Unit 3. Data Compression

CS 3570
Shang-Hong Lai
CS Dept., NTHU

Content

- Introduction
- Lossless compression
- Lossy compression
- JPEG Image Compression

Introduction

- Compression: the process of coding that will effectively reduce the total number of bits needed to represent certain information.

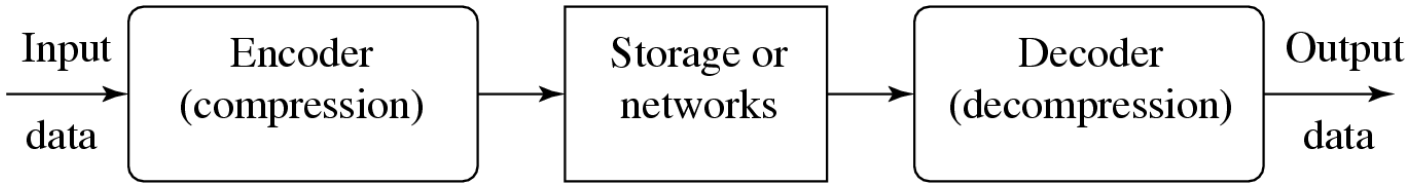


Fig. 7.1: A General Data Compression Scheme.

- If the compression and decompression processes induce no information loss, then the compression scheme is lossless; otherwise, it is lossy.
- Compression ratio:

$$compression\,ratio = \frac{B_0}{B_1}$$

- B0 – # of bits before compression, B1 – # of bits after compression

Image compression

- Image compression may be **lossy** or **lossless**.
- Lossless compression is preferred for archival purposes and often for medical imaging, technical drawing or comics.
- Lossy compression is suitable for natural images where minor loss of fidelity is acceptable.

Lossless Compression Algorithms

- Introduction
- Basics of Information Theory
- Run-Length Coding
- Variable-Length Coding (VLC)
- Dictionary-based Coding
- Arithmetic Coding
- Lossless Image Compression

Basics of Information Theory

- The entropy η of an information source with alphabet S = {s_1, s_2, . . . , s_n} is:

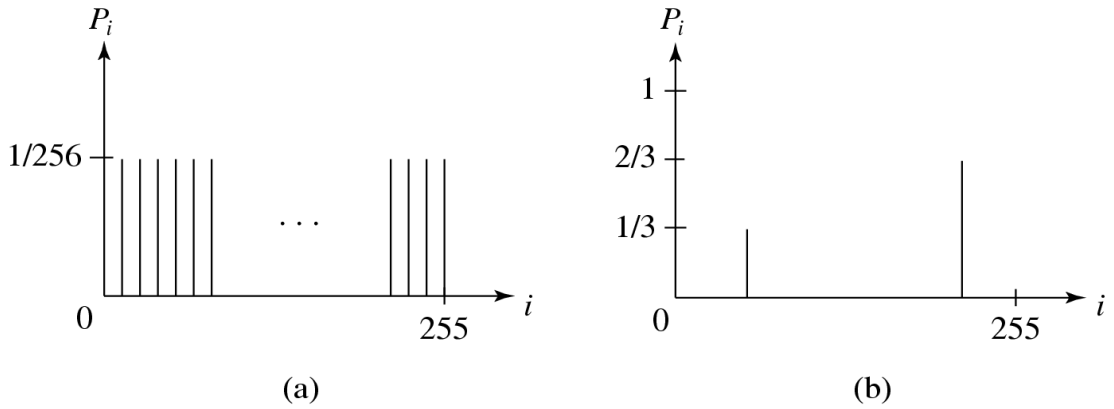
$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \tag{7.2}$$

$$= -\sum_{i=1}^{n} p_i \log_2 p_i \tag{7.3}$$

p_i – probability that symbol s_i will occur in S.

$\log_2 \frac{1}{p_i}$ – indicates the amount of information (self-information as defined by Shannon) contained in s_i, which corresponds to the number of bits needed to encode s_i.

Distribution of Gray-Level Intensities



- **Fig. 7.2**: Histograms for Two Gray-level Images.

- Fig. 7.2(a) shows the histogram of an image with *uniform* distribution of gray-level intensities, i.e., $\forall i\ p_i = 1/256$. Hence, the entropy of this image is:

$$\log_2 256 = 8 \qquad (7.4)$$

- Fig. 7.2(b) shows the histogram of an image with two possible values. Its entropy is 0.92.

Entropy and Code Length

- In Eq. (7.3): the entropy η is a weighted-sum of terms $\log_2 \frac{1}{p_i}$; hence it represents the *average* amount of information contained per symbol in the source S.

- The entropy η specifies the lower bound for the average number of bits to code each symbol in S, i.e.,

$$\eta \leq \bar{l} \qquad (7.5)$$

- $\bar{l}$ - the average length (measured in bits) of the codewords produced by the encoder.

Run-Length Coding

- Memoryless Source: an information source that is independently distributed. Namely, the value of the current symbol does not depend on the values of the previously appeared symbols.
- Instead of assuming memoryless source, Run-Length Coding (RLC) exploits memory present in the information source.
- Rationale for RLC: if the information source has the property that symbols tend to form continuous groups, then such symbol and the length of the group can be coded.

Variable-Length Coding (VLC)

- Shannon-Fano Algorithm

— a top-down approach

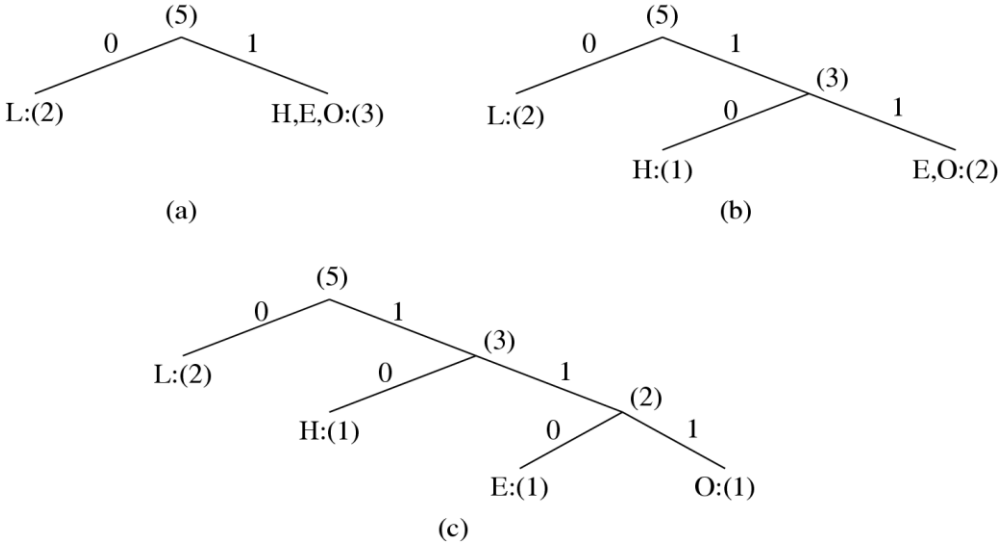
  - Sort the symbols according to the frequency count of their occurrences.
  - Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.

- An Example: coding of “HELLO”

Symbol	H	E	L	O
Count	1	1	2	1

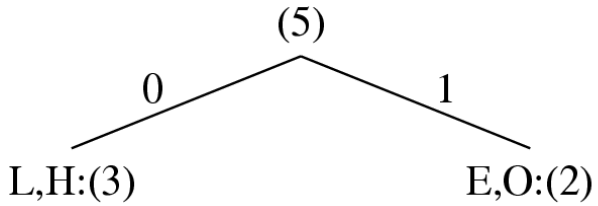
Frequency count of the symbols in ”HELLO”.

Coding Tree for HELLO by Shannon-Fano

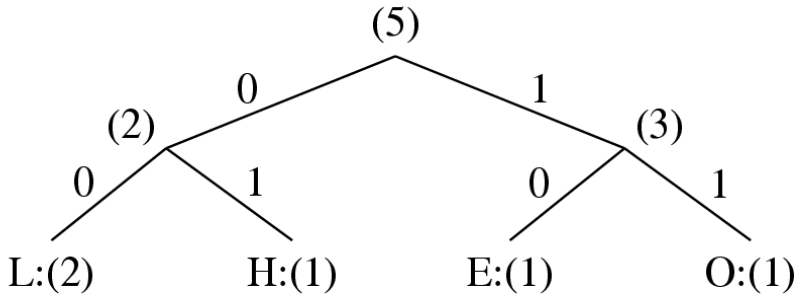


Symbol	Count	Log_2	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
			TOTAL # of bits:	10

Another coding tree for HELLO by Shannon-Fano



(a)



(b)

Symbol	Count	Log_2	Code	# of bits used
L	2	1.32	00	4
H	1	2.32	01	2
E	1	2.32	10	2
O	1	2.32	11	2
			TOTAL # of bits:	10

Huffman Coding

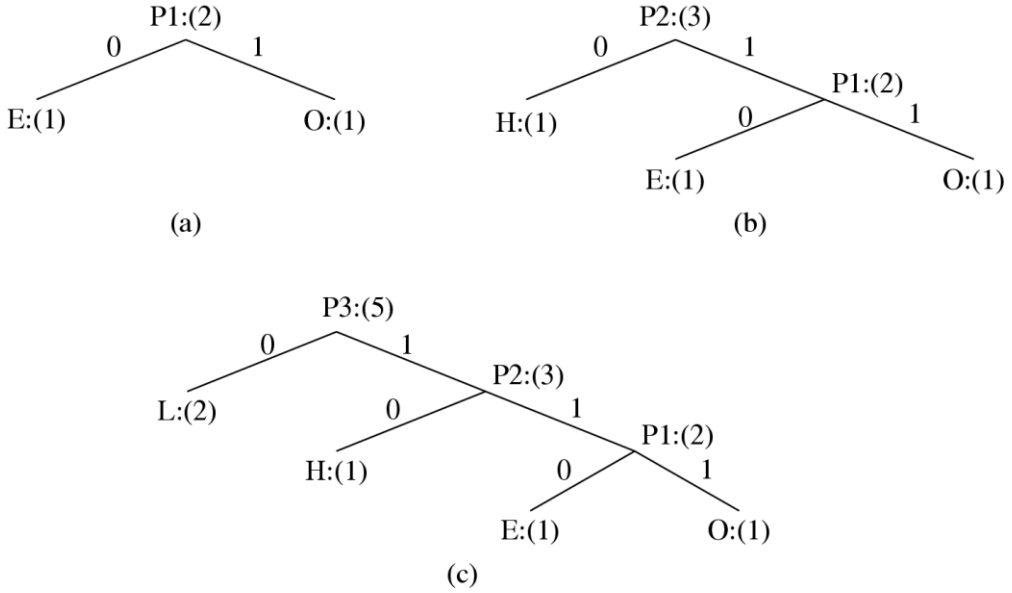
- ALGORITHM 7.1 Huffman Coding Algorithm
- — a bottom-up approach

- 1. Initialization: Put all symbols on a list sorted according to their frequency counts.

- 2. Repeat until the list has only one symbol left:
  - From the list pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node.
  - Assign the sum of the children's frequency counts to the parent and insert it into the list such that the order is maintained.
  - Delete the children from the list.

- 3. Assign a codeword for each leaf based on the path from the root.

Coding Tree for "HELLO" Using Huffman Algorithm



- After initialization: L H E O
- After iteration (a): L P1 H
- After iteration (b): L P2
- After iteration (c): P3

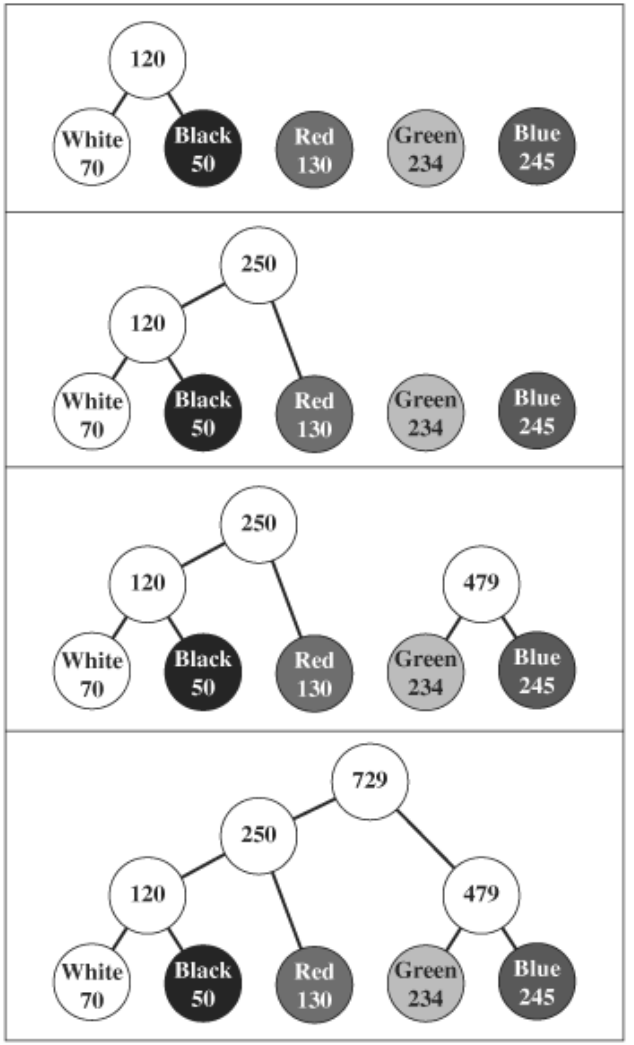
Huffman encoding - example

- The image file has only 729 pixels in it, with the following colors and the corresponding frequencies
  - White 70
  - Black 50
  - Red 130
  - Green 234
  - Blue 245

- A node is created for each of the colors in the image, with the frequency of that color's appearance stored in the node

Huffman encoding - example

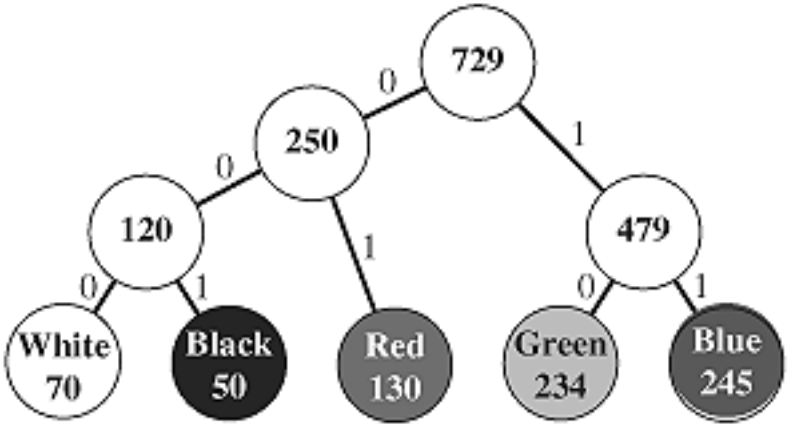
- Now the two nodes with the smallest value for *freq* are joined such that they are the children of a common parent node, and the parent node's *freq* value is set to the sum of the *freq* values in the children nodes
- This node-combining process repeats until you arrive at the creation of a root node



Huffman encoding - example

- Once the tree has been created, the branches are labeled with 0s on the left and 1s on the right
- After the codes have been created, the image file can be compressed using these codes.

Label branches with 0s on the left and 1s on the right.



For each leaf node, traverse tree from root to leaf node and gather code for the color associated with the leaf node.

White	000
Black	001
Red	01
Green	10
Blue	11

Note that not all codes are the same number of bits, and no code is a prefix of any other code.

Properties of Huffman Coding

- Unique Prefix Property: No Huffman code is a prefix of any other Huffman code - precludes any ambiguity in decoding.

- Optimality: minimum redundancy code - proved optimal for a given data model (i.e., a given, accurate, probability distribution):
  - The two least frequent symbols will have the same length for their Huffman codes, differing only at the last bit.
  - Symbols that occur more frequently will have shorter Huffman codes than symbols that occur less frequently.
  - The average code length for an information source S is strictly less than η + 1. Combined with Eq. (7.5), we have:

$$\bar{l} < \eta + 1 \qquad (7.6)$$

Extended Huffman Coding

- Motivation: All codewords in Huffman coding have integer bit lengths. It is wasteful when p_i is very large and hence $\log_2 \frac{1}{p_i}$ is close to 0.
- Why not group several symbols together and assign a single codeword to the group as a whole?
- Extended Alphabet: For alphabet S = {s1, s2, . . . , sn}, if k symbols are grouped together, then the extended alphabet is:

$$S^{(k)} = \{\overbrace{s_1 s_1 \cdots s_1}^{k \text{ symbols}}, s_1 s_1 \cdots s_2, \cdots, s_1 s_1 \cdots s_n, s_1 s_1 \cdots s_2 s_1, \cdots, s_n s_n \cdots s_n\}.$$

— the size of the new alphabet S(k) is nk.

Extended Huffman Coding (Cont'd)

- It can be proven that the average # of bits for each symbol is:

$$\eta \le \bar{l} < \eta + \frac{1}{k} \qquad (7.7)$$

An improvement over the original Huffman coding, but not much.

- Problem: If k is relatively large (e.g., k ≥ 3), then for most practical applications where n ≫ 1, nk implies a huge symbol table — impractical.

Dictionary-based Coding

- LZW uses fixed-length codewords to represent variable-length strings of symbols/characters that commonly occur together, e.g., words in English text.
- The LZW encoder and decoder build up the same dictionary dynamically while receiving the data.
- LZW places longer and longer repeated entries into a dictionary, and then emits the code for an element, rather than the string itself, if the element has already been placed in the dictionary.

- ALGORITHM 7.2 - LZW Compression

```
BEGIN
        s = next input character;
        while not EOF
        {
          c = next input character;

          if s + c exists in the dictionary
                  s = s + c;
          else
             {
                  output the code for s;
                  add string s + c to the dictionary with a new code;
                  s = c;
             }
        }
        output the code for s;
END
```

Example of LZW compression

- LZW compression for string “ABABBABCABABBA”
- Let’s start with a very simple dictionary (also referred to as a “string table”), initially containing only 3 characters, with codes as follows:

Code	String
1	A
2	B
3	C

- Now if the input string is “ABABBABCABABBA”, the LZW compression algorithm works as follows:

LZW Compression for "ABABBABCABABBA"

S	C	Output	Code	String
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

- The output codes are: 1 2 4 5 2 3 4 6 1. Instead of sending 14 characters, only 9 codes need to be sent (compression ratio = 14/9 = 1.56).

LZW Decompression

- **ALGORITHM 7.3 LZW Decompression (simple version)**

```
BEGIN
   s = NIL;
   while not EOF
    {
     k = next input code;
     entry = dictionary entry for k;
     output entry;
     if (s != NIL)
        add string s + entry[0] to dictionary with a new code;
        s = entry;
    }
END
```

- **Example 7.3:** LZW decompression for string "ABABBABCABABBA".
  Input codes to the decoder are 1 2 4 5 2 3 4 6 1.
  The initial string table is identical to what is used by the encoder.

LZW Decompression

- The LZW decompression algorithm then works as follows:

S	K	Entry/output	Code	String
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

- Apparently, the output string is “ABABBABCABABBA”, a truly lossless result!

- **ALGORITHM 7.4 LZW Decompression (modified)**

```
BEGIN
  s = NIL;
  while not EOF
  {
    k = next input code;
    entry = dictionary entry for k;

    /* exception handler */
    if (entry == NULL)
                entry = s + s[0];

    output entry;
    if (s != NIL)
        add string s + entry[0] to dictionary with a new code;
        s = entry;
    }
END
```

LZW Coding (Cont'd)

- In real applications, the code length l is kept in the range of [l0, lmax]. The dictionary initially has a size of 2^l0. When it is filled up, the code length will be increased by 1; which can be repeated until l = lmax.
- If the data lacks any repetitive structure, the chance of using the new codes in the dictionary entries could be low. This may lead to data expansion instead of data reduction.
- To deal with this problem, the algorithm can build in two modes: compressed and transparent. The latter turns off compression and is invoked when data expansion is detected.
- When lmax is reached and the dictionary is filled up, it needs to be flushed (as in Unix compress, or to have the LRU (least recently used) entries removed.

Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually outperforms Huffman coding.
- Huffman coding assigns each symbol a codeword which has an integral bit length. Arithmetic coding can treat the whole message as one unit.
- A message is represented by a half-open interval [a, b) where a and b are real numbers between 0 and 1. Initially, the interval is [0, 1). When the message becomes longer, the length of the interval shortens and the number of bits needed to represent the interval increases.

Arithmetic Coding Encoder

- **ALGORITHM 7.5 Arithmetic Coding Encoder**

```
BEGIN
    low = 0.0;  high = 1.0;  range = 1.0;

    while (symbol != terminator)
    {
        get (symbol);
        high = low + range * Range_high(symbol);
        low = low + range * Range_low(symbol);
        range = high - low;
    }

    output a code so that low <= code < high;
END
```

Example of Arithmetic Coding

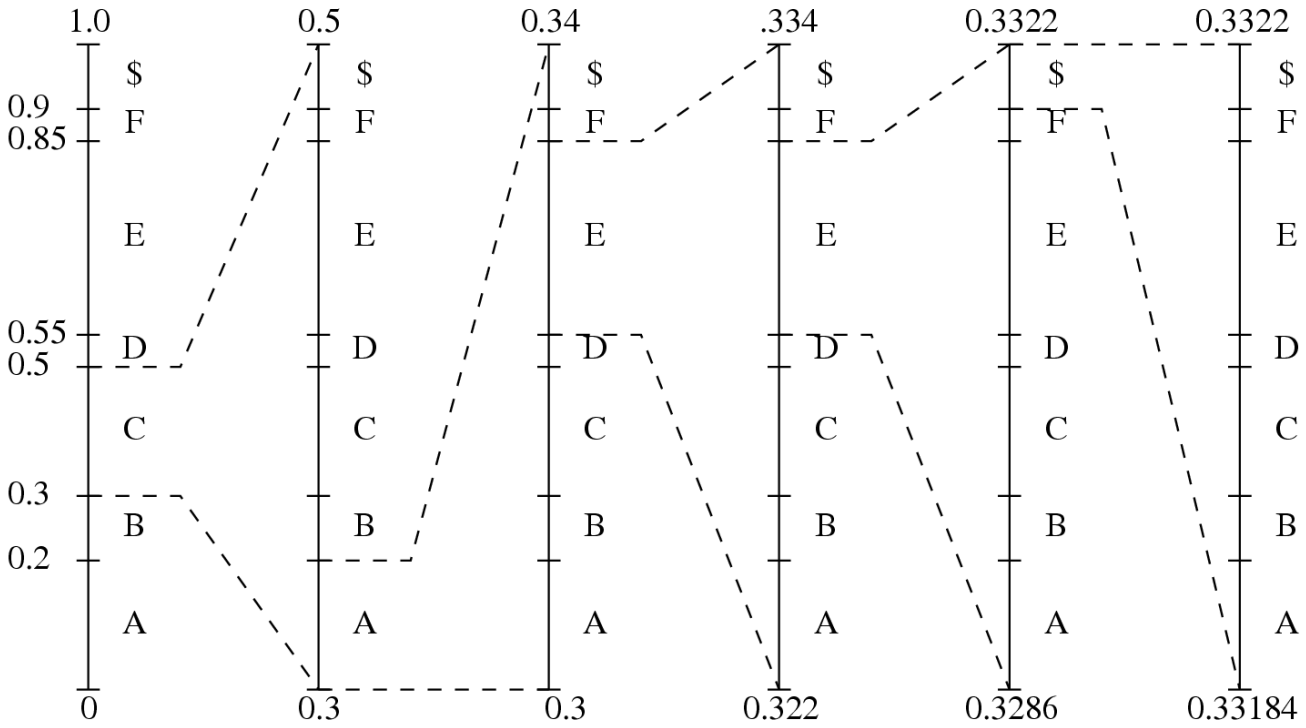
- **Example: Encoding in Arithmetic Coding**

Symbol	Probability	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

(a) Probability distribution of symbols.

- **Fig. 7.9**: Arithmetic Coding: Encode Symbols "CAEE$"

Example of Arithmetic Coding



Symbol	Prob.	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

Graphical display of shrinking ranges for encoding symbols “CAEE$”.

Example of Arithmetic Coding

- **Example: Encoding in Arithmetic Coding**

Symbol	Low	High	Range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

$$range = P_C \times P_A \times P_E \times P_E \times P_\$ = 0.2 \times 0.2 \times 0.3 \times 0.3 \times 0.1 = 0.00036$$

(c) New *low*, *high*, and *range* generated.

Arithmetic Coding: Encode Symbols “CAEE$”

Arithmetic Encoding

- **PROCEDURE 7.2 Generating Codeword for Encoder**

```
BEGIN
    code = 0;
    k = 1;
    while (value(code) < low)
    {
        assign 1 to the kth binary fraction bit
        if (value(code) > high)
                replace the kth bit by 0
        k = k + 1;
    }
END
```

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range [*low*, *high*). The above algorithm will ensure that the shortest binary codeword is found.

Arithmetic Decoder

- **ALGORITHM 7.6 Arithmetic Coding Decoder**

```
BEGIN
    get binary code and convert to
    decimal value = value(code);
    DO
    {
        find a symbol s so that
                Range_low(s) <= value < Range_high(s);
        output s;
        low = Rang_low(s);
        high = Range_high(s);
        range = high - low;
        value = [value - low] / range;
    }
    UNTIL symbol s is a terminator
END
```

- **Table 7.5:** Arithmetic coding: decode symbols “CAEE$”

Value	Output Symbol	Range_low	Range_high	range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

Lossless Image Compression

- **Approaches of Differential Coding of Images:**
  - Given an original image $I(x, y)$, using a simple difference operator we can define a difference image $d(x, y)$ as follows:

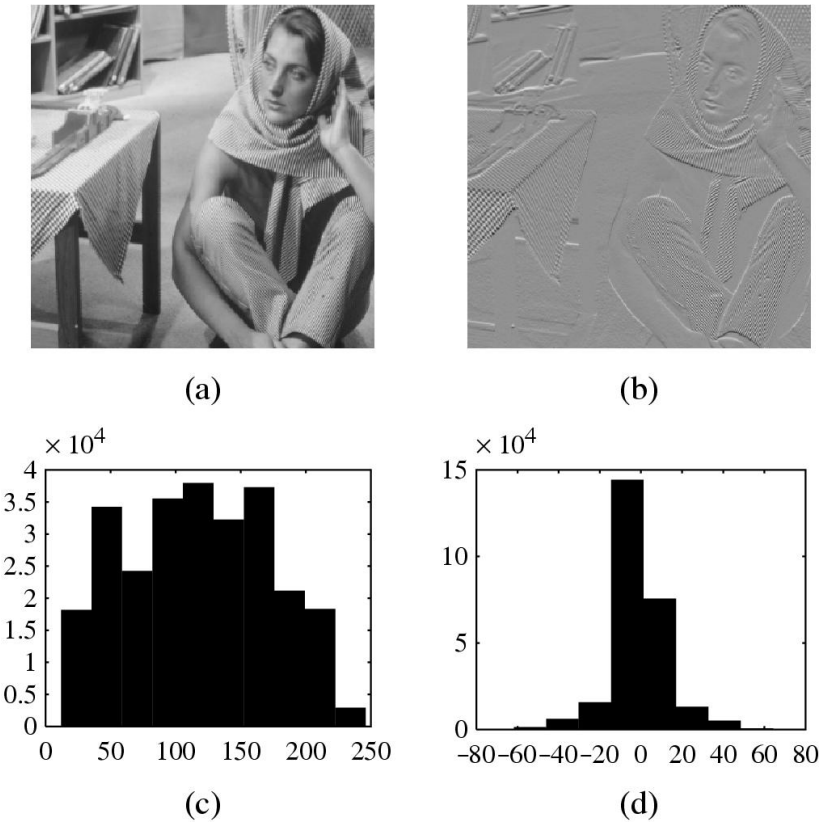
$$d(x, y) = I(x, y) - I(x - 1, y) \quad (7.9)$$

  - or use the discrete version of the 2-D Laplacian operator to define a difference image $d(x, y)$ as

$$d(x, y) = 4\,I(x, y) - I(x, y - 1) - I(x, y + 1) - I(x + 1, y) - I(x - 1, y) \quad (7.10)$$

- Due to *spatial redundancy* existed in normal images I, the difference image d will have a narrower histogram and hence a smaller entropy.

Example of Differential Image Coding



Distributions for Original versus Derivative Images. (a,b): Original gray-level image and its partial derivative image; (c,d): Histograms for original and derivative images.

Lossless JPEG

- Lossless JPEG: A special case of the JPEG image compression.
- The Predictive method
- Forming a differential prediction: A predictor combines the values of up to three neighboring pixels as the predicted value for the current pixel, indicated by 'X' in Fig. 7.10. The predictor can use any one of the seven schemes listed in Table 7.6.
- Encoding: The encoder compares the prediction with the actual pixel value at the position 'X' and encodes the difference using one of the lossless compression techniques we have discussed, e.g., the Huffman coding scheme.

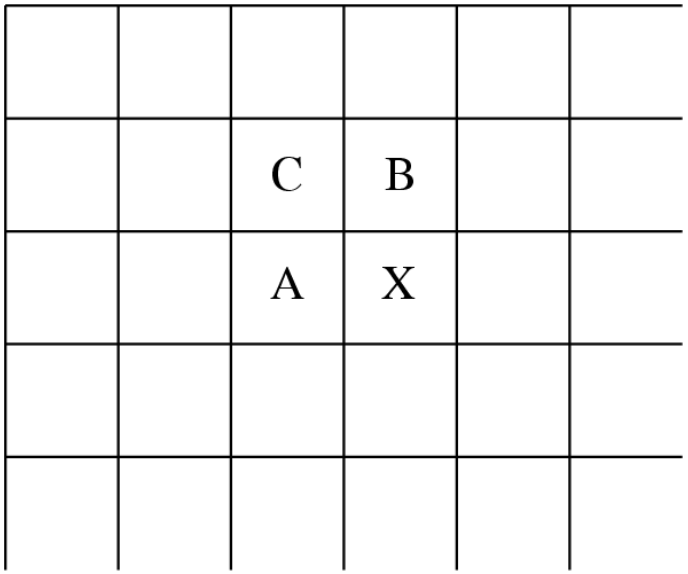


Fig. 7.10: Neighboring Pixels for Predictors in Lossless JPEG.

- **Note**: Any of A, B, or C has already been decoded before it is used in the predictor, on the decoder side of an encode-decode cycle.

- **Table 7.6**: Predictors for Lossless JPEG

Predictor	Prediction
P1	A
P2	B
P3	C
P4	A + B – C
P5	A + (B – C) / 2
P6	B + (A – C) / 2
P7	(A + B) / 2

- **Table 7.7:** Comparison with other lossless compression programs

Compression Program	Compression Ratio			
	Lena	Football	F-18	Flowers
Lossless JPEG	1.45	1.54	2.29	1.26
Optimal Lossless JPEG	1.49	1.67	2.71	1.33
Compress (LZW)	0.86	1.24	2.21	0.87
Gzip (LZ77)	1.08	1.36	3.10	1.05
Gzip -9 (optimal LZ77)	1.08	1.36	3.13	1.05
Pack(Huffman coding)	1.02	1.12	1.19	1.00

Lossy Compression Algorithms

- Distortion Measures
- Rate-Distortion Theory
- Quantization
- Transform Coding
  - DCT

Introduction

- Lossless compression algorithms do not deliver compression ratios that are high enough for most multimedia applications. Hence, most multimedia compression algorithms are lossy.

- What is lossy compression?
  - The compressed data is not the same as the original data, but a close approximation of it.
  - Yields a much higher compression ratio than that of lossless compression.

Distortion Measures

The three most commonly used distortion measures in image compression are:

– ***Mean Squared Error*** **(MSE)** σ_d^2 ,

$$\sigma_d^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2 \tag{8.1}$$

where x_n, y_n, and N are the input data sequence, reconstructed data sequence, and length of the data sequence, respectively.

– *Signal to Noise Ratio* (SNR), in decibel units (dB),

$$SNR = 10\log_{10}\frac{\sigma_x^2}{\sigma_d^2} \tag{8.2}$$

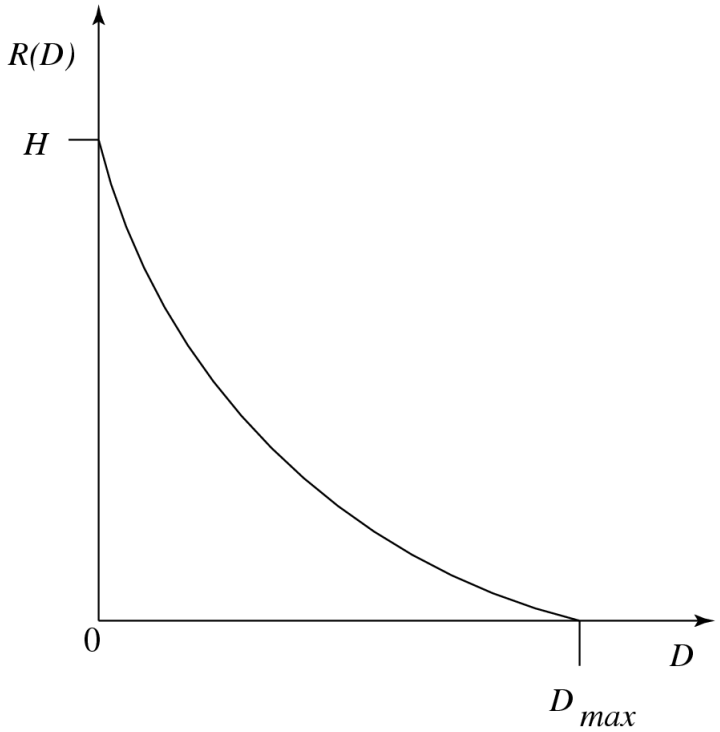
where σ_x^2 is the average square value of the original data sequence and σ_d^2 is the MSE.

– *Peak Signal to Noise Ratio* (PSNR),

$$PSNR = 10\log_{10}\frac{x_{peak}^2}{\sigma_d^2} \tag{8.3}$$

The Rate-Distortion Theory

- Provides a framework for the study of tradeoffs between Rate and Distortion.



Typical Rate Distortion Function.

Quantization

- Reduce the number of distinct output values to a much smaller set.
- Main source of the “loss” in lossy compression.
- Three different forms of quantization.
  - Uniform: midrise and midtread quantizers.
  - Nonuniform: companded quantizer.
  - Vector Quantization.

Uniform Scalar Quantization

- A uniform scalar quantizer partitions the domain of input values into equally spaced intervals, except possibly at the two outer intervals.
  - The output or reconstruction value corresponding to each interval is taken to be the midpoint of the interval.
  - The length of each interval is referred to as the step size, denoted by the symbol Δ.

- Two types of uniform scalar quantizers:
  - Midrise quantizers have even number of output levels.
  - Midtread quantizers have odd number of output levels, including zero as one of them (see Fig. 8.2).

- For the special case where $\Delta = 1$, we can simply compute the output values for these quantizers as:

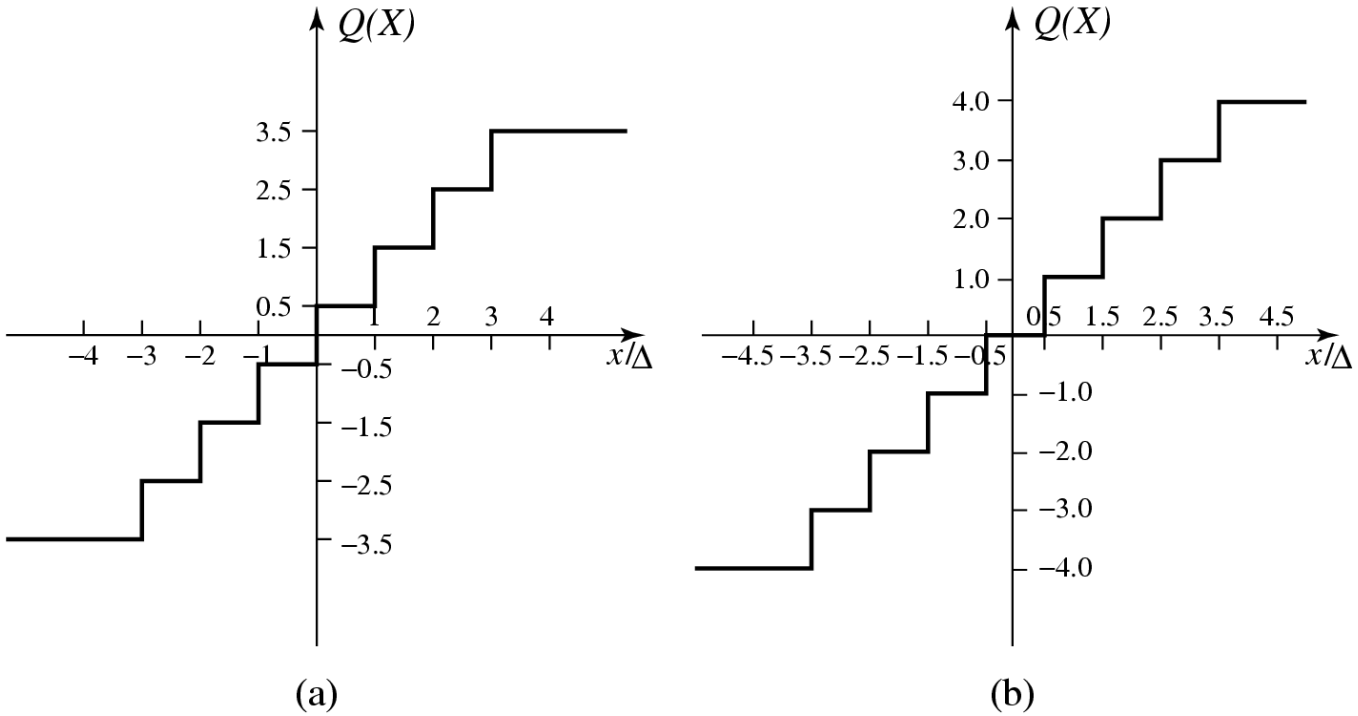
$$Q_{midrise}(x) = \lceil x \rceil - 0.5 \tag{8.4}$$

$$Q_{midtread}(x) = \lfloor x + 0.5 \rfloor \tag{8.5}$$

- Performance of an *M-level* quantizer. Let $B = \{b_0, b_1, \ldots, b_M\}$ be the set of decision boundaries and $Y = \{y_1, y_2, \ldots, y_M\}$ be the set of reconstruction or output values.

- Suppose the input is uniformly distributed in the interval $[-X_{max}, X_{max}]$. The rate of the quantizer is:

$$R = \lceil \log_2 M \rceil \tag{8.6}$$



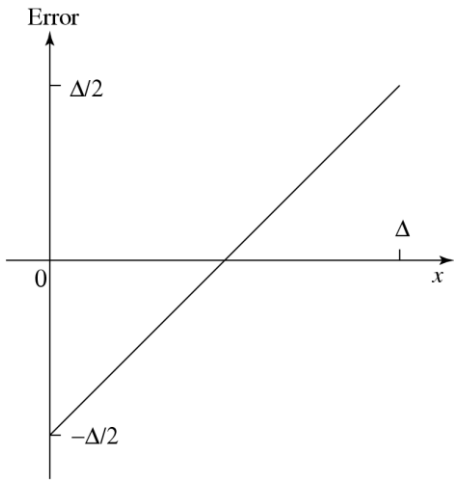
Uniform Scalar Quantizers: (a) Midrise, (b) Midtread.

Quantization Error of Uniformly Distributed Source

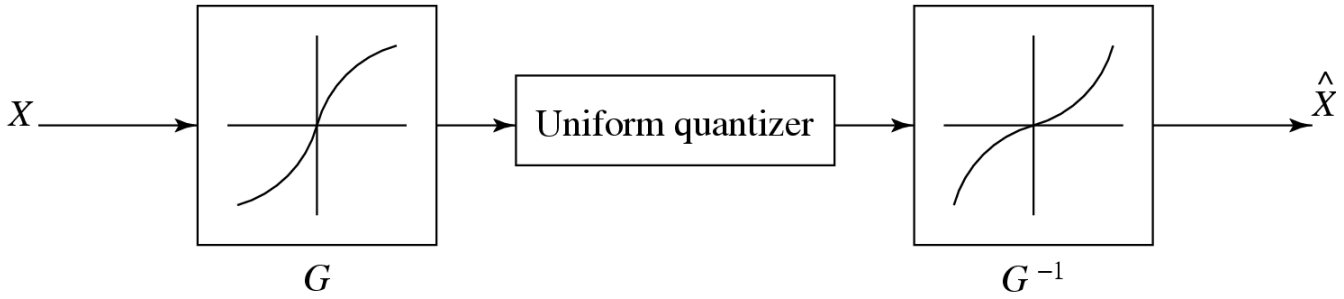
- **Granular distortion**: quantization error caused by the quantize for bounded input.
  - For granular distortion, decision boundaries b_i for a midrise quantizer are $[(i-1)\Delta, i\Delta]$, $i = 1..M/2$, covering positive data X (and another half for negative X values).
  - Output values y_i are the midpoints $i\Delta - \Delta/2$, $i = 1..M/2$, again just considering the positive data. The total distortion is

$$D_{gran} = 2\sum_{i=1}^{\frac{M}{2}} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta \right)^2 \frac{1}{2X_{max}} \, dx$$

- Since the values y_i are the midpoints of each interval, the quantization error must lie within the values $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$.



Companded Quantization



- **Fig. 8.4**: Companded quantization.

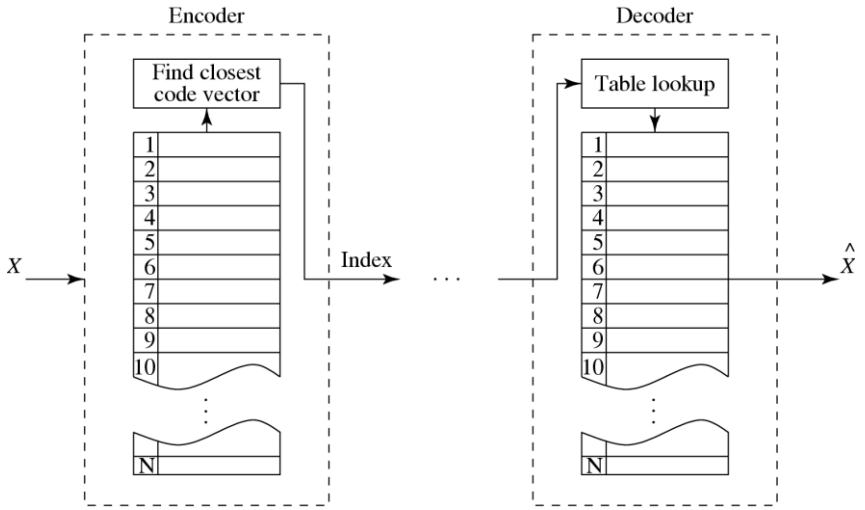
- *Companded quantization* is **nonlinear**.

- As shown above, a *compander* consists of a *compressor function* G, a uniform quantizer, and an *expander function* G^{-1}.

- The two commonly used companders are the μ-law and A-law companders.

Vector Quantization (VQ)

- According to Shannon's original work on information theory, any compression system performs better if it operates on vectors or groups of samples rather than individual symbols or samples.
- Form vectors of input samples by simply concatenating a number of consecutive samples into a single vector.
- Instead of scalar quantization, in VQ code vectors with n components are used. A collection of these code vectors form the codebook.



Transform Coding

- The rationale behind transform coding:
- If Y is the result of a linear transform T of the input vector X in such a way that the components of Y are much less correlated, then Y can be coded more efficiently than X.
- If most information is accurately described by the first few components of a transformed vector, then the remaining components can be coarsely quantized, or even set to zero, with little signal distortion.
- Discrete Cosine Transform (DCT) has been used in JPEG and will be studied in this lecture.

Spatial Frequency and DCT

- Spatial frequency indicates how many times pixel values change across an image block.
- The DCT formalizes this notion with a measure of how much the image contents change in correspondence to the number of cycles of a cosine wave per block.
- The role of the DCT is to decompose the original signal into its DC and AC components; the role of the IDCT is to reconstruct (re-compose) the signal.

Discrete Cosine Transform (DCT)

1D Discrete Cosine Transform (1D DCT) for signal f(t):

$$F(u) = \frac{C(u)}{2} \sum_{i=0}^{7} \cos \frac{(2i+1)u\pi}{16} f(i) \qquad (8.19)$$

where $u = 0, 1, \ldots, 7$.

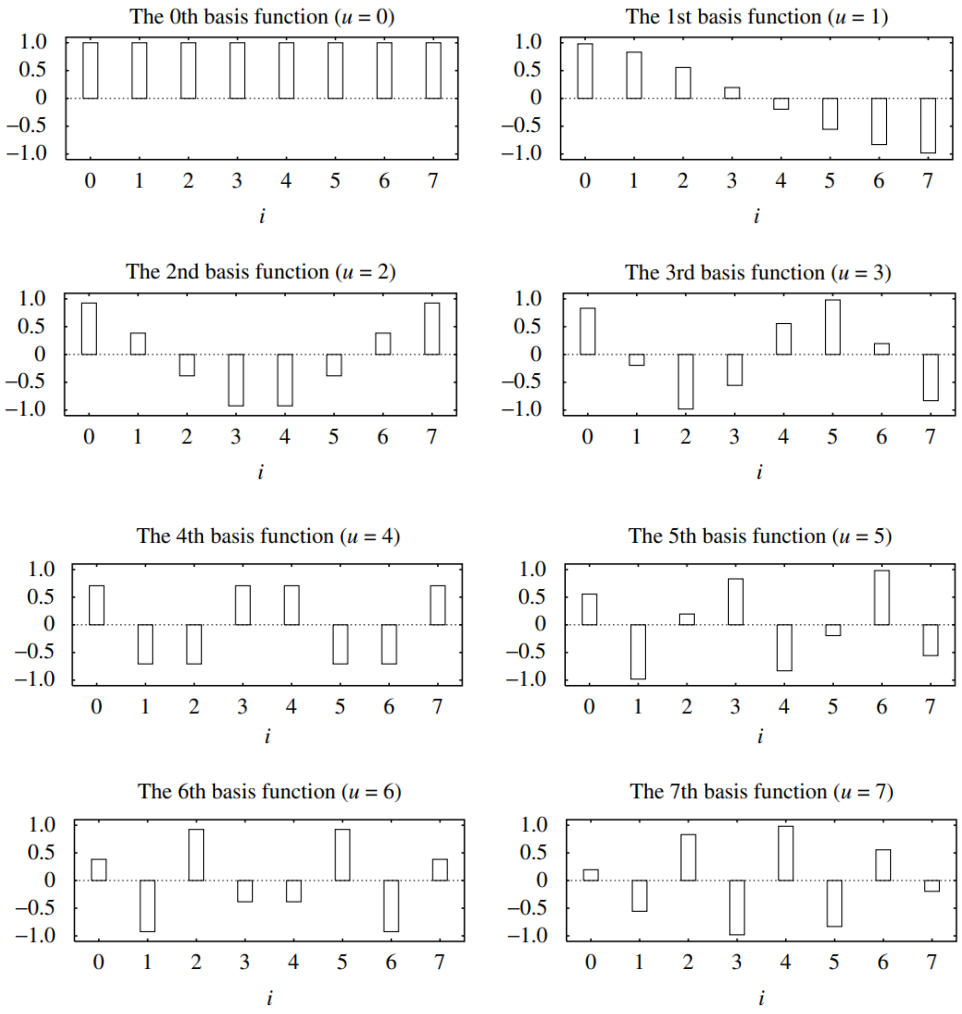
$$C(\xi) = \begin{cases} \frac{\sqrt{2}}{2} & \text{if } \xi = 0, \\ 1 & \text{otherwise.} \end{cases}$$

1D Inverse Discrete Cosine Transform (1D IDCT):

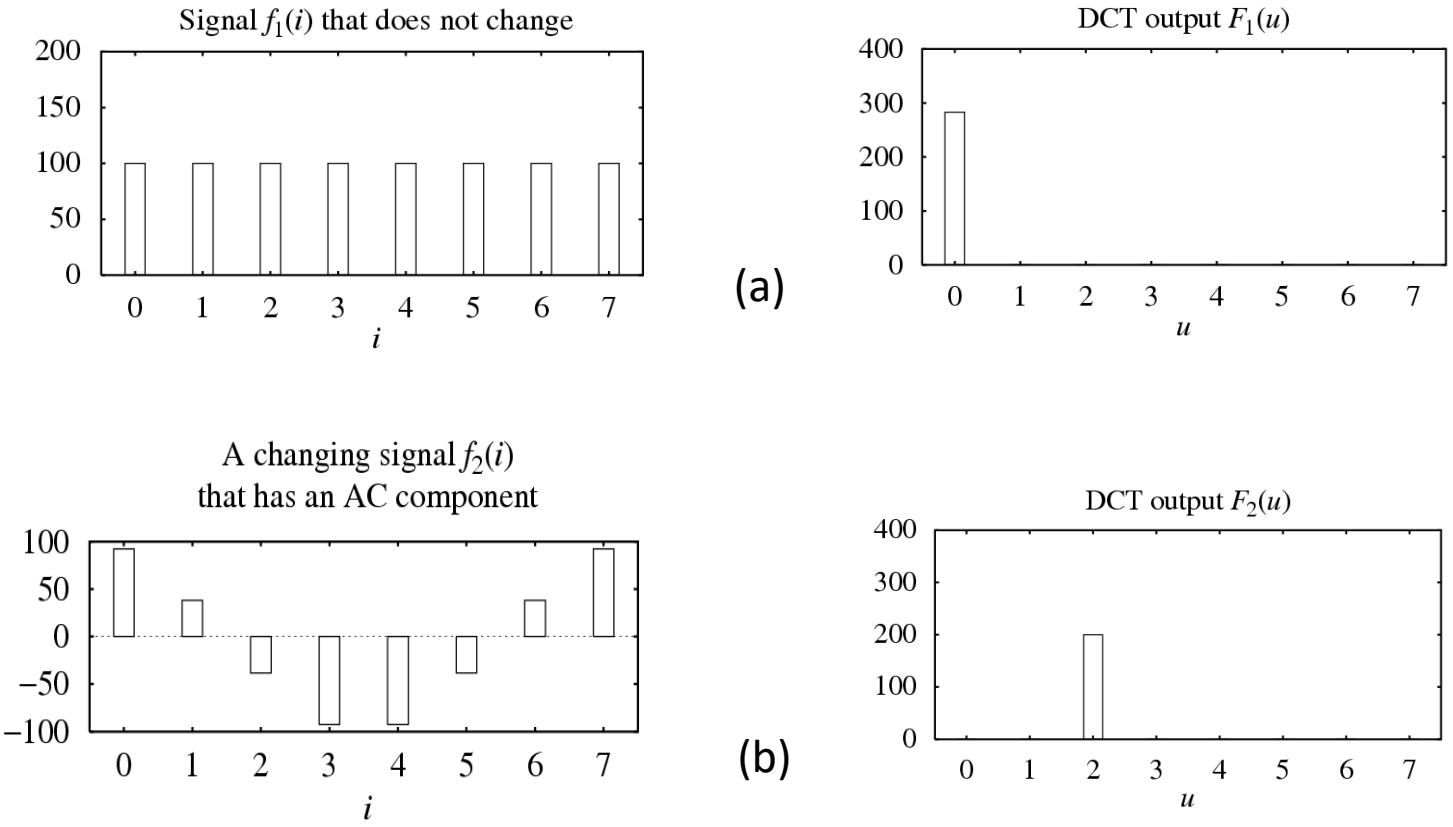
$$\tilde{f}(i) = \sum_{u=0}^{7} \frac{C(u)}{2} \cos \frac{(2i+1)u\pi}{16} F(u) \qquad (8.20)$$

where $i = 0, 1, \ldots, 7$.

1D DCT Basis Functions

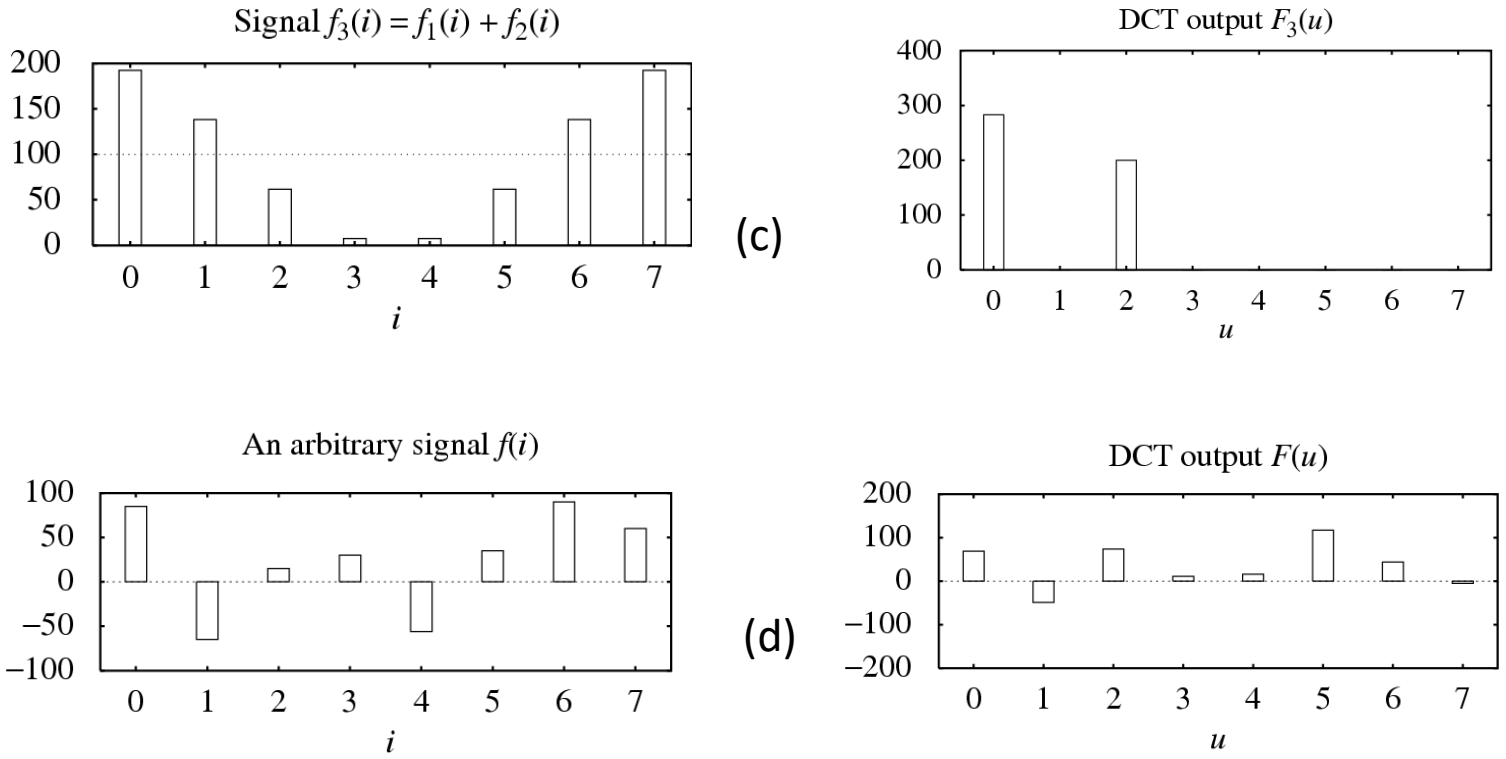


Examples of 1D DCT



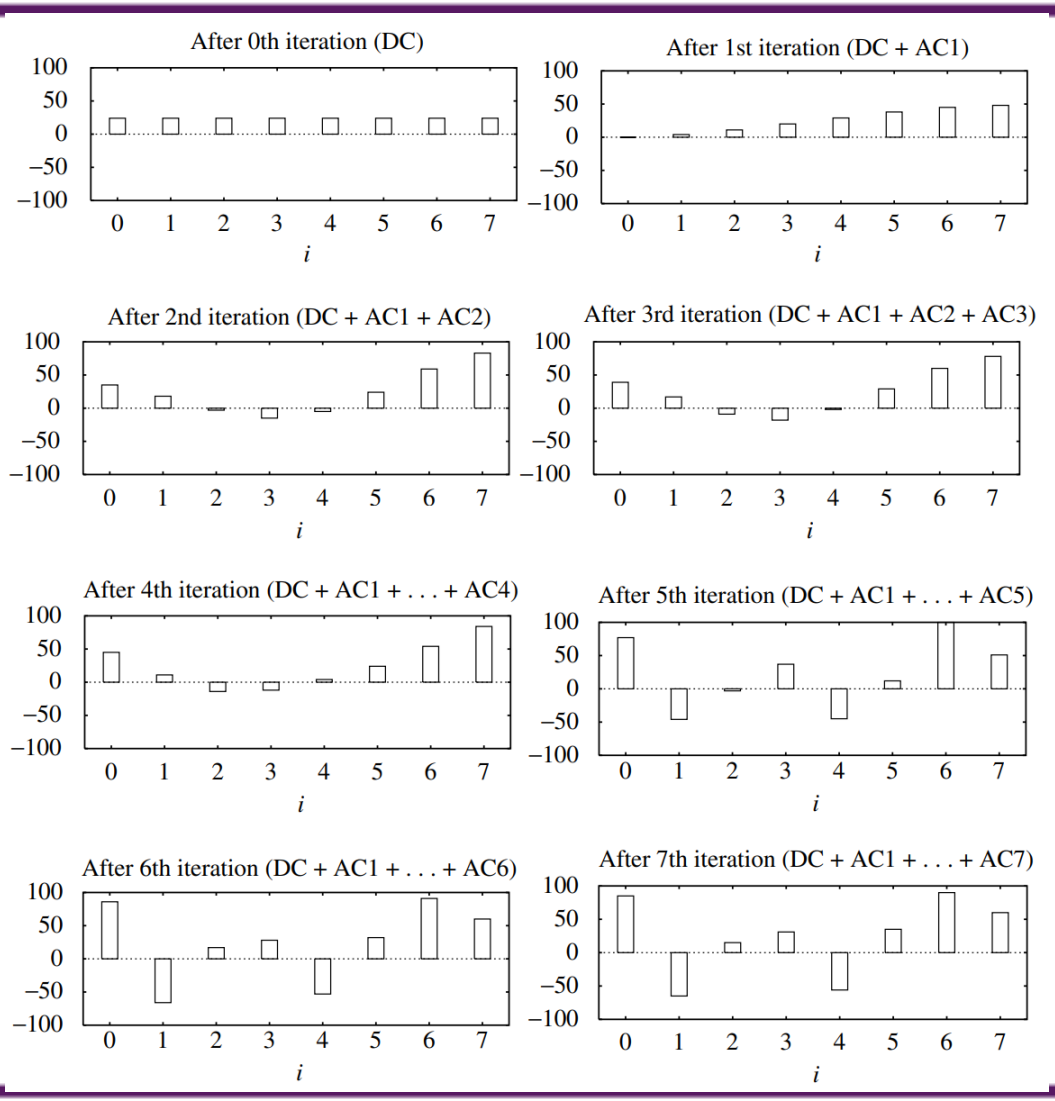
(a) A DC signal $f_1(i)$, **(b)** An AC signal $f_2(i)$.

Examples of 1D DCT



(c) $f_3(i) = f_1(i) + f_2(i)$, and **(d)** an arbitrary signal $f(i)$.

An Example of 1D DCT



DCT is a Linear Transform

- In general, a transform T (or function) is linear, iff

$$\mathcal{T}(\alpha p + \beta q) = \alpha \mathcal{T}(p) + \beta \mathcal{T}(q), \quad (8.21)$$

where α and β are constants, p and q are any functions, variables or constants.

- From the definition of DCT, this property can readily be proven for the DCT because it uses only simple arithmetic operations.

The Cosine Basis Functions

- Function $B_p(i)$ and $B_q(i)$ are *orthogonal*, if

$$\sum_i [B_p(i) \cdot B_q(i)] = 0 \qquad if \quad p \neq q \tag{8.22}$$

- Function $B_p(i)$ and $B_q(i)$ are *orthonormal*, if they are orthogonal and

$$\sum_i [B_p(i) \cdot B_q(i)] = 1 \qquad if \quad p = q \tag{8.23}$$

- It can be shown that:

$$\sum_{i=0}^{7} \left[\cos \frac{(2i+1) \cdot p\pi}{16} \cdot \cos \frac{(2i+1) \cdot q\pi}{16} \right] = 0 \qquad if \quad p \neq q$$

$$\sum_{i=0}^{7} \left[\frac{C(p)}{2} \cos \frac{(2i+1) \cdot p\pi}{16} \cdot \frac{C(q)}{2} \cos \frac{(2i+1) \cdot q\pi}{16} \right] = 1 \qquad if \quad p = q$$

2D Discrete Cosine Transform (DCT)

Definition of 2D DCT:

- Given an input function $f(i, j)$ over two integer variables i and j (a piece of an image), the 2D DCT transforms it into a new function $F(u, v)$, with integer u and v running over the same range as i and j. The general definition of the transform is:

$$F(u,v) = \frac{2\,C(u)\,C(v)}{\sqrt{MN}} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \cos\frac{(2i+1)u\pi}{2M} \cos\frac{(2j+1)v\pi}{2N} f(i,j) \quad (8.15)$$

where $i, u = 0, 1, \ldots, M-1$, $j, v = 0, 1, \ldots, N-1$, and the constants $C(u)$ and $C(v)$ are determined by

$$C(\xi) = \begin{cases} \frac{\sqrt{2}}{2} & \text{if } \xi = 0, \\ 1 & \text{otherwise.} \end{cases} \quad (8.16)$$

2D DCT and IDCT

2D DCT for 8X8 images:

$$F(u,v)=\frac{C(u)C(v)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7}\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}f(i,j) \qquad (8.17)$$

where $i, j, u, v = 0, 1, \ldots, 7$, and the constants $C(u)$ and $C(v)$ are determined by Eq. (8.16).

2D Inverse Discrete Cosine Transform (2D IDCT) for 8X8 images

The inverse function is almost the same, with the roles of $f(i, j)$ and $F(u, v)$ reversed, except that now $C(u)C(v)$ must stand inside the sums:

$$\tilde{f}(i,j)=\sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)C(v)}{4}\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}F(u,v) \qquad (8.18)$$

where $i, j, u, v = 0, 1, \ldots, 7$.

2D Basis Functions

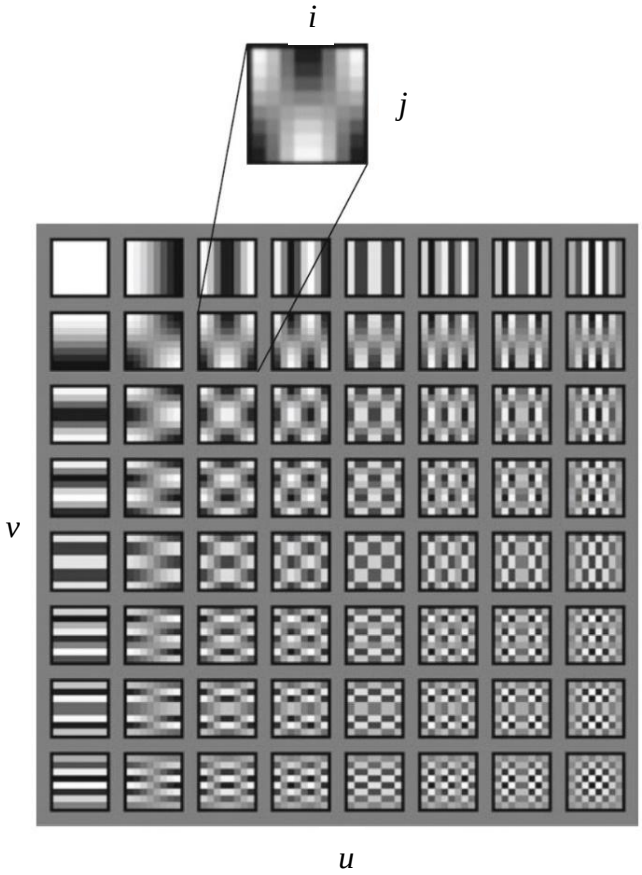
- For a particular pair of u and v, the respective 2D basis function is:

$$\cos\frac{(2i+1)\cdot u\pi}{16}\cdot\cos\frac{(2j+1)\cdot v\pi}{16}, \qquad (8.24)$$

- The enlarged block shown in Fig. 8.9 is for the basis function:

$$\cos\frac{(2i+1)\cdot 1\pi}{16}\cdot\cos\frac{(2j+1)\cdot 2\pi}{16}.$$

2D DCT Basis



- **Fig. 8.9:** Graphical Illustration of 8 × 8 2D DCT basis.

8 × 8 DCT Bases

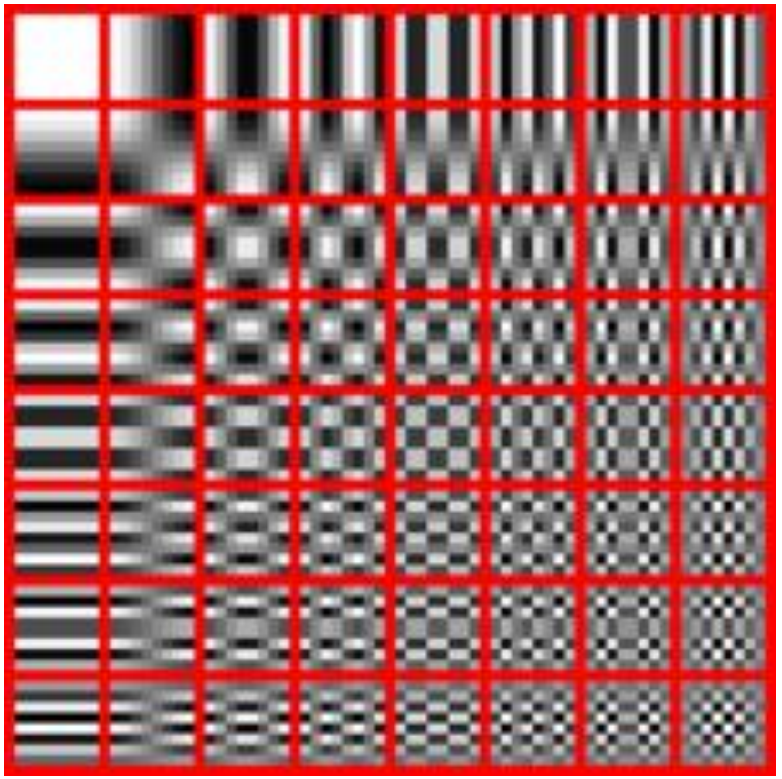
$$F_{u,v} = \sum_{x=0}^{7}\sum_{y=0}^{7} \frac{C(u)\,C(v)}{4} f_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

DCT coefficients

Image

→ *u*

↓ *v*



Inverse DCT

$$f_{x,y} = \sum_{u=0}^{7}\sum_{v=0}^{7} \frac{C(u)\,C(v)}{4} F_{u,v} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

2D Separable Basis

- The 2D DCT can be separated into a sequence of two, 1D DCT steps:

$$G(u, j) = \frac{1}{2}C(u)\sum_{i=0}^{7}\cos\frac{(2i+1)u\pi}{16}f(i, j). \quad (8.25)$$

$$F(u, v) = \frac{1}{2}C(v)\sum_{j=0}^{7}\cos\frac{(2j+1)v\pi}{16}G(u, j). \quad (8.26)$$

- It is straightforward to see that this simple change saves many arithmetic steps. The number of iterations required is reduced from 8 × 8 to 8+8.

2D DCT Matrix Implementation

- The above factorization of a 2D DCT into two 1D DCTs can be implemented by two consecutive matrix multiplications:

$$[\, \boldsymbol{F(u, v)} \,] = \mathbf{T} \;\; [\, \boldsymbol{f(i, j)} \,] \cdot \mathbf{T}^T . \tag{8.27}$$

- We will name T the DCT-matrix.

$$\mathbf{T}[i,\, j] = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } \; i = 0 \\ \sqrt{\frac{2}{N}} \cdot \cos\frac{(2j+1)\cdot i\pi}{2N}, & \text{if } \; i > 0 \end{cases} \tag{8.28}$$

- Where i = 0, … , N-1 and j = 0, … , N-1 are the row and column indices, and the block size is N x N.

- When $N = 8$, we have:

$$\mathbf{T_8}[i, j] = \begin{cases} \frac{1}{2\sqrt{2}}, & \text{if } i = 0 \\ \frac{1}{2} \cdot \cos\frac{(2j+1)\cdot i\pi}{16}, & \text{if } i > 0. \end{cases} \tag{8.29}$$

$$\mathbf{T_8} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \cdots & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cdot\cos\frac{\pi}{16} & \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{5\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{16} \\ \frac{1}{2}\cdot\cos\frac{\pi}{8} & \frac{1}{2}\cdot\cos\frac{3\pi}{8} & \frac{1}{2}\cdot\cos\frac{5\pi}{8} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{8} \\ \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{9\pi}{16} & \frac{1}{2}\cdot\cos\frac{15\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{45\pi}{16} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}\cdot\cos\frac{7\pi}{16} & \frac{1}{2}\cdot\cos\frac{21\pi}{16} & \frac{1}{2}\cdot\cos\frac{35\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{105\pi}{16} \end{bmatrix}. \tag{8.30}$$

2D IDCT Matrix Implementation

- The 2D IDCT matrix implementation is simply:

$$[\, f(i,j) \,] = \mathbf{T}^T \; [\, F(u,v) \,] \; \mathbf{T}. \tag{8.31}$$

- See the textbook for step-by-step derivation of the above equation.
  - The key point is: the DCT-matrix is orthogonal, hence,

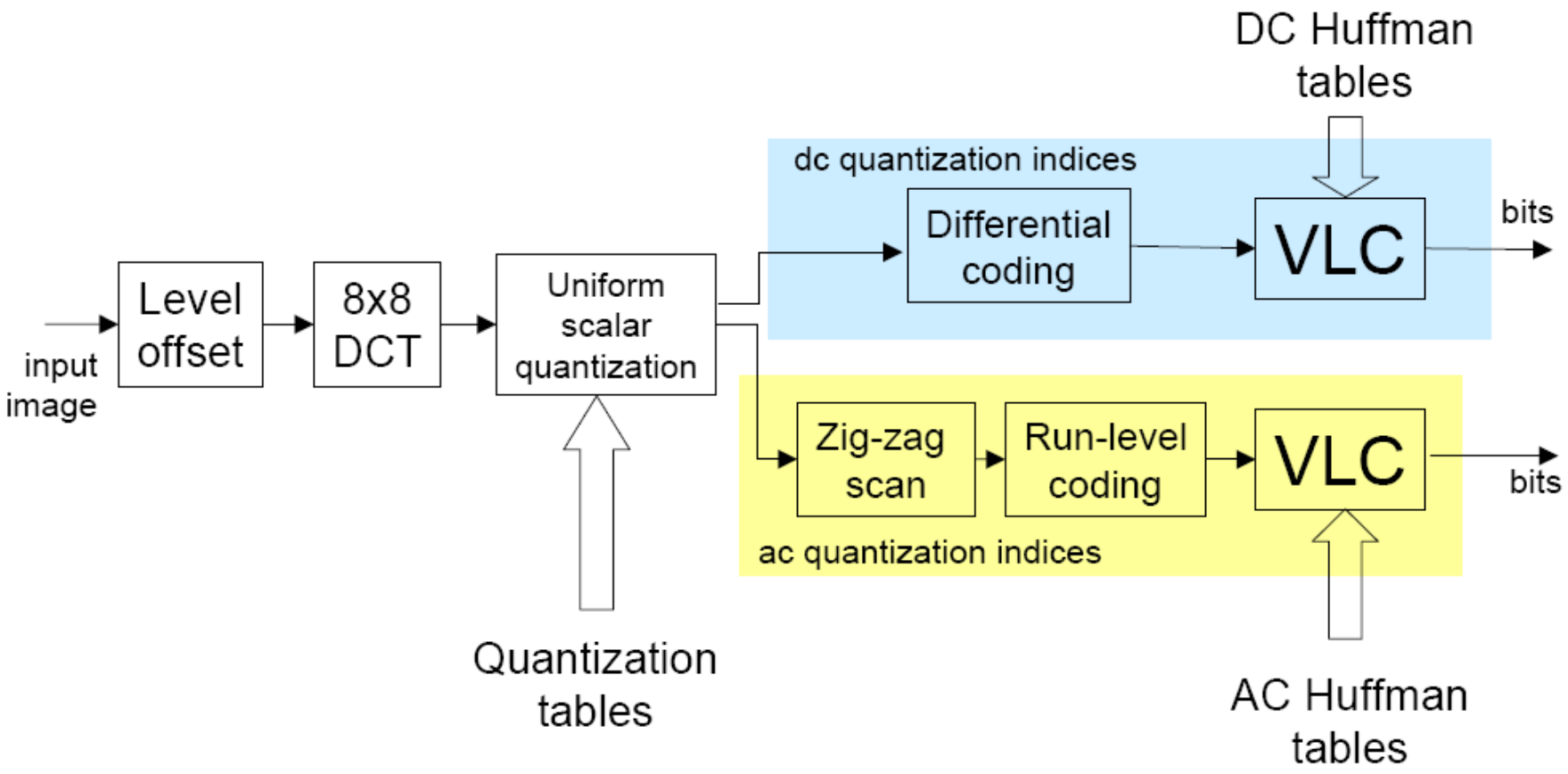
$$\mathbf{T}^T = \mathbf{T}^{-1}.$$

JPEG Compression

- JPEG is an image compression standard that was developed by the "Joint Photographic Experts Group". JPEG was formally accepted as an international standard in 1992.

- JPEG is a **lossy** image compression method. It employs a **transform coding** method using the DCT (*Discrete Cosine Transform*).
- Image processing programs allow you to choose the JPEG compression rate

JPEG Image Compression



Observations for JPEG Image Compression

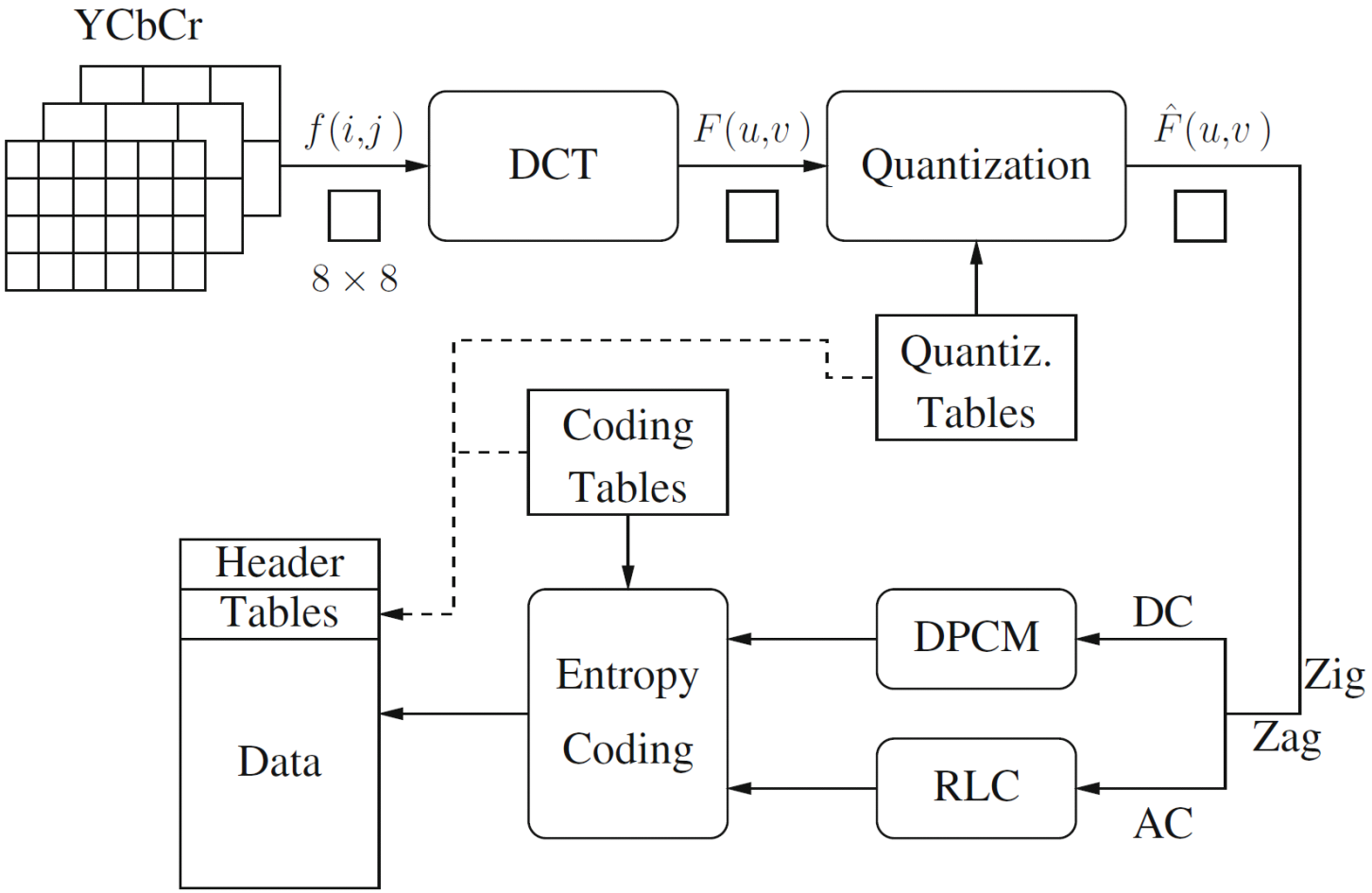
- The effectiveness of the DCT transform coding method in JPEG relies on 3 major observations:
- **Observation 1**: Useful image contents change relatively slowly across the image, i.e., it is unusual for intensity values to vary widely several times in a small area, for example, within an 8×8 image block.
- Much of the information in an image is repeated, hence "spatial redundancy".

Observations for JPEG Image Compression

- **Observation 2**: Psychophysical experiments suggest that humans are much less likely to notice the loss of very high spatial frequency components than the loss of lower frequency components.
  - The spatial redundancy can be explored by largely reducing the high spatial frequency contents.

- **Observation 3**: Visual acuity (accuracy in distinguishing closely spaced lines) is much greater for gray ("black and white") than for color.
  - Chroma subsampling (4:2:0) is used in JPEG.

Block Diagram for JPEG Encoder



Main Steps in JPEG Compression

- Transform RGB to YCbCr and subsample color
- DCT on image blocks
- Quantization
- Zigzag ordering and run-length encoding
- Entropy coding

JPEG Compression – step1

- **Step1 : Divide the image into 8 × 8 pixel blocks and convert RGB to a luminance/chrominance color model**
- The image is divided into 8 × 8 pixel blocks to make it computationally more manageable for the next steps.
- Converting color to a luminance/chrominance model makes it possible to remove some of the chrominance information, to which the human eye is less sensitive, without significant loss of quality in the image.

RGB → YC_bC_r

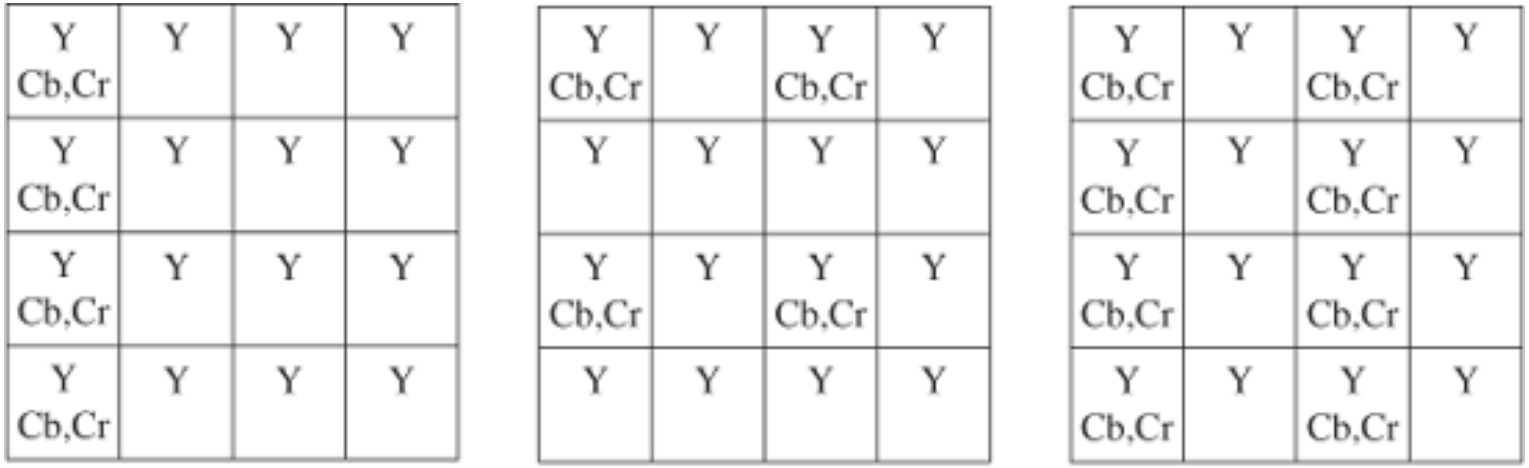
- YCbCr color model represents color in terms of one luminance component, Y, and two chrominance components, Cb and Cr.
- The human eye is more sensitive to changes in light (*i.e.*, luminance) than in color (*i.e.*, chrominance).

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ 0.148 & 0.291 & 0.439 \\ 0.128 & 0.439 & 0.368 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

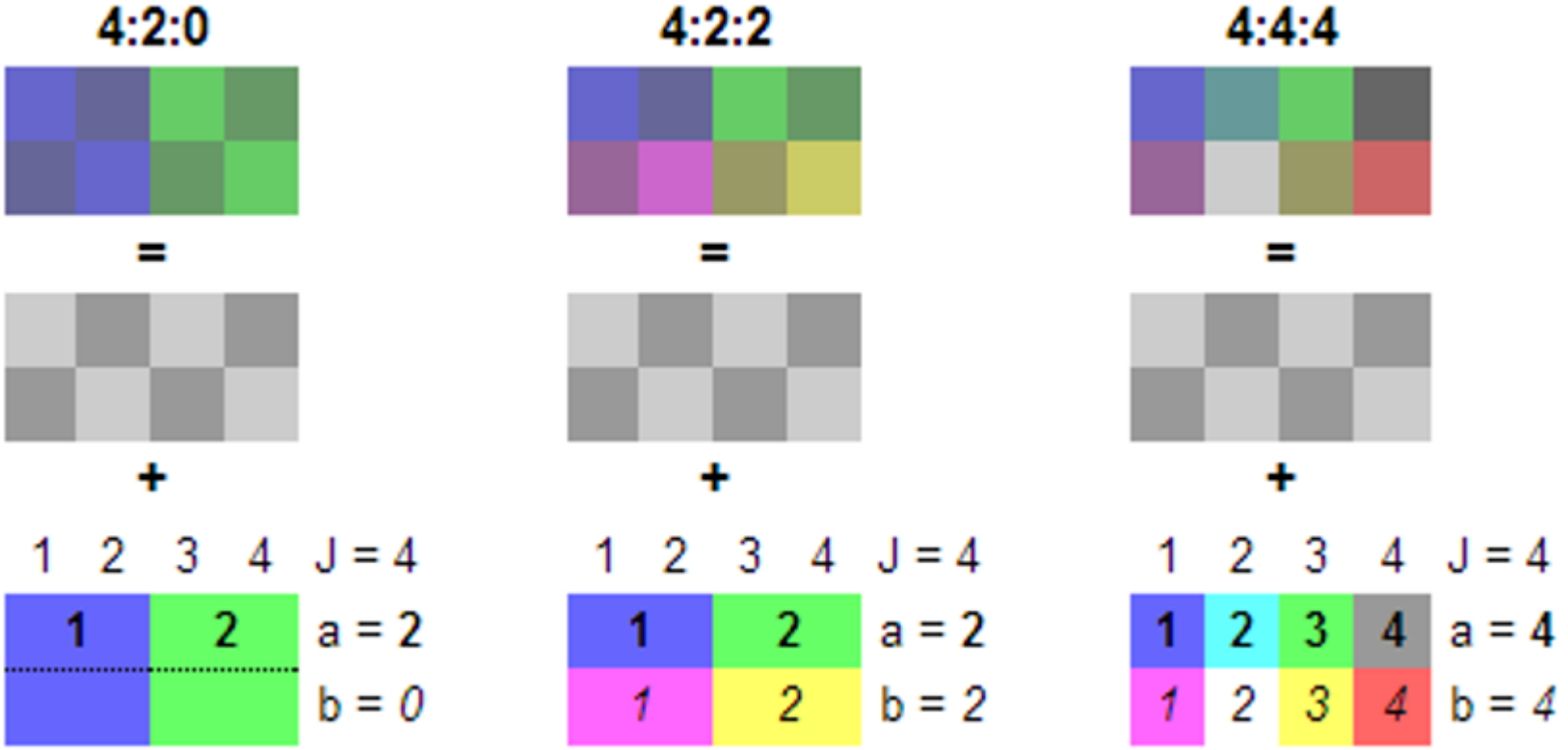
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & 0.392 & 0.439 \\ 1.164 & 2.017 & 0 \end{bmatrix} \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Chrominance Subsampling

- luminance/chrominance subsampling is represented in the form *a*:*b*:*c*
- For each pair of four-pixel-wide rows, *a* is the number of Y samples in both rows, *b* and c are the numbers of C_b (C_r) samples in the 1st and 2nd rows, respectively.



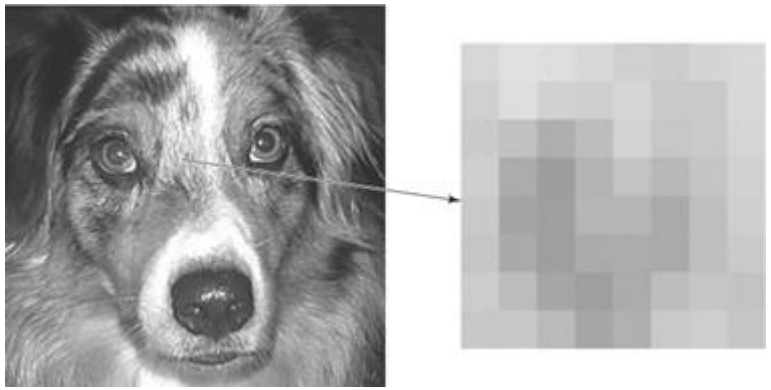
Chrominance Subsampling



JPEG Compression – step2

- **Step 2: Shift values by –128 and transform each block from the spatial to the DCT domain**
- On an intuitive level, shifting the values by –128 is like looking at the image function as a waveform that cycles through positive and negative values.
- This step is a preparation for representing the function in terms of its DCT (frequency) components. Transforming from the spatial to the DCT domain makes it possible to remove high frequency components.
- High frequency components are present if color values go up and down quickly. Small changes are barely perceptible for human vision, so removing them does not compromise image quality significantly.

JPEG Compression – step2



Grayscale Values for 8 × 8 Pixel Area							
222	231	229	224	216	213	220	224
216	229	217	215	221	210	209	223
211	202	283	198	218	207	209	221
214	180	164	188	203	193	205	217
209	171	166	190	190	178	199	215
206	177	166	179	180	178	199	210
212	197	173	166	179	198	206	203
208	208	195	174	184	210	214	206

→

Pixel Values for Image in Figure 3.49 Shifted by –128							
94	103	101	96	88	85	92	96
88	101	89	87	93	82	81	95
83	74	55	70	90	79	81	93
86	52	36	60	75	65	77	89
81	43	38	62	62	50	71	87
78	49	38	51	52	50	71	82
84	69	45	38	51	70	78	75
80	80	67	46	56	82	86	78

JPEG Compression – step2

- Take the two-dimensional DCT step.

DCT of an 8 × 8 Pixel Area							
585.7500	–24.5397	59.5959	21.0853	25.7500	–2.2393	–8.9907	1.8239
78.1982	12.4534	–32.6034	–19.4953	10.7193	–10.5910	–5.1086	–0.5523
57.1373	24.829	–7.5355	–13.3367	–45.0612	–10.0027	4.9142	–2.4993
–11.8655	6.9798	3.8993	–14.4061	8.5967	12.9151	–0.3122	–0.1844
5.2500	–1.7212	–1.0824	–3.2106	1.2500	9.3595	2.6131	1.1199
–5.9658	–4.0865	7.6451	13.0616	–1.1927	1.1782	–1.0733	–0.5631
–1.2074	–5.7729	–2.0858	–1.9347	1.6173	2.6671	–0.4645	0.6144
0.6362	–1.4059	–0.719	1.6339	–0.1438	0.2755	–0.0268	–0.2255

JPEG Compression – step3

- **Step 3: Quantize the frequency values**
- Quantization involves dividing each frequency coefficient by an integer and rounding off. The coefficients for high-frequency components are typically small, so they often round down to 0—which means, in effect, that they are thrown away.

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0, 1, 2, \ldots, 7; k = 0, 1, 2, \ldots, 7$$

G is the unquantized DCT coefficients ; Q is the quantization matrix and B is the quantized DCT coefficients

JPEG Compression – step3

Quantization Table for example							
8	6	6	7	6	5	8	7
7	7	9	9	8	10	12	20
13	12	11	11	12	25	18	19
15	20	29	26	31	30	29	26
28	28	32	36	46	39	32	34
44	35	28	28	40	55	41	44
48	49	52	52	52	31	39	57
61	56	50	60	46	51	52	50

Quantized DCT Values							
73	–4	10	3	4	0	–1	0
11	2	–4	–2	1	–1	0	0
4	2	–1	–1	–4	0	0	0
–1	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	0	0	0	0	0

Standard JPEG Quantization Tables

- Luminance

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	36	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

- Chrominance

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

JPEG compression for a smooth image block

An 8 × 8 block from the Y image of 'Lena'

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

f(i, j)

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

F(u, v)

JPEG compression for a smooth image block (Cont'd)

32	6	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
-1	0	1	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
0	0	0	0	0	0	0	0

$\hat{F}(u,v)$

512	66	-10	0	0	0	0	0
-12	0	0	0	0	0	0	0
-14	0	16	0	0	0	0	0
-14	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
0	0	0	0	0	0	0	0

$\tilde{F}(u,v)$

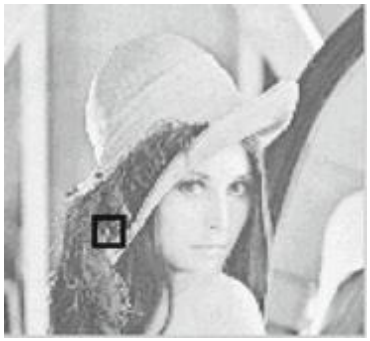
199	196	191	186	182	178	177	176
201	199	196	192	188	183	180	178
203	203	202	200	195	189	183	180
202	203	204	203	198	191	183	179
200	201	202	201	196	189	182	177
200	200	199	197	192	186	181	177
204	202	199	195	190	186	183	181
207	204	200	194	190	187	185	184

$\tilde{f}(i,j)$

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

$\epsilon(i,j) = f(i,j) - \tilde{f}(i,j)$

JPEG compression for a textured image block



Another 8 × 8 block from the Y image of 'Lena'

70	70	100	70	87	87	150	187
85	100	96	79	87	154	87	113
100	85	116	79	70	87	86	196
136	69	87	200	79	71	117	96
161	70	87	200	103	71	96	113
161	123	147	133	113	113	85	161
146	147	175	100	103	103	163	187
156	146	189	70	113	161	163	197

f(i, j)

-80	-40	89	-73	44	32	53	-3
-135	-59	-26	6	14	-3	-13	-28
47	-76	66	-3	-108	-78	33	59
-2	10	-18	0	33	11	-21	1
-1	-9	-22	8	32	65	-36	-1
5	-20	28	-46	3	24	-30	24
6	-20	37	-28	12	-35	33	17
-5	-23	33	-30	17	-5	-4	20

F(u, v)

JPEG compression for a textured image block (Cont'd)

-5	-4	9	-5	2	1	1	0
-11	-5	-2	0	1	0	0	-1
3	-6	4	0	-3	-1	0	1
0	1	-1	0	1	0	0	0
0	0	-1	0	0	1	0	0
0	-1	1	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\hat{F}(u,v)$

-80	-44	90	-80	48	40	51	0
-132	-60	-28	0	26	0	0	-55
42	-78	64	0	-120	-57	0	56
0	17	-22	0	51	0	0	0
0	0	-37	0	0	109	0	0
0	-35	55	-64	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\tilde{F}(u,v)$

70	60	106	94	62	103	146	176
85	101	85	75	102	127	93	144
98	99	92	102	74	98	89	167
132	53	111	180	55	70	106	145
173	57	114	207	111	89	84	90
164	123	131	135	133	92	85	162
141	159	169	73	106	101	149	224
150	141	195	79	107	147	210	153

$\tilde{f}(i,j)$

0	10	-6	-24	25	-16	4	11
0	-1	11	4	-15	27	-6	-31
2	-14	24	-23	-4	-11	-3	29
4	16	-24	20	24	1	11	-49
-12	13	-27	-7	-8	-18	12	23
-3	0	16	-2	-20	21	0	-1
5	-12	6	27	-3	2	14	-37
6	5	-6	-9	6	14	-47	44

$\epsilon(i,j) = f(i,j) - \tilde{f}(i,j)$

Quantization Tables - Quality Factor

- Quantization tables can be scaled to a **quality factor Q_f**. The quality factor allows the image creation device to choose between larger, higher quality images and smaller, lower quality images.
- The value of Q_f can range between 1 and 100 and is used to compute the **scaling factor, S**.

$$S = (Q_f < 50)? \frac{5000}{Q_f} : 200\text{-}2Q_f$$

Quantization Tables - Quality Factor

- Each element i in the scaled table T_s is computed using the i^{th} element in the base table T_b.

$$T_s[i] = \left\lfloor \frac{S * T_b[i] + 50}{100} \right\rfloor$$

Ex:

$Q_f = 80 \Rightarrow S = 40$

$6 = \lfloor (40 * 16 + 50)/100 \rfloor$

- Luminance

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Standard JPEG quantization table

6	4	4	6	10	16	20	24
5	5	6	8	10	23	24	22
6	5	6	10	16	23	28	22
6	7	9	12	20	35	32	25
7	9	15	22	27	44	41	31
10	14	22	26	32	42	45	37
20	26	31	35	41	48	48	40
29	37	38	39	45	40	41	40

Standard JPEG quantization table scaled with **Q_f = 80**

Difference of Quality factor

Quality = 100

Quality = 10

Quality = 50

Quality = 1

JPEG Compression – step4

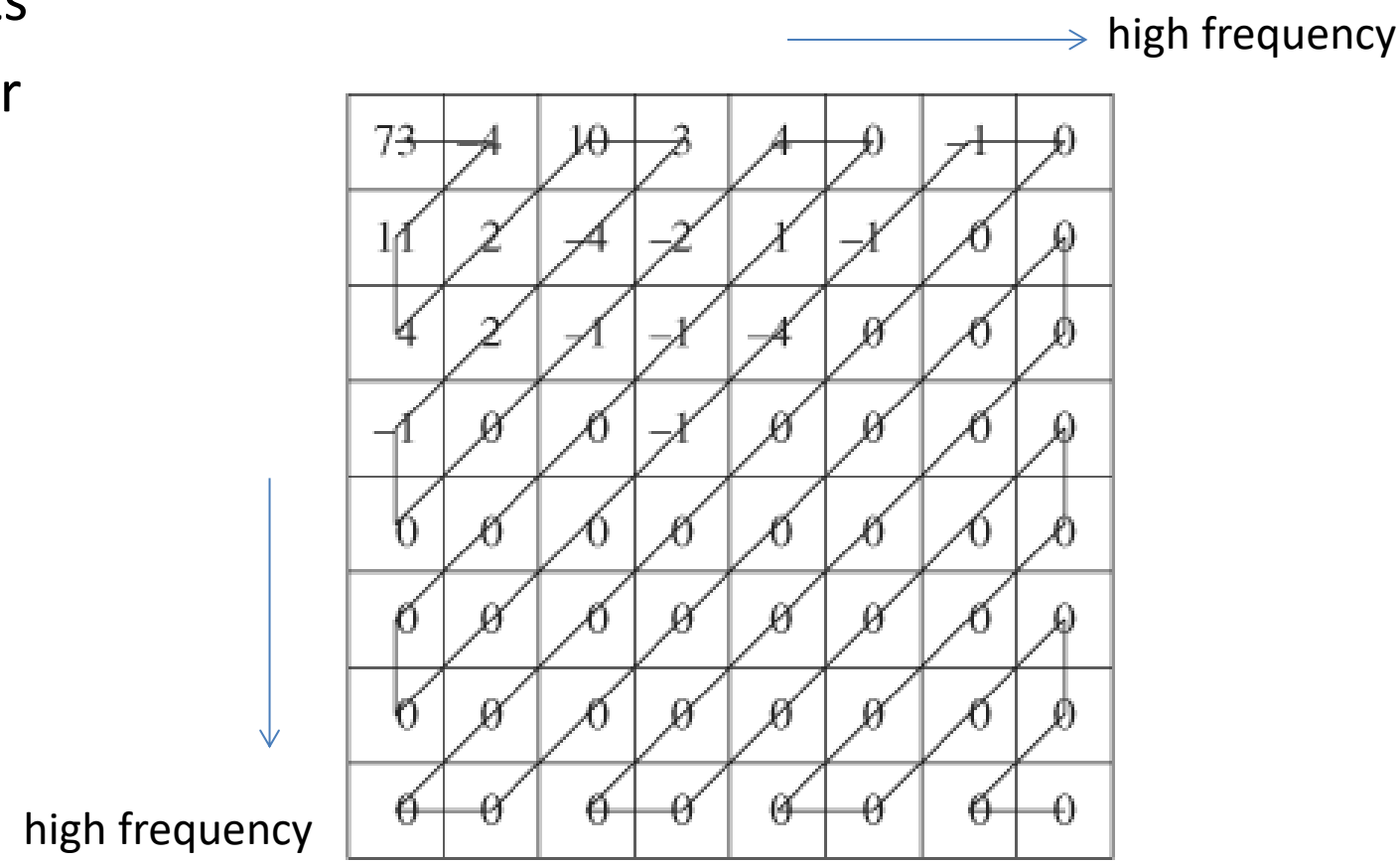
- **Step 4: Apply DPCM to the DC coefficients**
- ***DPCM*** is the abbreviation for ***differential pulse code modulation***. In this context, DCPM is simply storing the difference between the first value in the previous 8×8 block and the first value in the current block. Since the difference is generally smaller than the actual value, this step adds to the compression.
- If the DC coefficients for the first 5 image blocks are 150, 155, 149, 152, 144, then the DPCM would produce 150, 5, -6, 3, -8, assuming $d_i = DC_{i+1} - DC_i$, and $d_0 = DC_0$.

JPEG Compression – step5

- **Step 5: Arrange the AC part of quantized DCT coefficients in a zigzag order and do run-length coding (RLC).**
- The zigzag reordering sorts the values from low-frequency to high-frequency components. The high-frequency coefficients are grouped together at the end. If many of them round to zero after quantization, run-length coding is even more effective.
- To make it most likely to hit a long run of zeros: a *zig-zag scan* is used to turn the 8×8 matrix $\hat{F}(u,v)$ into a *64-vector*.
- RLC aims to turn the $\hat{F}(u,v)$ values into sets *{#-zeros-to-skip, next non-zero value*}.

JPEG Compression – step5

- Quantized DCT values rearranged from low- to high-frequency components
- Zigzag order



JPEG Compression – step6

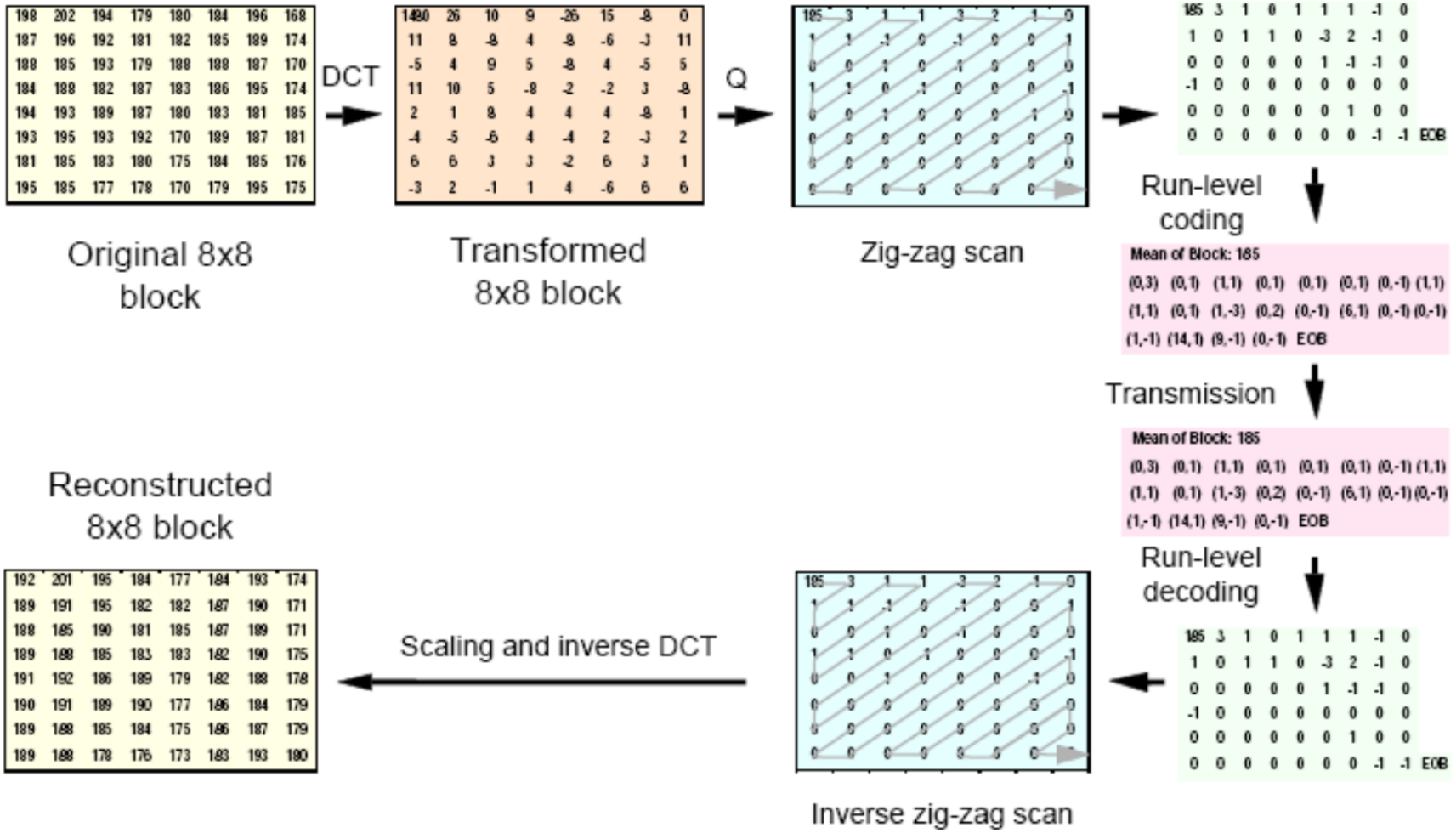
- **Step 6: Do entropy encoding.**
- The DC and AC coefficients finally undergo an entropy coding step to gain a possible further compression.
- Use DC as an example: each DPCM coded DC coefficient is represented by (SIZE, AMPLITUDE), where SIZE indicates how many bits are needed for representing the coefficient, and AMPLITUDE contains the actual bits.
- In the example we're using, codes 150, 5, −6, 3, −8 will be turned into

  (8, 10010110), (3, 101), (3, 001), (2, 11), (4, 0111) .
- SIZE is Huffman coded since smaller SIZEs occur much more often. AMPLITUDE is not Huffman coded, its value can change widely so Huffman coding has no appreciable benefit.

JPEG Compression Algorithm

algorithm jpeg
/*Input: A bitmap image in RGB mode.
Output: The same image, compressed.*/
{
  Divide image into 8 × 8 pixel blocks
  Convert image to a luminance/chrominance model such as YCbCr (optional)
  Shift pixel values by subtracting 128
  Use discrete cosine transform to transform the pixel data from the spatial domain to the frequency domain
  Quantize frequency values
  Store DC value (upper left corner) as the difference between current DC value and DC from previous block
  Arrange the block in a zigzag order
  Do run-length encoding
  Do entropy encoding (e.g., Huffman)
}

Transform Coding



Four Commonly Used JPEG Modes

- Sequential Mode — the default JPEG mode, implicitly assumed in the discussions so far. Each graylevel image or color image component is encoded in a single left-to-right, top-to-bottom scan.
- Progressive Mode.
- Hierarchical Mode.
- Lossless Mode — rarely used due to its low compression ratio, replaced by JPEG-LS

Progressive Mode

- Progressive JPEG delivers low quality versions of the image quickly, followed by higher quality passes.

1. **Spectral selection**: Takes advantage of the "spectral" (spatial frequency spectrum) characteristics of the DCT coefficients: higher AC components provide detail information.

   - Scan 1: Encode DC and first few AC components, e.g., AC1, AC2.
   - Scan 2: Encode a few more AC components, e.g., AC3, AC4, AC5.
   - ...
   - Scan k: Encode the last few ACs, e.g., AC61, AC62, AC63.

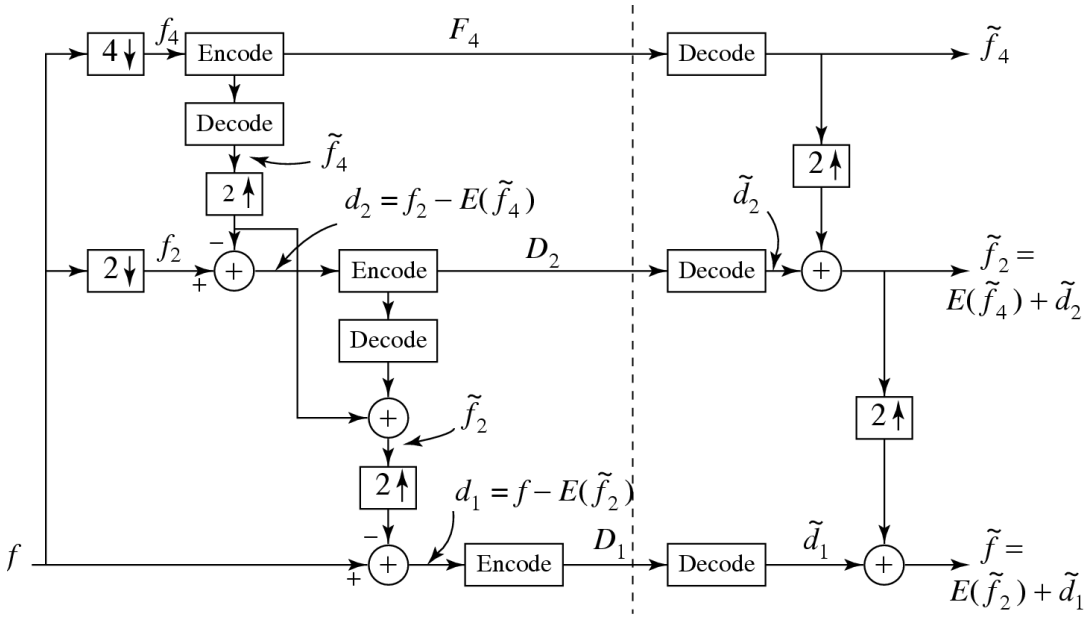
Progressive Mode (Cont'd)

2. **Successive approximation**: Instead of gradually encoding spectral bands, all DCT coefficients are encoded simultaneously but with their most significant bits (MSBs) first.

   - Scan 1: Encode the first few MSBs, e.g., Bits 7, 6, 5, 4.
   - Scan 2: Encode a few more less significant bits, e.g., Bit 3.
   - ...
   - Scan m: Encode the least significant bit (LSB), Bit 0.

Hierarchical Mode

- The encoded image at the lowest resolution is basically a compressed low-pass filtered image, whereas the images at successively higher resolutions provide additional details.
- Similar to Progressive JPEG, the Hierarchical JPEG images can be transmitted in multiple passes progressively improving quality.



Algorithm 9.1: Three-Level Hierarchical JPEG Encoder

1. **Reduction of image resolution.** Reduce resolution of the input image f (e.g., 512×512) by a factor of 2 in each dimension to obtain f_2 (e.g., 256×256). Repeat this to obtain f_4 (e.g., 128×128).

2. **Compress low-resolution image f_4.** Encode f_4 using any other JPEG method (e.g., Sequential, Progressive) to obtain F_4.

3. **Compress difference image d_2.**

   (a) Decode F_4 to obtain $\tilde{f}_4$. Use any interpolation method to expand $\tilde{f}_4$ to be of the same resolution as f_2 and call it $E(\tilde{f}_4)$.

   (b) Encode difference $d_2 = f_2 - E(\tilde{f}_4)$ using any other JPEG method (e.g., Sequential, Progressive) to generate D_2.

4. **Compress difference image d_1.**

   (a) Decode D_2 to obtain $\tilde{d}_2$; add it to $E(\tilde{f}_4)$ to get $\tilde{f}_2 = E(\tilde{f}_4) + \tilde{d}_2$, which is a version of f_2 after compression and decompression.

   (b) Encode difference $d_1 = f - E(\tilde{f}_2)$ using any other JPEG method (e.g., Sequential, Progressive) to generate D_1.

Algorithm 9.2: Three-Level Hierarchical JPEG Decoder

1. **Decompress the encoded low-resolution image F_4.** Decode F_4 using the same JPEG method as in the encoder, to obtain $\tilde{f}_4$.
2. **Restore image $\tilde{f}_2$ at the intermediate resolution.** Use $E(\tilde{f}_4) + \tilde{d}_2$ to obtain $\tilde{f}_2$.
3. **Restore image $\tilde{f}$ at the original resolution.** Use $E(\tilde{f}_2) + \tilde{d}_1$ to obtain $\tilde{f}$.

Summary

- JPEG Image Compression
- JPEG2000
  - better rate-distortion trade-off, improved subjective image quality, and additional functionalities, including large image size handling, error resilience, ROI coding, handling multi-spectral images up to 256 channels, compound document, etc.
  - JPEG 2000 is mostly used in high-end applications such as remote sensing satellite images, medical imaging, and digital cinema, where very high-resolution and high-quality images are required.
- JPEG-LS
  - low-complexity lossless image compression standard