It generates a fixed or variable length code to represent the quantizer output and maps the output in accordance with the code. In many cases, a variable –length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values- thus minimizing coding redundancy.

  • Future application requirements may be unknown

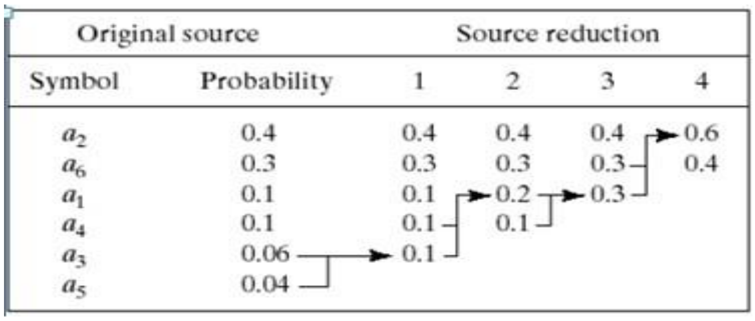
Some Basic Compression Methods

Huffman coding

Provides a data representation with the smallest possible number of code symbols per source symbol (when coding the symbols of an information source individually)

Huffman code construction is done in two steps

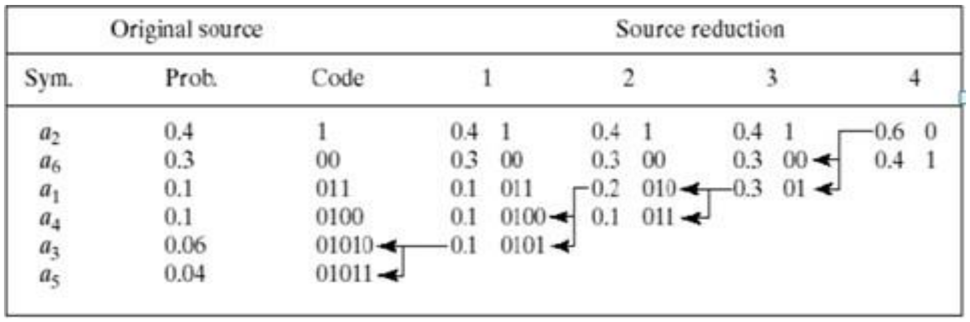
1. Source reductions: Create a series of source reductions by ordering the probabilities of the symbols and then combine the symbols of lowest probability to form new symbols recursively. This step is sometimes referred to as "building a Huffman tree" and can be done statistically



Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Figure: Huffman source reduction

2. Code assignment: When only two symbols remain, retrace the steps in 1) and assign a code bit to the symbols in each step. Encoding and decoding is done using these codes in a lookup table manner



Original source			Source reduction							
Sym.	Prob.	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Figure: Huffman code assignment procedure

The average length of this code is: $L_{avg} = (0.4)(1) + (0.3)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$
$=2.2$bits/pixels

Arithmetic coding

Provides a data representation where the entire symbol sequence is encoded as a single arithmetic code word (which is represented as an interval of real numbers between 0 and 1)

Arithmetic code construction is done in 3 steps

1. Subdivide the half-open interval [0,1) based on the probabilities of the source symbols

2. For each source symbol recursively. Narrow the interval to the sub-interval designated by the encoded symbol. Subdivide the new interval among the source symbols based on probability
3. Append an end-of-message indicator. Encoding is done by choosing any number in the interval to represent the data. Decoding is done by retracing the steps in the code construction

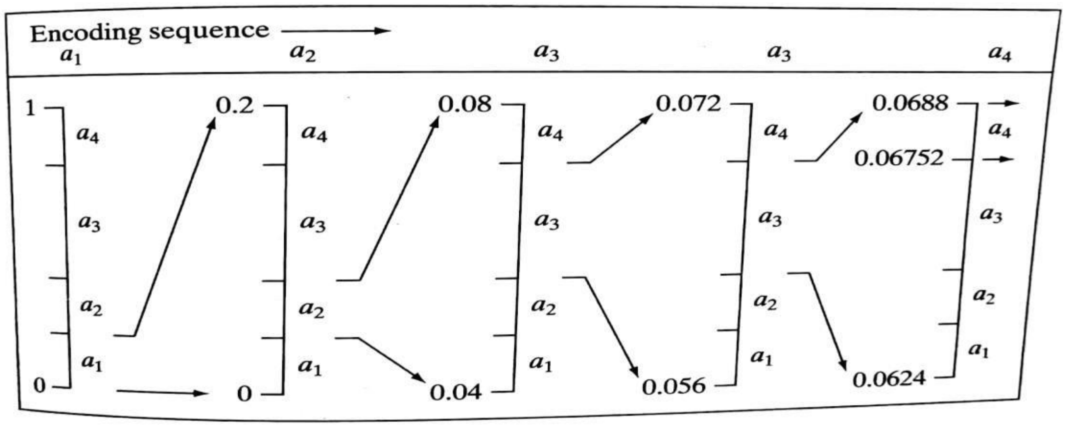
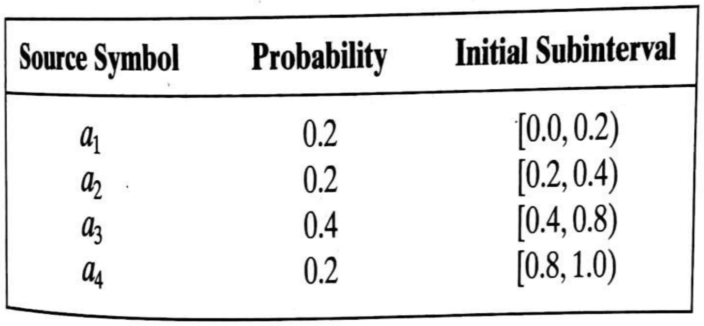


Figure : Arithmetic coding procedure.

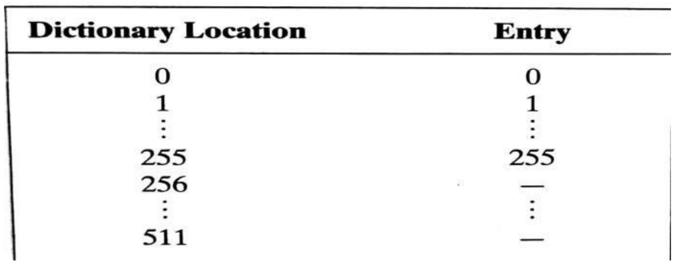


Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

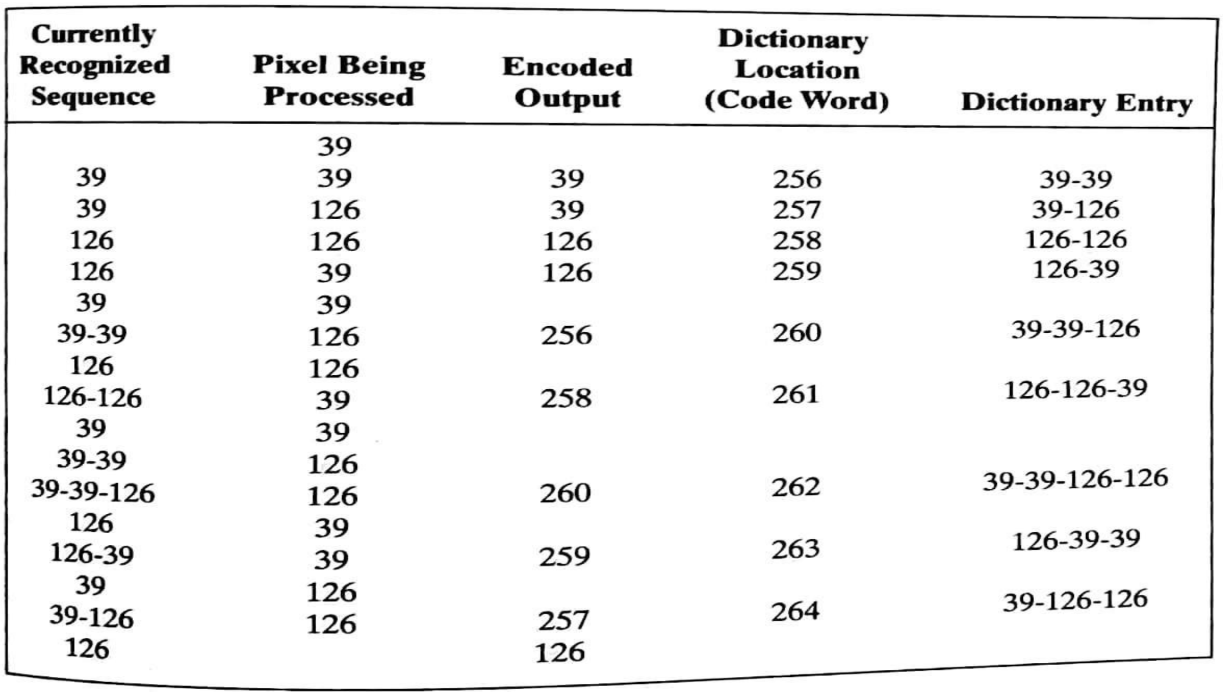
Figure : Arithmetic coding example.

Lempel-Ziv-Welch (LZW) coding

Provides a data representation where fixed-length code words are assigned to variable length sequences of source symbols. LZW reduces both coding redundancies as well as interpixel redundancies and requires no knowledge of the source symbol probabilities. LZW code construction is done by constructing a dictionary which translates encountered source symbol sequences to code symbols. This dictionary is created as the data stream is being processed and can be flushed (reinitialized) when it is full or when coding ratios decline. The size of the dictionary affects compression performance. Small dictionaries are less likely to contain an encountered source symbol. Large dictionaries results in larger code words (lower compression ratios)



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



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		

Figure : LZW coing example

Run-length coding

Provides a data representation where sequences of symbol values of the same value are coded as "run-lengths"

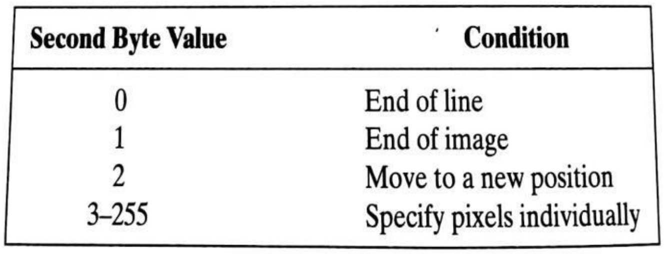
- 1D – the image is treated as a row-by-row array of pixels
- 2D – blocks of pixels are viewed as source symbols

Input data 22222222222266666666666666699999999999999999

Code 2,11 6,14 9,17

Here compression is achieved by eliminating a simple form of spatial redundancy – groups of identical intensities. When there are few runs of identical pixels, run length encoding results in data expansion. The BMP file format uses a form of rum-length encoding In which image data is represented in two different modes :

1. Encoded mode – A two byte RLE representation is used. The first byte specifies the number of consecutive pixels that have the color index contained in the second byte. The 8 bit color index selects the run's intensity from a table of 256 possible intensities.
2. Absolute mode – The first byte here is 0 and the second byte signals one of four possible conditions, as shown in below table.



Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually

Figure : BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0

Transform Coding

In transform coding a linear transform is used to map the image data into a set of transform coefficients (which are then quantized and coded). For reasons of computational complexity the image is subdivided into smaller subimages before the transformation computation. The goal of the transformation process is to decorrelate (minimize the variance) the pixels of the subimages as much as possible. This will result in a localization of the image information in a minimal number of transform coefficients - the transformation coefficients with little data can then be more coarsely quantized or eliminated. The better information compaction, the better reconstruction approximations.

The following figure shows a typical block transform coding system. The decoder implements the inverse sequence of steps of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. M x N input image is subdivided first into subimages of size n X n, which are then transformed to generate MN/n^2 subimage transform arrays, each of size n X n. The quatization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least amount of information in a predefined sense. The encoding process terminates by coding the quantized coefficients. Transformation coding can be

- Adaptive – each subimage is treated in an individual manner
- Non-adaptive – all subimages are treated in the same way

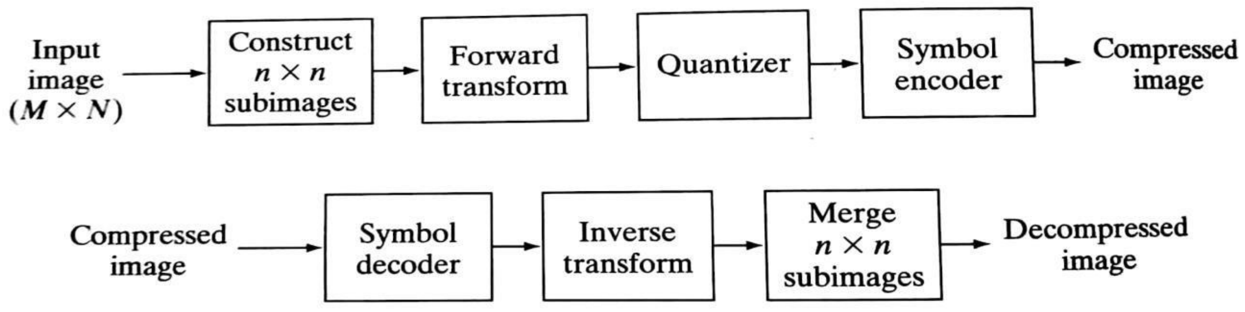
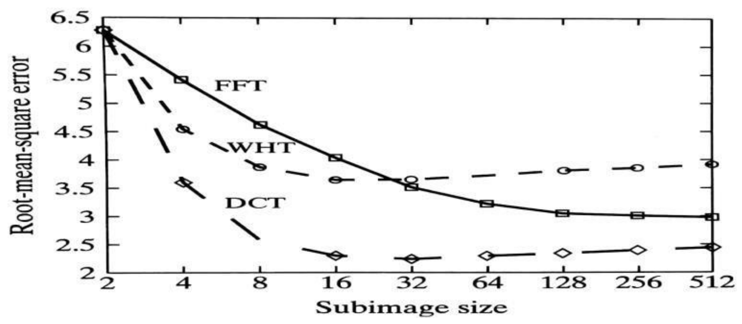


Figure : A block transform coding system i. encoder ii. Decoder.

Subimage size selection

Both transform coding reconstruction error and computational complexity are functions of the subimage size. Usually a (integer power of two) size is chosen that will reduce the correlation. The following figure illustrates the graphically the impact of subimage size on transform coding reconstruction error. between adjacent subimages in the reconstructed image.



Bit allocation

The process of truncating, quantizing and coding the coefficients of the transformed subimage is referred to as bit allocation. The reconstruction error in a decompressed image is a function of

- The number of coefficients discarded
- The precision used to store the remaining coefficients
- The relative importance of these coefficients

Most transform coding systems select which coefficients to retain

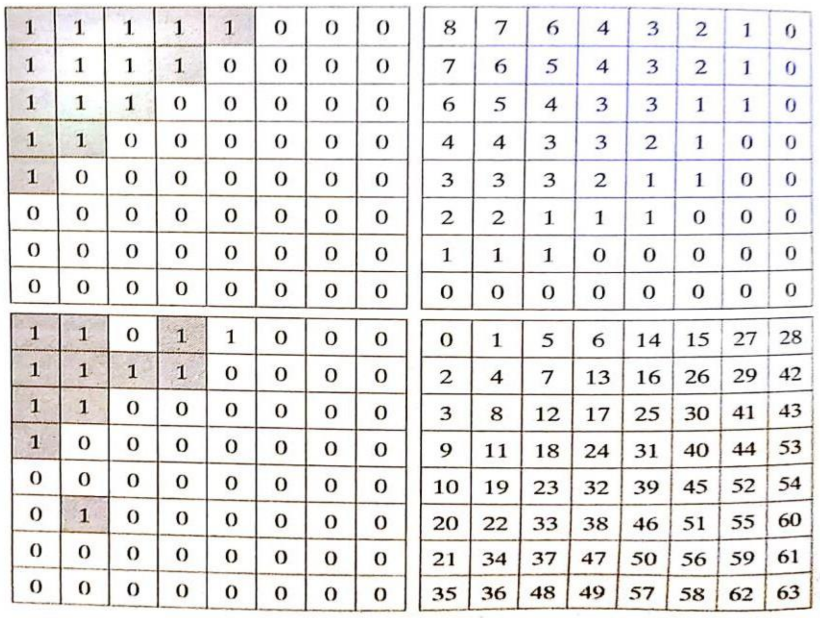
- On the basis of maximum variance – zonal coding
- On the basis of maximum magnitude – threshold coding

Zonal coding - transform coefficients are selected using a fix filter mask, which has been formed to fit the maximum variance of an average image. Knowing which transform coefficients that are going to be retained makes it possible to optimize the transform computation – only the selectedtransform coefficients need to be computed.

Threshold coding - coefficients are selected with respect to their magnitude. There are several ways of implementing threshold coding

- Global thresholding – use the same (global) threshold value for all images
- Adaptive thresholding – the threshold value is based on the coefficients
- Local thresholding – the threshold value is based on the location of the coeff
- N-largest – the N largest coefficients are retained (requires ordering)

Note that some thresholding coding schemes result in a predefined reconstruction error rather than a specified compression rate



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

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

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

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

a b
c d

FIGURE 8.29 A typical (a) zonal mask, (b) zonal bit allocation, (c) threshold mask, and (d) thresholded coefficient ordering sequence. Shading highlights the coefficients that are retained.

Predictive coding

Provides a data representation where code words express source symbol deviations from predicted values (usually values of neighboring pixels). Predictive coding efficiently reduces interpixel redundancies

- 1D & 2D – pixels are predicted from neighboring pixels
- 3D – pixels are predicted between frames as well

Works well for all images with a high degree of interpixel redundancies. Works in the presence of noise (just not as efficiently)

Input data 22222222222666666666666666999999999999999999

Code 20000000000400000000000000300000000000000000

Predictive coding can be used in both lossless and lossy compression schemes

Lossless predictive coding

The following figure shows the basic components of lossless predictive coding system. The system consist of an encoder and decoder, each containing an identical predictor. As successive samples of discrete time input signal, f(n), are introduced to the encoder, the predictor generates the anticipated value of each sample based on a specified number of past samples. The output of the predictor is then rounded to the nearest integer, The prediction error

$$e(n) = f(n) - \hat{f}(n)$$

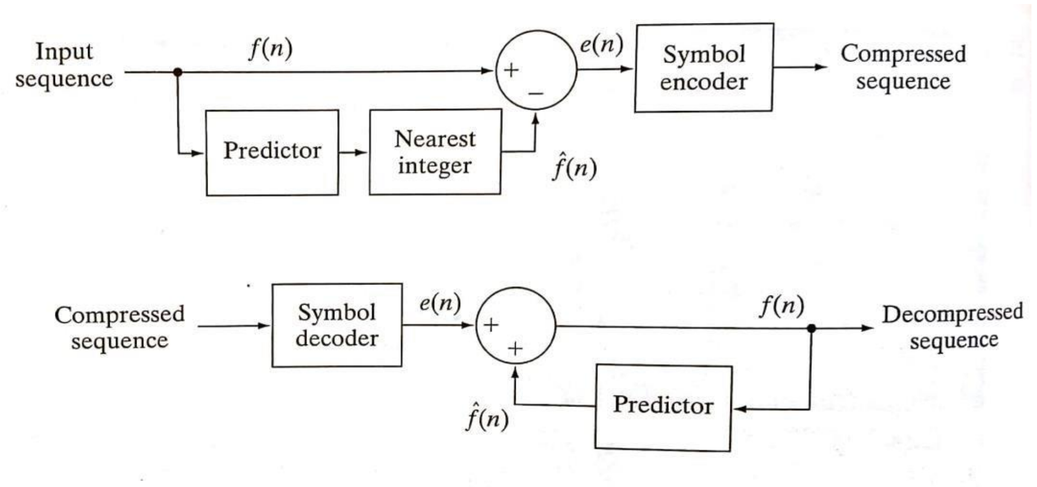


Figure : A lossless predictive coding model: i. encoder ii. Decoder.

The decoder shown in second part of the figure reconstruct e(n) from the received variable-length code words and performs the inverse operation to decompress or recreate the original input sequence.

$$f(n) = e(n) + \hat{f}(n)$$

Lossy predictive coding

In this lossy predictive coding, we add quantizer to the lossless predictive coding model introduced earlier and examine the trade –off between reconstruction accuracy and compression performance within the context of spatial predictors.

As it shown in following figure the quantizer, which replaces 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 that occurs. For a lossy encoder's predictor within the feedback loop,

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n)$$

The second part of the figure shows the lossy decoder.

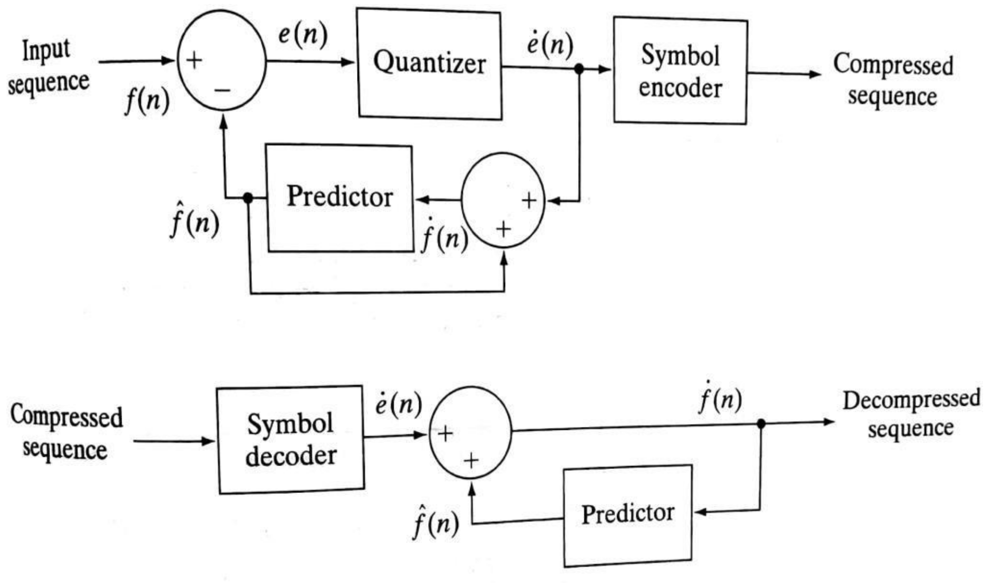


Figure : A lossy predictive coding model: i. encoder ii. Decoder

Wavelet coding

Like transform coding, wavelet coding is based on the premise that using a linear transform (here a wavelet transform) will result in transform coefficients that can be stored more efficiently than the pixels themselves. Due to the facts that wavelets are computationally efficient and that the wavelet basis functions are limited in duration, subdivision of the original image is Unnecessary. Wavelet coding typically produces a more efficient compression than DCTbased systems

- Objectively – the mean square error of wavelet-based reconstructions are typically lower
- Subjectively - the blocking artifacts (characteristic of DCT-based systems at high compression ratios) are not present in wavelet reconstructions

The choice of wavelet to use greatly affects the compression efficiency

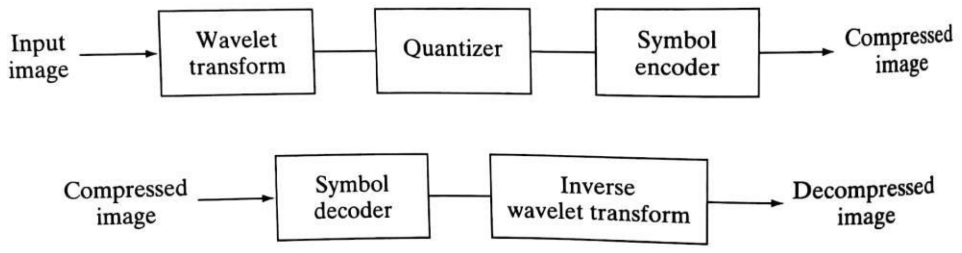
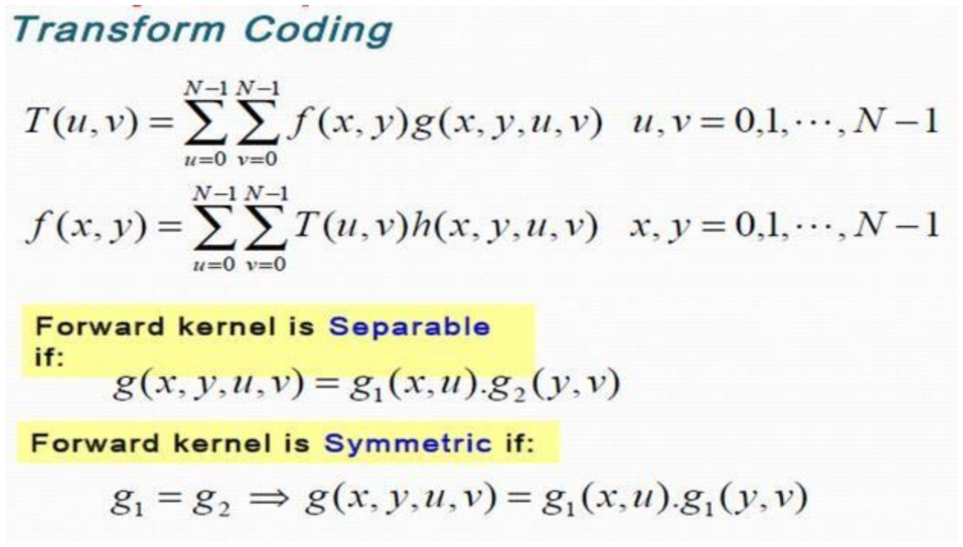


Figure : A wavelet coding system i. Encoder ii. Decoder



Transform Coding

$$T(u,v)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} f(x,y)g(x,y,u,v) \quad u,v=0,1,\cdots,N-1$$

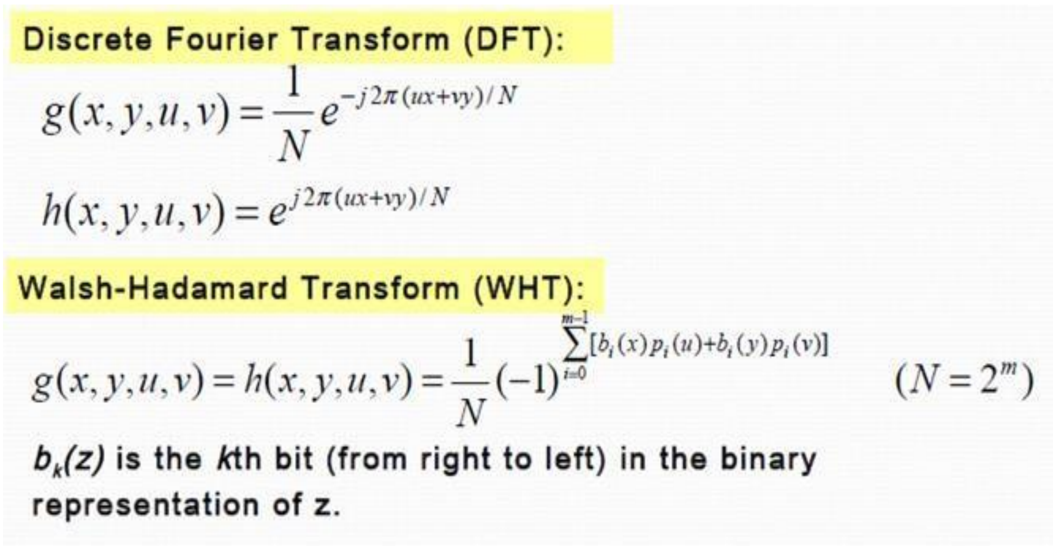
$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} T(u,v)h(x,y,u,v) \quad x,y=0,1,\cdots,N-1$$

Forward kernel is Separable if:

$$g(x,y,u,v)=g_1(x,u).g_2(y,v)$$

Forward kernel is Symmetric if:

$$g_1=g_2 \Rightarrow g(x,y,u,v)=g_1(x,u).g_1(y,v)$$



Discrete Fourier Transform (DFT):

$$g(x,y,u,v)=\frac{1}{N}e^{-j2\pi(ux+vy)/N}$$

$$h(x,y,u,v)=e^{j2\pi(ux+vy)/N}$$

Walsh-Hadamard Transform (WHT):

$$g(x,y,u,v)=h(x,y,u,v)=\frac{1}{N}(-1)^{\sum_{i=0}^{m-1}[b_i(x)p_i(u)+b_i(y)p_i(v)]} \qquad (N=2^m)$$

$b_k(z)$ **is the** ***k*****th bit (from right to left) in the binary representation of z.**