12.11.2021

Digital Image Processing (CSE/ECE 478)

Lecture-21: Image Compression (contd.)

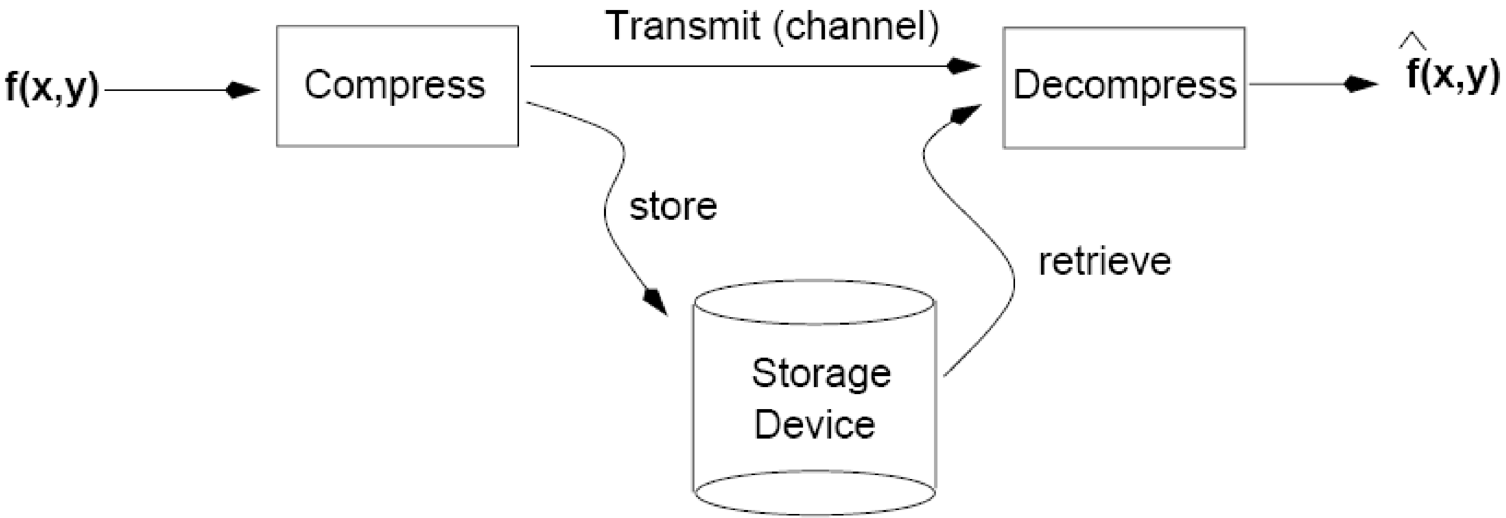
Ravi Kiran and Sudipta Banerjee

Center for Visual Information Technology (CVIT), IIIT Hyderabad

Some slides borrowed from Vineet Gandhi @CVIT!

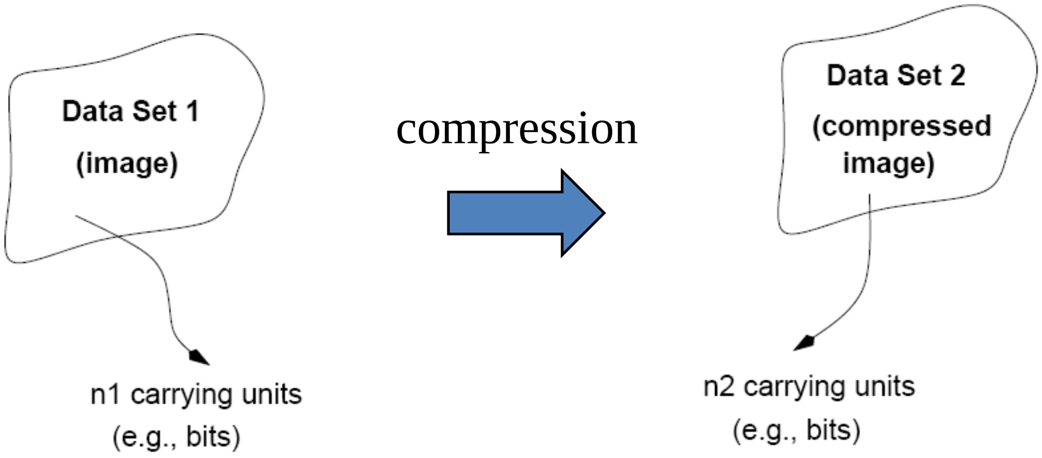
Image Compression

- Goal: Reduce amount of data required to represent a digital <u>image</u>



.. By leveraging redundancies in image data

Compression Ratio



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

Relevant Data Redundancy

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

Example:

If $C_R = \frac{10}{1}$, then $R_D = 1 - \frac{1}{10} = 0.9$

(90% of the data in dataset 1 is redundant)

if $n_2 = n_1$, then C_R=1, R_D=0

if $n_2 \ll n_1$, then $C_R \to \infty$, $R_D \to 1$

Types of Redundancy

(1) Coding Redundancy

(2) Spatial/Temporal Redundancy

(3) Psychovisual Redundancy

- Image compression attempts to reduce one or more of these redundancy types.

Types of Redundancy

(1) Coding Redundancy
(2) Spatial/Temporal Redundancy
(3) Psychovisual Redundancy

- Image compression attempts to reduce one or more of these redundancy types.

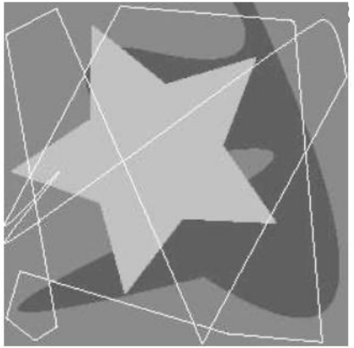
Coding - Definitions

- Code: a list of symbols (letters, numbers, bits etc.)
- Code word: a sequence of symbols used to represent some information (e.g., gray levels).
- Code word length: number of symbols in a code word.

Example: (binary code, symbols: 0,1, length: 3)

0: 000	4: 100
1: 001	5: 101
2: 010	6: 110
3: 011	7: 111

Coding Redundancy



- <u>Case 1:</u> $l(r_k)$ = constant length

Example:

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

Average # of bits: $L_{avg} = E(l(r_k)) = \sum_{k=0}^{L-1} l(r_k)P(r_k)$

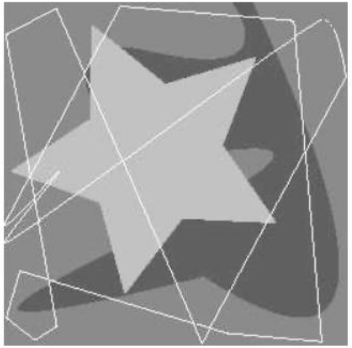
Total # of bits: NML_{avg}

Assume an image with $L = 8$

Assume $l(r_k) = 3$, $L_{avg} = \sum_{k=0}^{7} 3P(r_k) = 3\sum_{k=0}^{7} P(r_k) = 3$ bits

Total number of bits: $3NM$

Coding Redundancy (cont'd)



- <u>Case 2:</u> $l(r_k)$ = variable length

Table 6.1 Variable-Length Coding Example variable length

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

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

$$C_R = \frac{3}{2.7} = 1.11 \text{ (about 10\%)}$$

$$R_D = 1 - \frac{1}{1.11} = 0.099$$

$$L_{avg} = \sum_{k=0}^{7} l(r_k)P(r_k) = 2.7 \text{ bits}$$

Total number of bits: 2.7NM

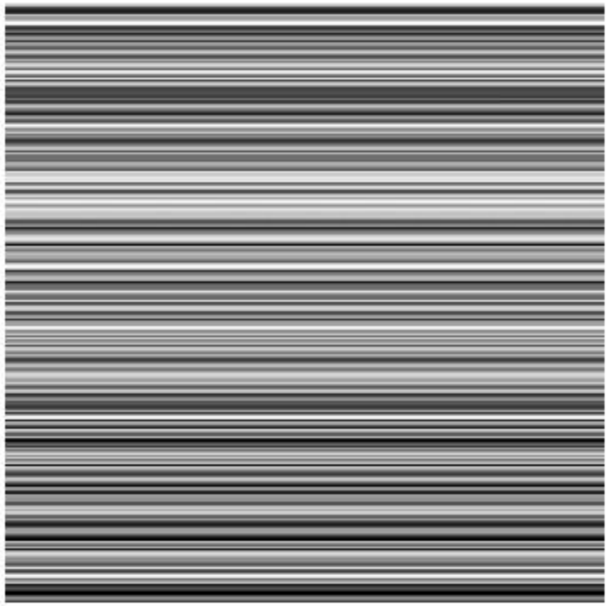
Coding redundancy occurs because least probable and most probable intensity values are represented using same length code. Most images do not have uniform intensity

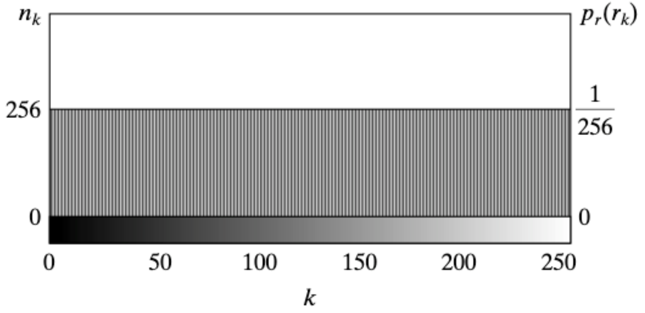
Types of Redundancy

(1) Coding Redundancy
(2) Spatial/Temporal Redundancy
(3) Psychovisual Redundancy

- Image compression attempts to reduce one or more of these redundancy types.

Spatial Redundancy

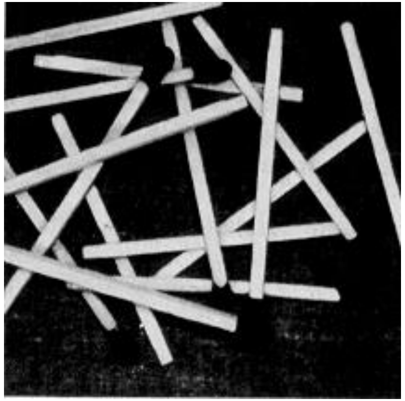


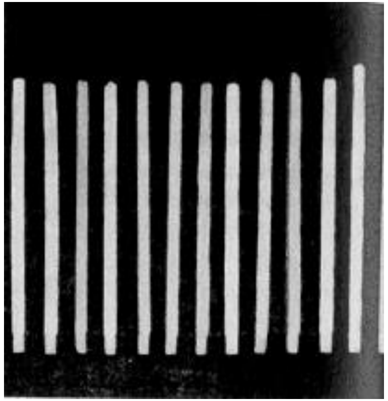


Run-length coding can be used to account for spatial redundancies

Spatial redundancy

- Interpixel redundancy exists ➔ pixel values are correlated
- i.e., a pixel value can be reasonably predicted by its neighbors





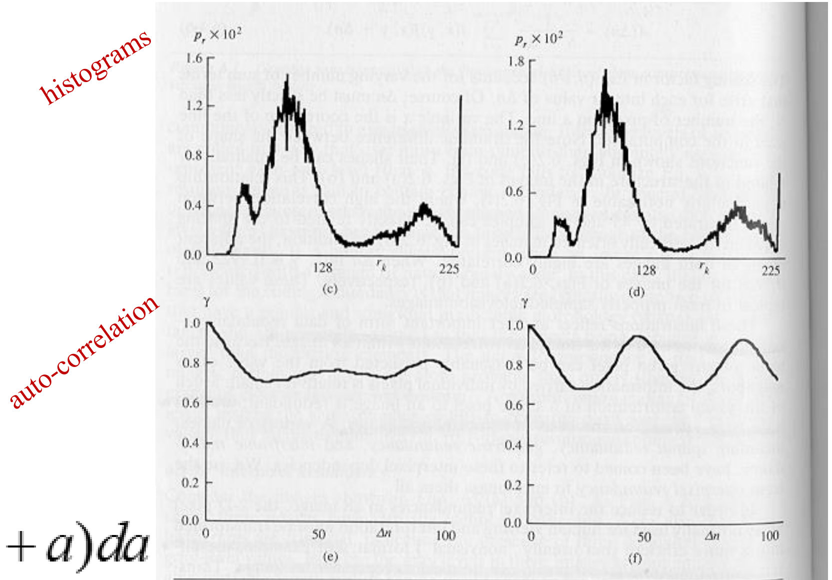


Figure 6.2 Two images and their gray-level histograms and normalized autocorrelation

$$f(x) \, o \, g(x) = \int_{-\infty}^{\infty} f(x) g(x+a) da$$

auto-correlation: f(x)=g(x)

Spatial and temporal redundancy

Types of Redundancy

(1) Coding Redundancy

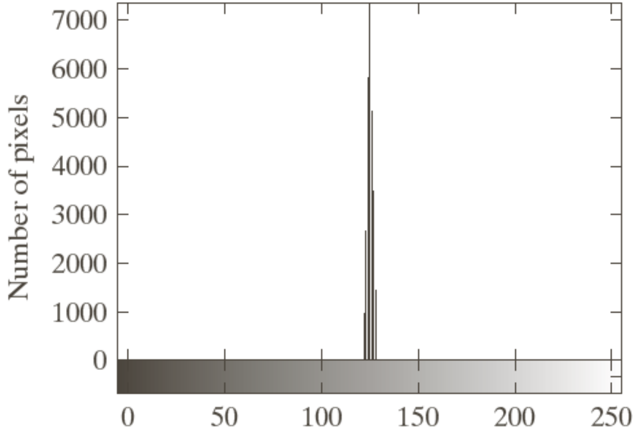
(2) Spatial/Temporal Redundancy

(3) Psychovisual Redundancy

- Image compression attempts to reduce one or more of these redundancy types.

Irrelevant information or perceptual redundancy

- Not all visual information is perceived by eye/brain, so throw away those that are not



Psychovisual redundancy (cont'd)

Example: quantization

256 gray levels

16 gray levels

16 gray levels + random noise

C=8/4 = 2:1

add a small pseudo-random number
to each pixel prior to quantization

Information theory

- Basic Premise: Generation of information can be treated as a probabilistic process defined over symbols.

- Symbol - carrier of information
- Consider a symbol with an occurrence probability p.
- The amount of information contained in the symbol is defined as:

$$I = \log_2 \frac{1}{p} \text{ bits} \quad \text{or} \quad I = -\log_2 p$$

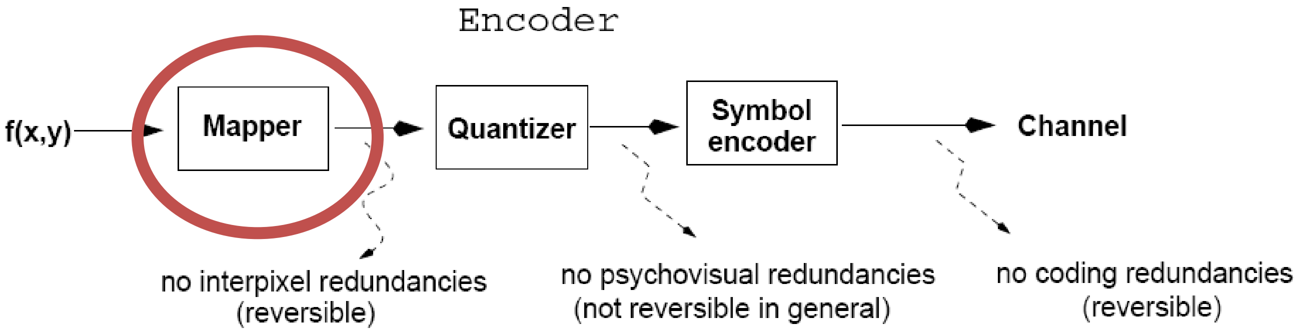
Information theory: Shannon’s theorem

- Shannon’s noiseless source coding theorem: For a discrete, memoryless, information source, the minimum bit rate required to encode a symbol on average is equal to the entropy of the source – **provides a lower bound on the bit rate for encoding statistically independent symbols**

- In other words: we can’t do better than the entropy

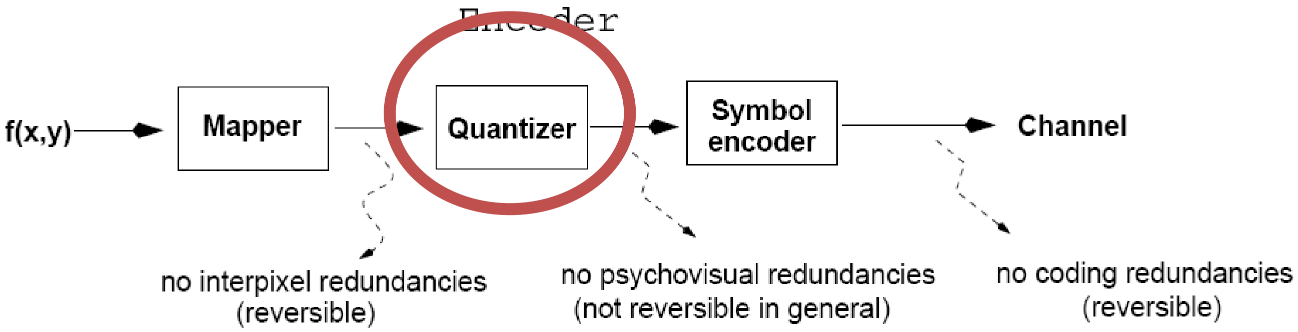
- For a memory information source (finite Markov), the bound might not hold

Image Compression Model



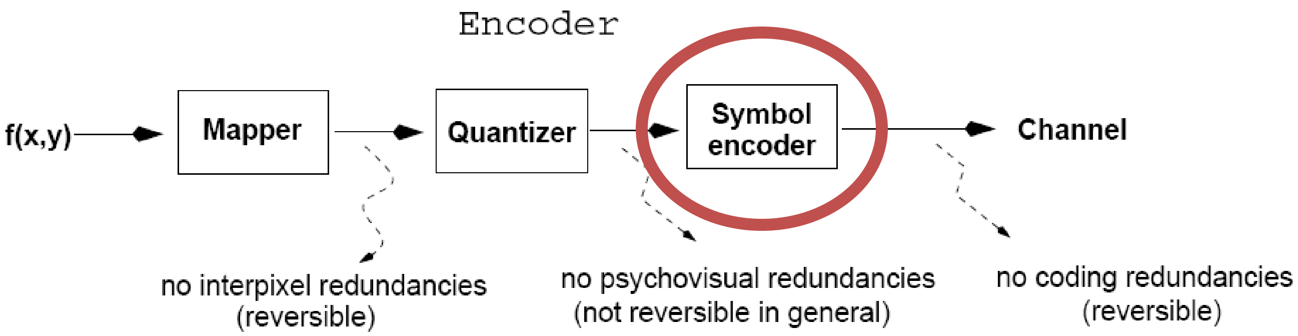
- **Mapper:** transforms data to account for interpixel redundancies.

Image Compression Model (cont'd)



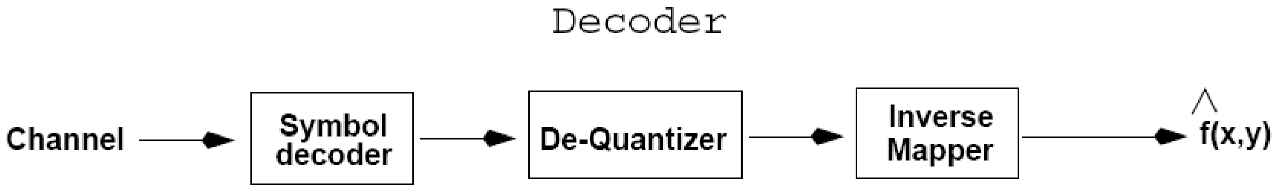
- **Quantizer:** quantizes the data to account for psychovisual redundancies.

Image Compression Model (cont'd)



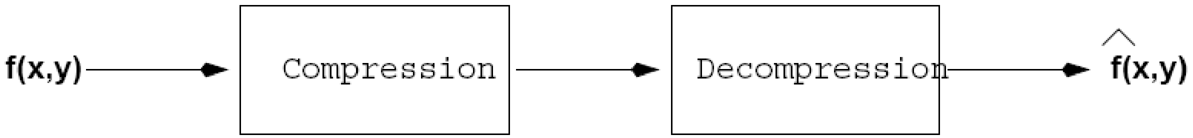
- **Symbol encoder:** encodes the data to account for coding redundancies.

Image Compression Models (cont'd)



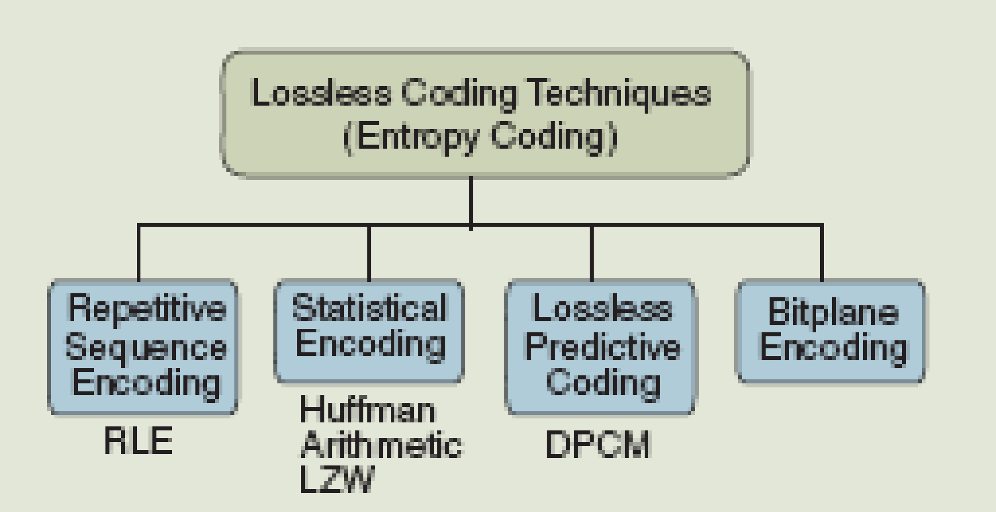
- The decoder applies the inverse steps.
- Note that quantization is irreversible in general.

Lossless Compression

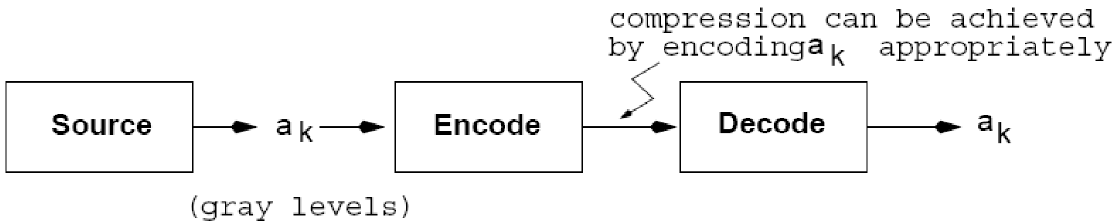


$$e(x,y) = \hat{f}(x,y) - f(x,y) = 0$$

Taxonomy of Lossless Methods



Huffman Coding
(addresses coding redundancy)



- A variable-length coding technique.
- Source symbols are encoded one at a time!
  - There is a one-to-one correspondence between source symbols and code words.
- Optimal code - minimizes code word length per source symbol.

Huffman Coding (cont'd)

- *Forward Pass*
  1. Sort probabilities per symbol
  2. Combine the lowest two probabilities
  3. Repeat *Step2* until only two probabilities remain.

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				

$$\underbrace{0\ 1\ 0\ 1\ 0}_{a_3}\ \underbrace{0\ 1\ 1}_{a_1}\ \underbrace{1}_{a_2}\ \underbrace{1}_{a_2}\ \underbrace{0\ 0}_{a_6}$$

- *Backward Pass*

  Assign code symbols going backwards

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								

Huffman Coding (cont'd)

- L_{avg} assuming Huffman coding:

$$L_{avg} = E(l(a_k)) = \sum_{k=1}^{6} l(a_k)P(a_k) =$$

3x0.1 + 1x0.4 + 5x0.06 + 4x0.1 + 5x0.04 + 2x0.3=2.2 bits/symbol

- 6 symbols, we need a 3-bit code

$(a_1\text{: }000,\ a_2\text{: }001,\ a_3\text{: }010,\ a_4\text{: }011,\ a_5\text{: }100,\ a_6\text{: }101)$

$$L_{avg} = \sum_{k=1}^{6} l(a_k)P(a_k) = \sum_{k=1}^{6} 3P(a_k) = 3\sum_{k=1}^{6} P(a_k) = 3 \text{ bits/symbol}$$

Huffman Coding/Decoding

- Coding/Decoding can be implemented using a look-up table.
- Decoding can be done unambiguously.

$$\underbrace{0\ 1\ 0\ 1\ 0}_{a_3}\ \underbrace{0\ 1\ 1}_{a_1}\ \underbrace{1}_{a_2}\ \underbrace{1}_{a_2}\ \underbrace{0\ 0}_{a_6}$$

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								

Original source		
Sym.	Prob.	Code
a_2	0.4	1
a_6	0.3	00
a_1	0.1	011
a_4	0.1	0100
a_3	0.06	01010
a_5	0.04	01011

Arithmetic (or Range) Coding
(addresses coding redundancy)

- The main weakness of Huffman coding is that it encodes source symbols one at a time.
- Arithmetic coding encodes sequences of source symbols together.
  - There is no one-to-one correspondence between source symbols and code words.
- Slower than Huffman coding but can achieve better compression.

Arithmetic Coding (cont'd)

- A sequence of source symbols is assigned to a sub-interval in [0,1) which can be represented by an arithmetic code, e.g.:

sub-interval / arithmetic code

$$\alpha_1\ \alpha_2\ \alpha_3\ \alpha_3\ \alpha_4 \Rightarrow [0.06752,\ 0.0688) \Rightarrow 0.068$$

- Start with the interval [0, 1) ; a sub-interval is chosen to represent the message which becomes smaller and smaller as the number of symbols in the message increases.

Arithmetic Coding (cont'd)

Encode message: $\alpha_1\ \alpha_2\ \alpha_3\ \alpha_3\ \alpha_4$

Source Symbol	Probability
a_1	0.2
a_2	0.2
a_3	0.4
a_4	0.2

1) Start with interval [0, 1)

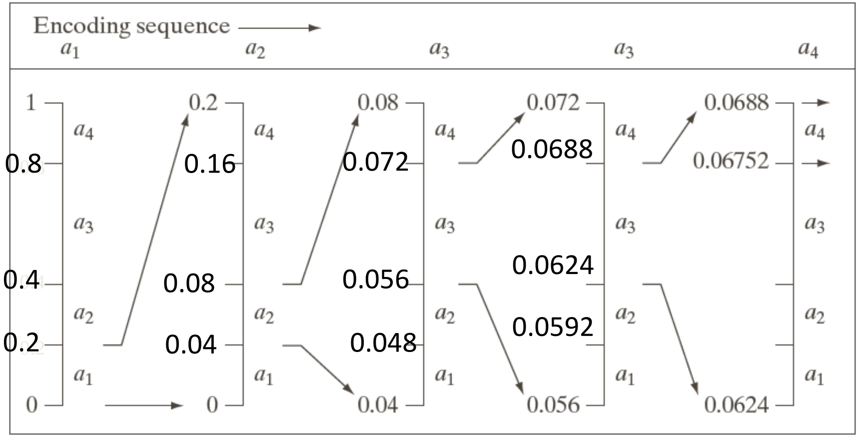
2) Subdivide [0, 1) based on the probabilities of α_i

Initial Subinterval
[0.0, 0.2)
[0.2, 0.4)
[0.4, 0.8)
[0.8, 1.0)

3) Update interval by processing source symbols

Example

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)

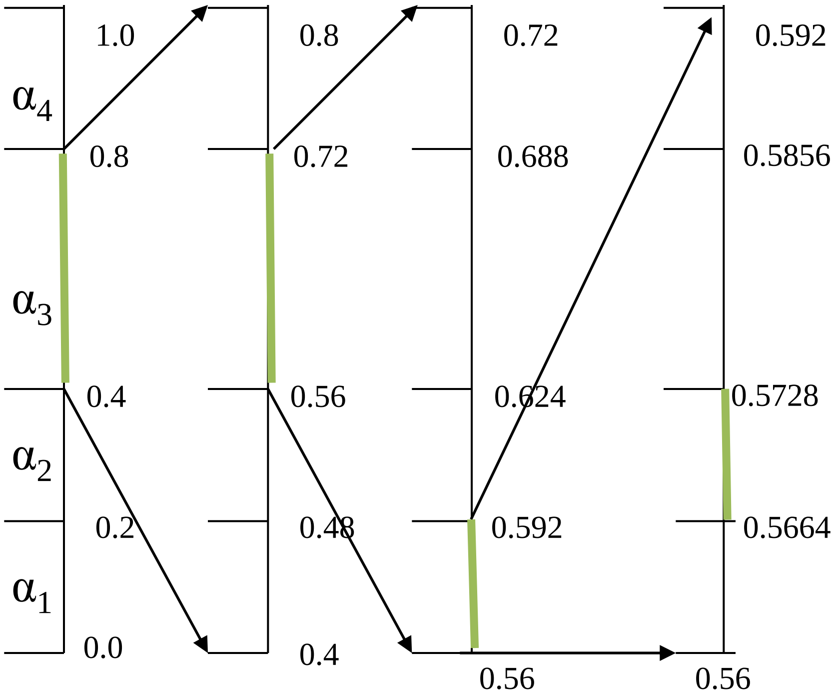


Encode

$\alpha_1\ \alpha_2\ \alpha_3\ \alpha_3\ \alpha_4$

[0.06752, 0.0688)

Arithmetic Decoding



Decode 0.572

(code length=4)

$\alpha_3\ \alpha_3\ \alpha_1\ \alpha_2$

LZW (Lempel-Ziv-Welch) Coding (addresses interpixel redundancy)

- Requires no prior knowledge of symbol probabilities.

- Assigns fixed length code words to variable length symbol sequences.
  - There is no one-to-one correspondence between source symbols and code words.

- Included in GIF, TIFF and PDF file formats

LZW Coding

- A codebook (or dictionary) needs to be constructed.

- Initially, the first 256 entries of the dictionary are assigned to the gray levels 0,1,2,..,255 (i.e., assuming 8 bits/pixel)

Consider a 4x4, 8 bit image

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

Initial Dictionary

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	-
511	-

LZW Coding (cont'd)

As the encoder examines image pixels, gray level sequences (i.e., blocks) that are not in the dictionary are assigned to a new entry.

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

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

- Is 39 in the dictionary……..Yes
- What about 39-39………….No
  * Add 39-39 at location 256

Example

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

Concatenated Sequence: CS = CR + P

(CR) (P)

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		

CR = empty
repeat
P=next pixel
CS=CR + P

If CS is found:
(1) No Output
(2) CR=CS

else:
(1) Output D(CR)
(2) Add CS to D
(3) CR=P

Decoding LZW

- Use the dictionary for decoding the “encoded output” sequence.
- The dictionary need not be sent with the encoded output.
- Can be built on the “fly” by the decoder as it reads the received code words.

Run-length coding (RLC)
(addresses interpixel redundancy)

- Reduce the size of a repeating string of symbols (i.e., runs):

1 1 1 1 1 0 0 0 0 0 0 1 → (1,5) (0, 6) (1, 1)

a a a b b b b b b c c → (a,3) (b, 6) (c, 2)

- Encodes a run of symbols into two bytes: **(symbol, count)**
- Can compress any type of data but cannot achieve high compression ratios compared to other compression methods.

Combining Huffman Coding with Run-length Coding

- Assuming that a message has been encoded using Huffman coding, additional compression can be achieved using run-length coding.

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

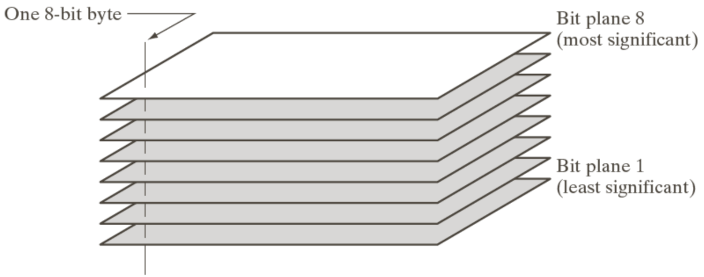
e.g., (0,1)(1,1)(0,1)(1,0)(0,2)(1,4)(0,2)

Bit-plane coding
(addresses interpixel redundancy)

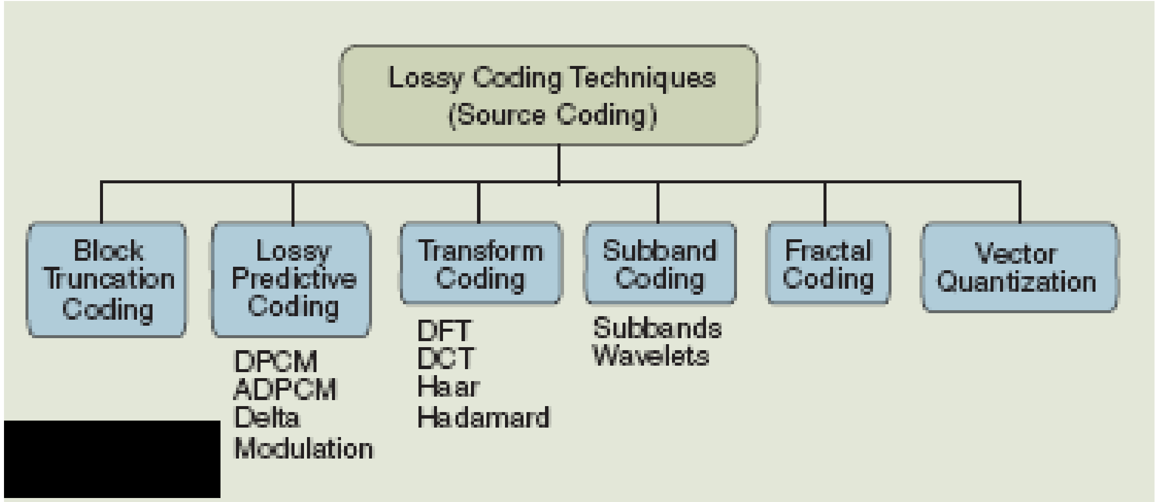
- Process each bit plane individually.

(1) Decompose an image into a series of binary images.

(2) Compress each binary image (e.g., using run-length coding)

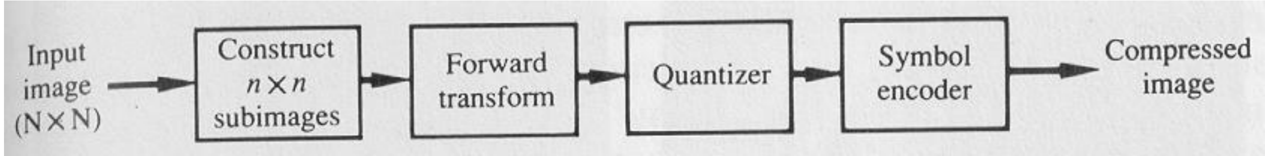


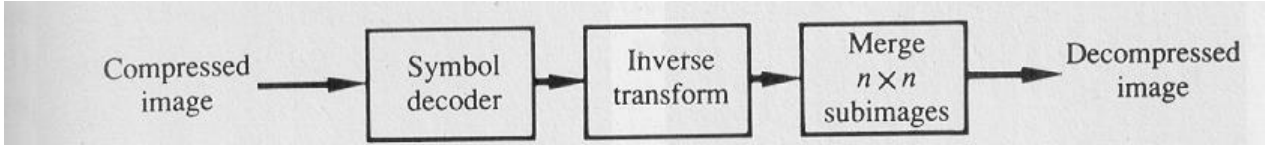
Lossy Methods - Taxonomy



Lossy Compression

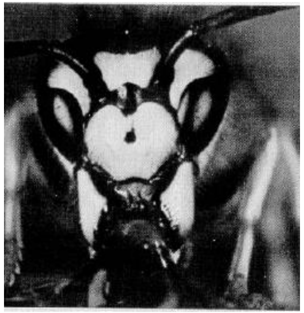
- Transform the image into some other domain to reduce interpixel redundancy.

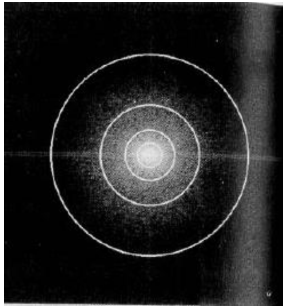




Example: Fourier Transform

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad \text{x,y=0,1,...,N-1}$$





Note that the magnitude of the FT decreases, as *u*, *v* increase!

K << N

$$\hat{f}(x,y) = \frac{1}{N}\sum_{u=0}^{K-1}\sum_{v=0}^{K-1} F(u,v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad \text{x,y=0,1,...,N-1}$$

$$\sum_{x,y}(\hat{f}(x,y) - f(x,y))^2 \text{ is very small !!}$$

Transform Selection

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

- T(u,v) can be computed using various transformations, for example:
  - DFT
  - DCT (Discrete Cosine Transform)
  - KLT (Karhunen-Loeve Transformation) or Principal Component Analysis (PCA)

- JPEG using DCT for handling interpixel redundancy.

DCT (Discrete Cosine Transform)

[proposed by Nasir Ahmed, T. Natarajan, K.R.Rao (1972)]

All Real

Forward: $$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)cos(\frac{(2x+1)u\pi}{2N})cos(\frac{(2y+1)v\pi}{2N}),$$

u, v=0,1,...,N-1

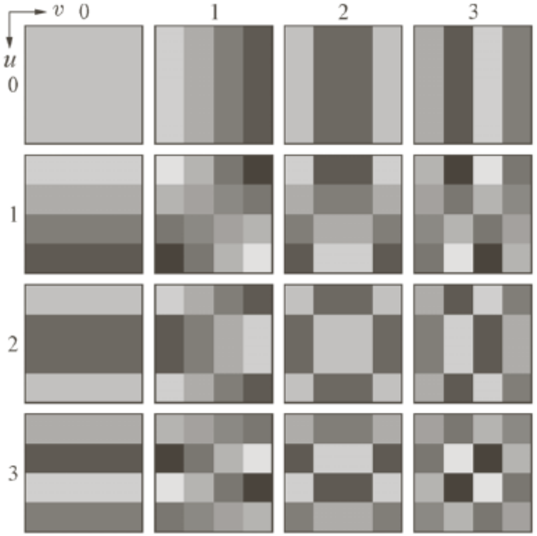
Inverse: $$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)cos(\frac{(2x+1)u\pi}{2N})cos(\frac{(2y+1)v\pi}{2N}),$$

x, y=0,1,...,N-1

$$\alpha(u) = \begin{cases}\sqrt{1/N} & \text{if u=0}\\ \sqrt{2/N} & \text{if u>0}\end{cases} \qquad \alpha(v) = \begin{cases}\sqrt{1/N} & \text{if v=0}\\ \sqrt{2/N} & \text{if v>0}\end{cases}$$

DCT (cont'd)

- Basis functions for a 4x4 image (i.e., cosines of different frequencies).



DCT (cont'd)

Using
8 x 8 sub-images
yields 64 coefficients
per sub-image.

Reconstructed images
by truncating
50% of the
coefficients

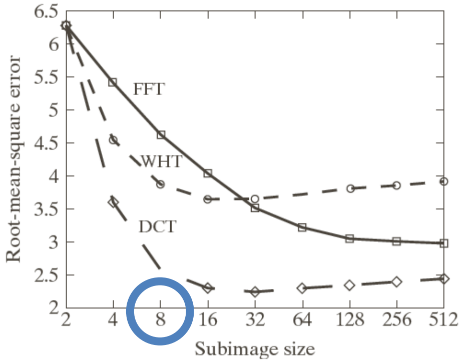
More compact
transformation

DFT | WHT | DCT

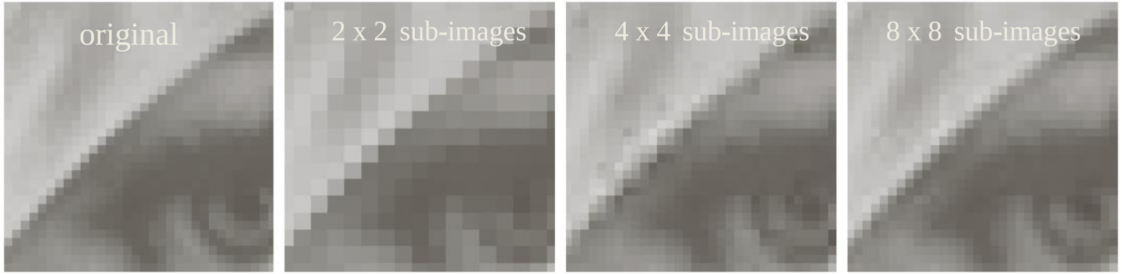
RMS error: 2.32 | 1.78 | 1.13

DCT (cont'd)

- Sub-image size selection:



Reconstructions

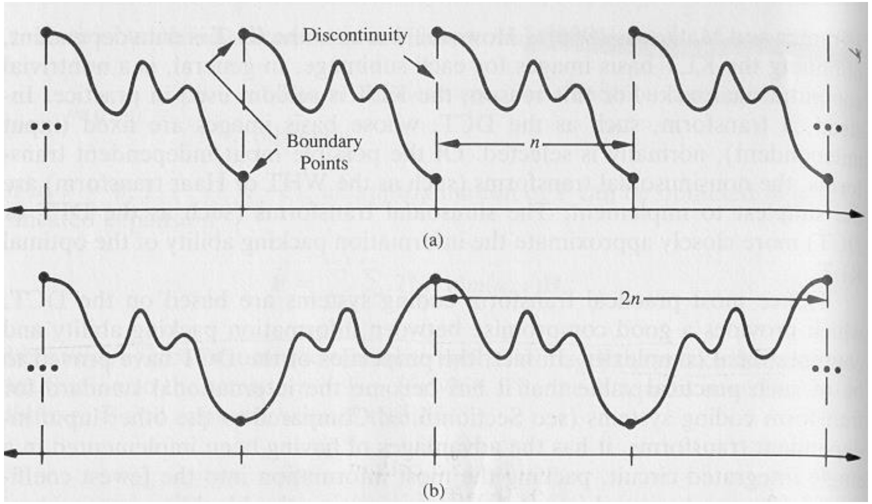


DCT (cont'd)

- DCT minimizes "blocking artifacts" (i.e., boundaries between subimages do not become very visible).

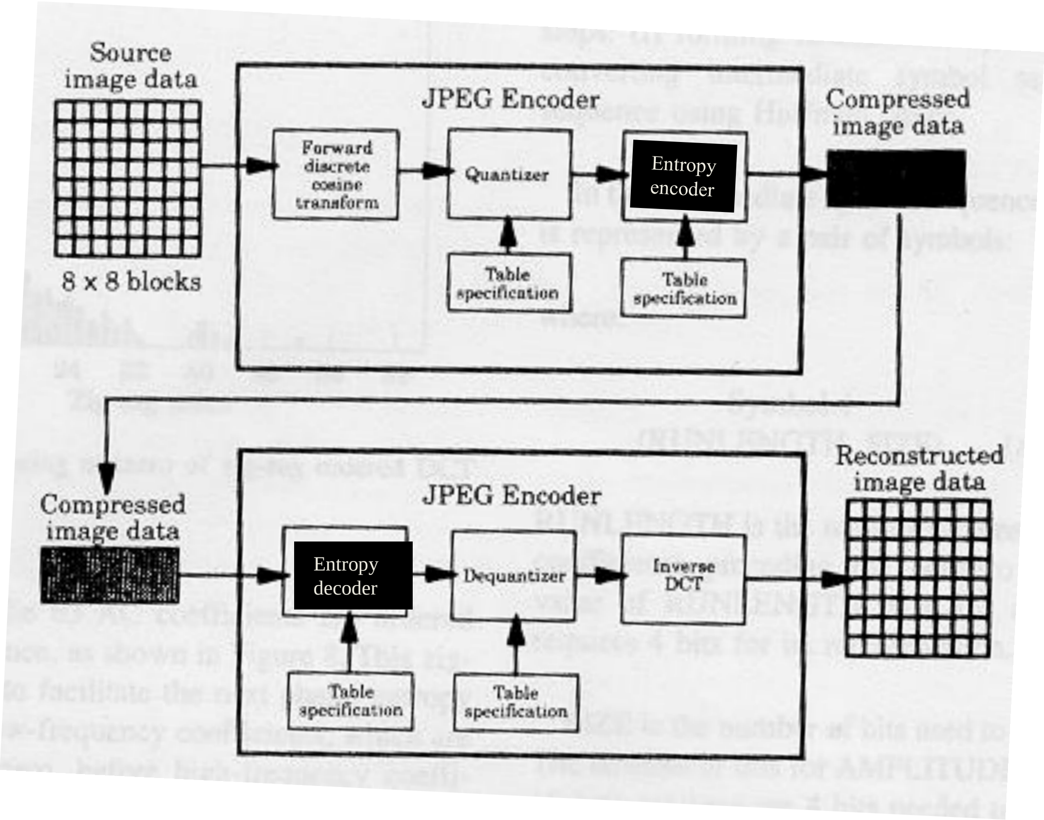
DFT
has n-point periodicity

DCT
has 2n-point periodicity



JPEG Compression

Accepted as an international image compression standard in 1992.



JPEG - Steps

1. Divide image into 8x8 subimages/bocks.

For each subimage do:

2. Shift the gray-levels in the range [-128, 127]

3. Apply DCT → 64 coefficients

1 DC coefficient: F(0,0)

63 AC coefficients: F(u,v)

Example

(a) Original 8 8 block

140	144	147	1140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

(b) Shifted block

12	16	19	12	11	27	51	47
16	24	12	19	12	20	39	51
24	27	8	39	35	34	24	44
40	17	28	32	24	27	8	32
34	20	28	20	12	8	19	34
19	39	12	27	27	12	8	34
8	28	-5	39	34	16	12	19
20	27	8	27	24	19	19	8

[-128, 127]

(c) Block after FDCT Eqn. (5)

185	-17	14	-8	23	-9	-13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

(non-centered spectrum)

JPEG Steps

4. Quantize the coefficients (i.e., reduce the amplitude of coefficients that do not contribute a lot).

$$C_q(u,v) = Round\left[\frac{C(u,v)}{Q(u,v)}\right]$$

Q(u,v): quantization table

Example

- Quantization Table Q[i][j]

(d) Quantization table (quality = 2)

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

```
for i=0 to n;
  for j=0 to n;
    Q[i,j]= 1 + (1+i+j)*quality;
  end j;
end i;
```

$$1 \leq quality \leq 25$$

(best - low compression) (worst - high compression)

Example (cont'd)

(c) Block after FDCT Eqn. (5)

185	-17	14	-8	23	-9	-13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

(d) Quantization table (quality = 2)

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

Quantization

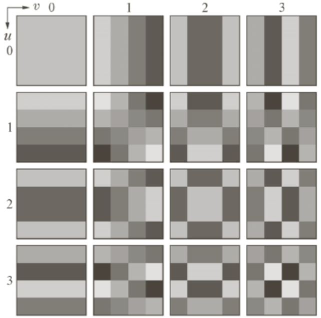
(e) Block after quantization Eqn. (6)

61	-3	2	0	2	0	0	-1
4	-4	2	0	0	0	0	0
-1	-2	0	0	-1	0	-1	0
0	0	1	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

JPEG Steps (cont'd)

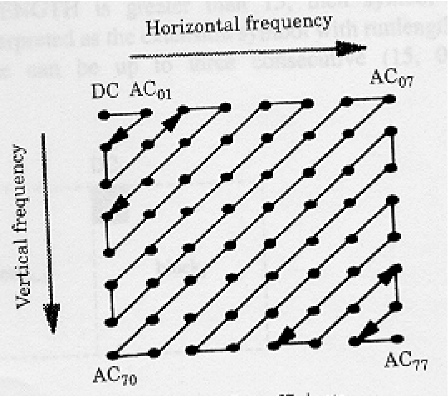
5. Order the coefficients using **zig-zag** ordering

- Creates long runs of zeros (i.e., ideal for run-length encoding)



(e) Block after quantization Eqn. (6)

61	-3	2	0	2	0	0	-1
4	-4	2	0	0	0	0	0
-1	-2	0	0	-1	0	-1	0
0	0	1	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



(f) Zig-zag sequence

61,-3,4,-1,-4,2,0,2,-2,0,0,0,0,0,2,0,0,0,1,0,0,0,0,0,0,-1,0,0,-1,0,0,
0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

JPEG Steps (cont’d)

6. Encode coefficients using variable length encoding:

   6.1 Form “intermediate” symbol sequence.

   6.2 Encode “intermediate” symbol sequence into a binary sequence.

Final Symbol Sequence

(g) Intermediate synbol sequence

(6)(61),(0,2)(-3),(0,3)(4),(0,1)(-1),(0,3)(-4),(0,2)(2),(1,2)(2),(0,2)(-2),
[redacted](5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

(e) Encoded bit sequence (total 98 bits)

1110111101001001000001000110110110111001011111111011
11011101011111011011100011101101111110100101 0

What is the effect of the "Quality" parameter?

(58k bytes)

(21k bytes)

(8k bytes)

lower compression → higher compression

$$1 \leq quality \leq 25$$

Results using JPEG compression

file size 45853 bytes
compression ratio: 12.9

Fine details have been lost.

Image has an artificial “blocky” pattern superimposed on it.

Artifacts will affect the performance of fingerprint recognition.

Results using WSQ compression

file size 45621 bytes
compression ratio: 12.9

Fine details are better preserved.

No “blocky” artifacts.

JPEG

- JPEG compression exploits two observations:
  - **#1**: Human eyes don't see color (chrominance) quiteas well as brightness (luminance) ➔ Compress in a color-dominant space (RGB --> YCbCr)
  - **#2**: Human eyes can't distinguish high frequency changes in image intensity (downsample (Cb,Cr), quantize, DCT)

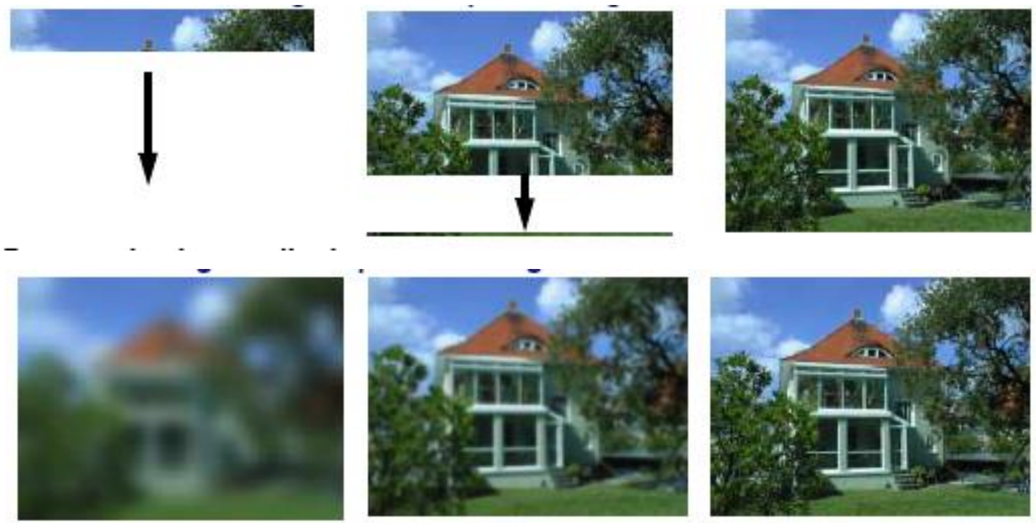
JPEG Modes

- JPEG supports several different modes
  - Sequential Mode
  - Progressive Mode
  - Hierarchical Mode
  - Lossless Mode

- The default mode is "sequential"
  - Image is encoded in a single scan (left-to-right, top-to-bottom).

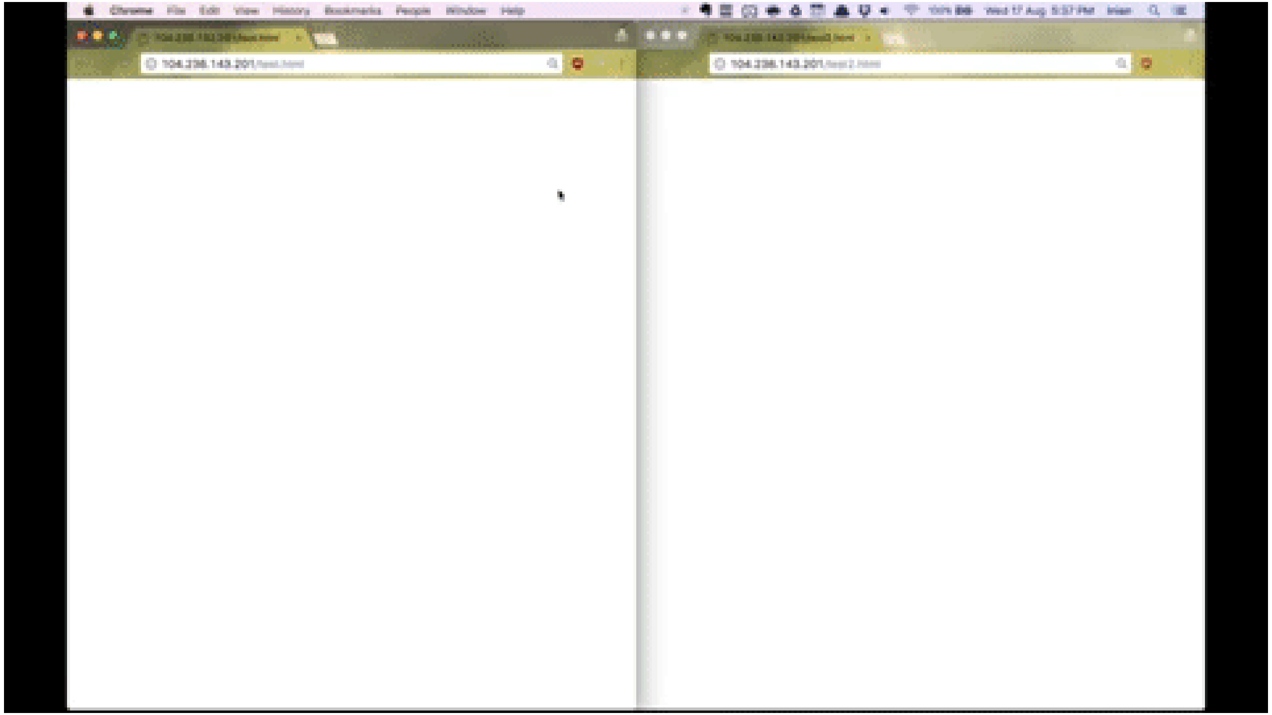
Progressive JPEG

- Image is encoded in multiple scans, in order to produce a quick, rough decoded image when transmission time is long.



Sequential

Progressive



JPEG at 0.125 bpp (enlarged)

C. Christopoulos, A. Skodras, T. Ebrahimi, JPEG2000 (online tutorial)

JPEG2000 at 0.125 bpp

C. Christopoulos, A. Skodras, T. Ebrahimi, JPEG2000 (online tutorial)

Other popular formats

- GIF → lossy
- PNG → lossless
- Video
  - MPEG etc. (exploit temporal redundancy also)

Reference

- Ch 8, G&W textbook