Digital Image Processing (CSE 478)
Lecture 19-20: Image Compression

Vineet Gandhi

Center for Visual Information Technology (CVIT), IIIT Hyderabad

Motivation

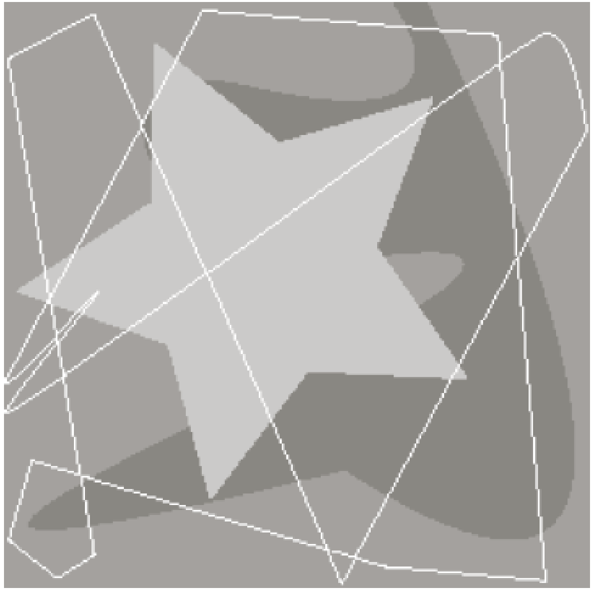
- Consider a 2 hour, full HD video (resolution of 1920×1080)
- The storage space required per frame :$1920 \times 1080 \times 24$ bits = 6.22 MB
- Space required per second: $1920 \times 1080 \times 24 \times 30$ bits
- Space required for entire movie: $1920 \times 1080 \times 24 \times 30 \times 2 \times 60 \times 60$ bits = $1920 \times 1080 \times 3 \times 30 \times 2 \times 60 \times 60$ bytes = 1.34×10^{12} bytes= **1340 GB**
- To put it on a 25 GB blu ray disc: required compression factor = **53.6**

Redundancy

- Coding redundancy
- Spatial and Temporal redundancy
- Irrelevant Information (often perceptually irrelevant)

Compression is all about exploiting these redundancies!

Coding redundancy



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

Average encoding length?

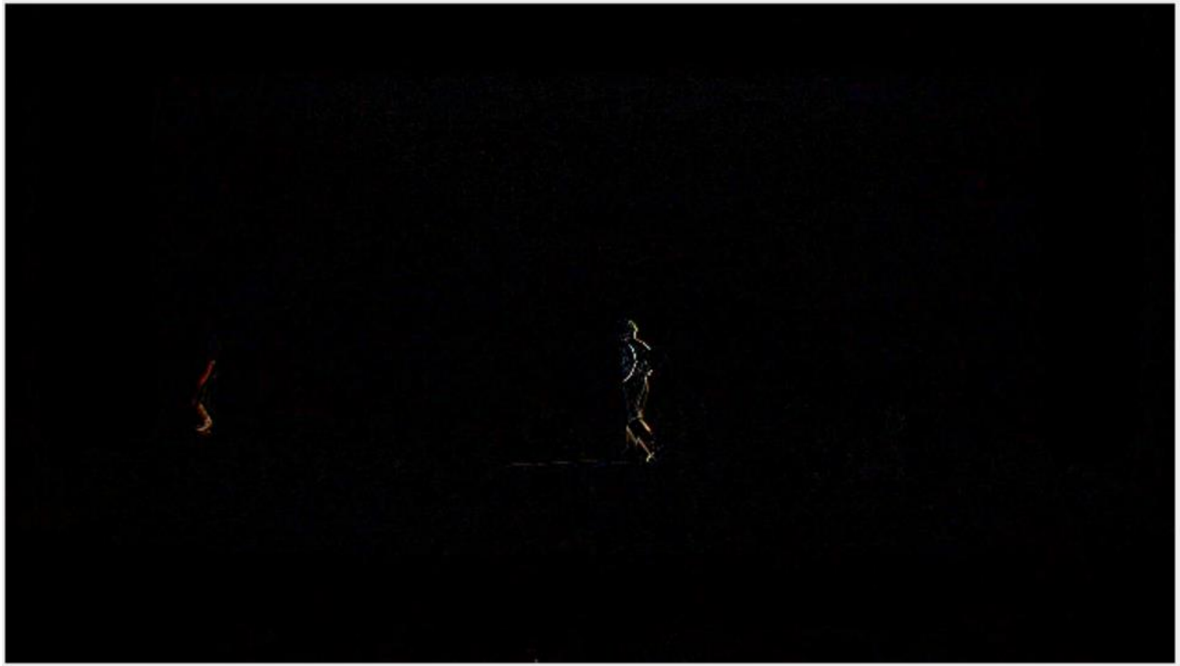
Spatial and temporal redundancy

Spatial and temporal redundancy

frame t

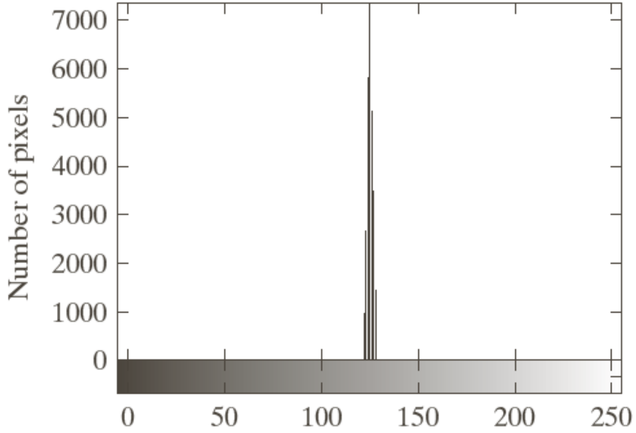
frame t+1

Spatial and temporal redundancy

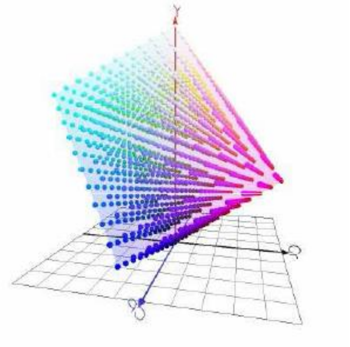


Irrelevant information or perceptual redundancy

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



Irrelevant information or perceptual redundancy



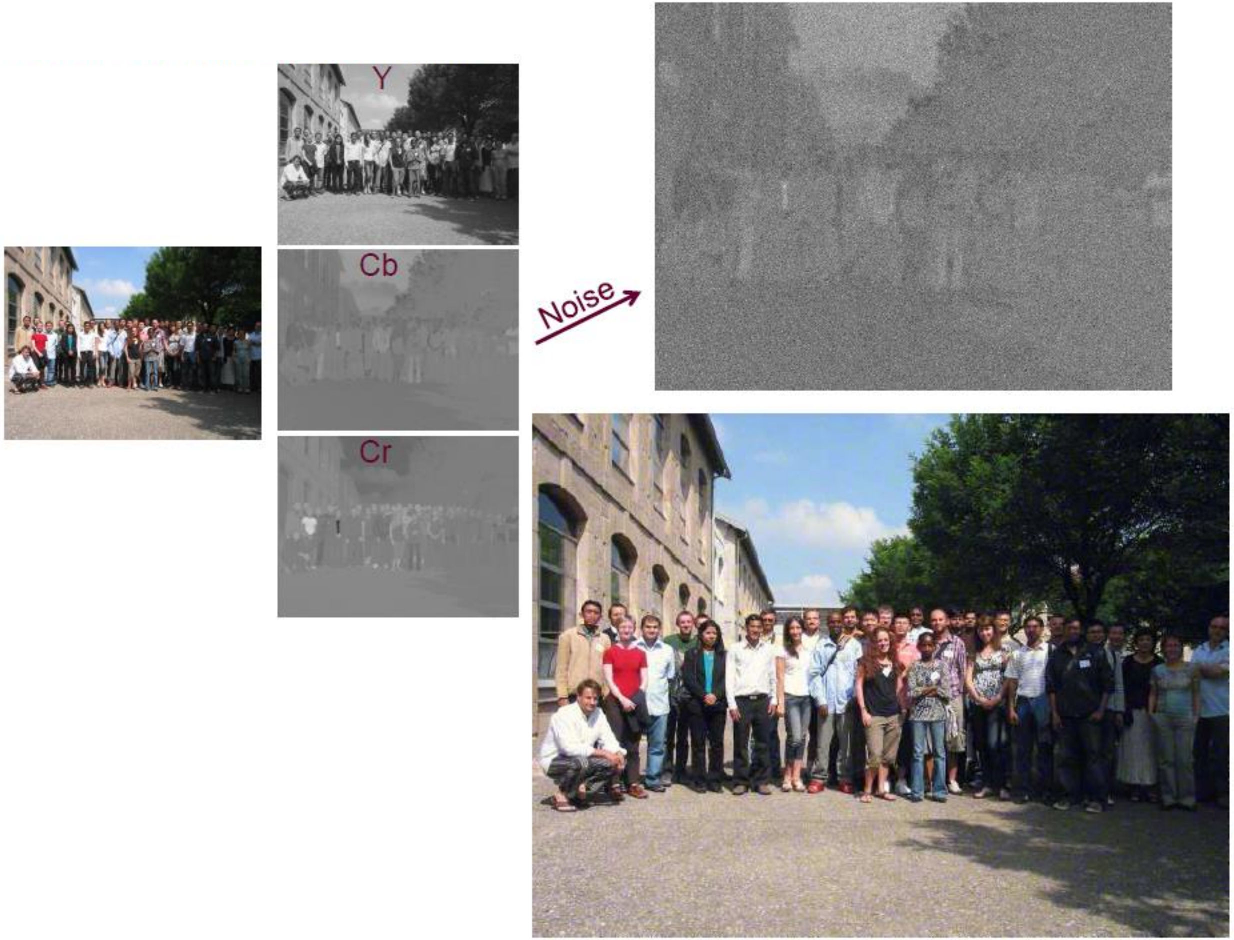
Y

Cb

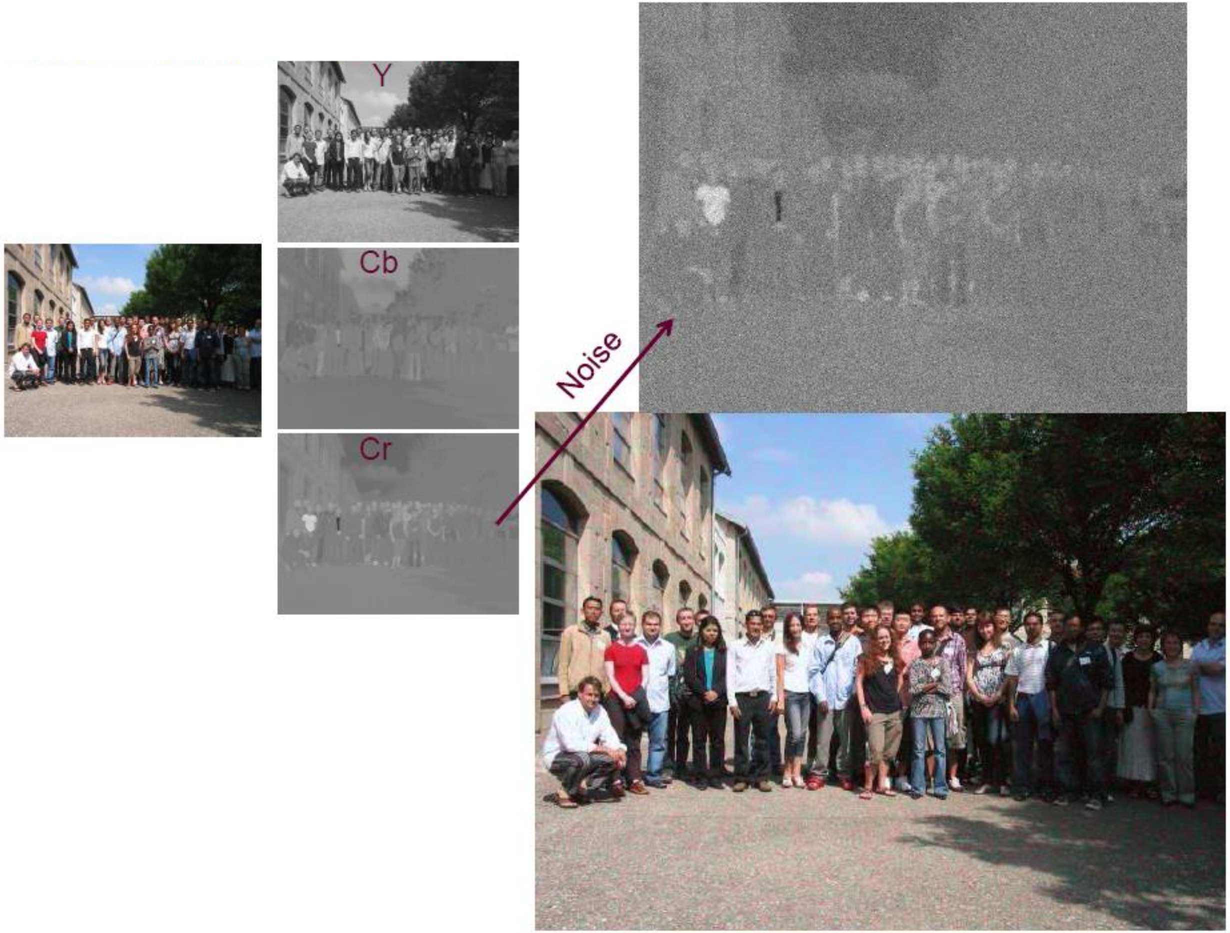
Cr

Irrelevant information or perceptual redundancy

Irrelevant information or perceptual redundancy



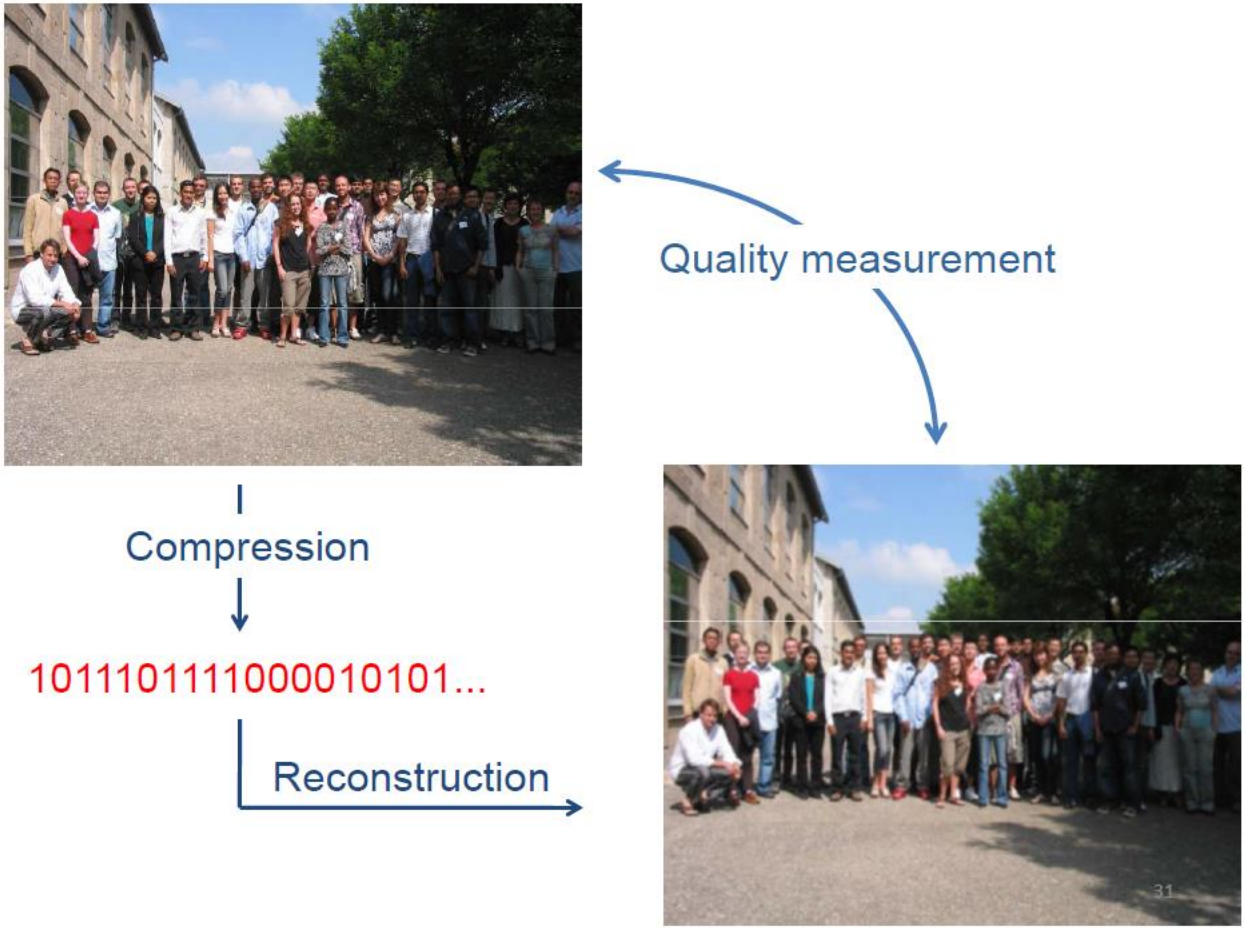
Irrelevant information or perceptual redundancy



Compression types and evaluations

Two kinds:

1. Lossless
2. Lossy

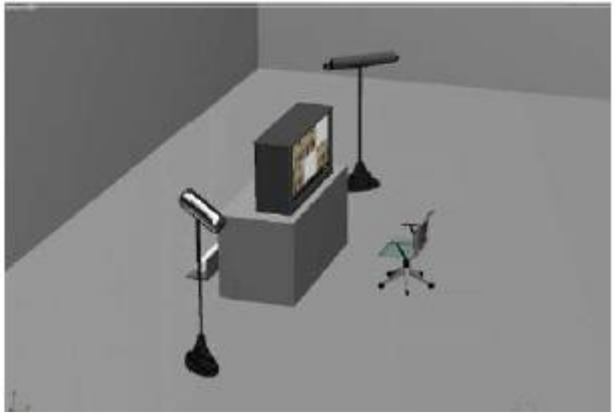


Quality measurement: judged by human viewers

- Five scale system on the degree of impairment
  1. Impairment is not noticeable
  2. Impairment is just noticeable
  3. Impairment is definitely noticeable, but not objectionable
  4. Impairment is objectionable
  5. Impairment is extremely objectionable

Advantages: relies on HVS

Drawbacks: time, viewing conditions, viewers?



Quality measurement: Signal to noise ratio

$$e(x,y) = f(x,y) - g(x,y).$$

$$E_{\mathrm{ms}} = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e(x,y)^2$$

1

$$SNR_{ms} = 10 \log_{10} \left(\frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y)^2}{MN \cdot E_{\mathrm{ms}}} \right)$$

2

$$PSNR = 10 \log_{10} \left(\frac{255^2}{E_{\mathrm{ms}}} \right)$$

Information theory: Self energy

- Information is defined as knowledge, fact, and news
- It can be measured quantitatively

- The carriers of information are symbols. 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: Entropy

- Consider a source that contains L possible symbols {s,i=0,1,2,…,L-1}
- With corresponding occurrence probabilities defined as {p_i, i=0,1,2,…,L−1}

- **Entropy**

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

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

log (0.47) = -1.09
log(0.03) = -5.06

Information theory: Shannon's theorem

- Shannon's lossless source coding theorem states that for a discrete, memoryless, stationary information source, the minimum bit rate required to encode a symbol on average is equal to the entropy of the source.
- In other words: we can't do better than the entropy
- Lets understand with an example

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

Validity of the code?

- Lets take an example

Symbol	Probability	Code1	Code2	Code3	Code4
s1	1/2	0	0	0	0
s2	1/4	0	1	10	01
s3	1/8	1	00	110	011
s4	1/8	10	11	111	0111

Image Compression: Overview

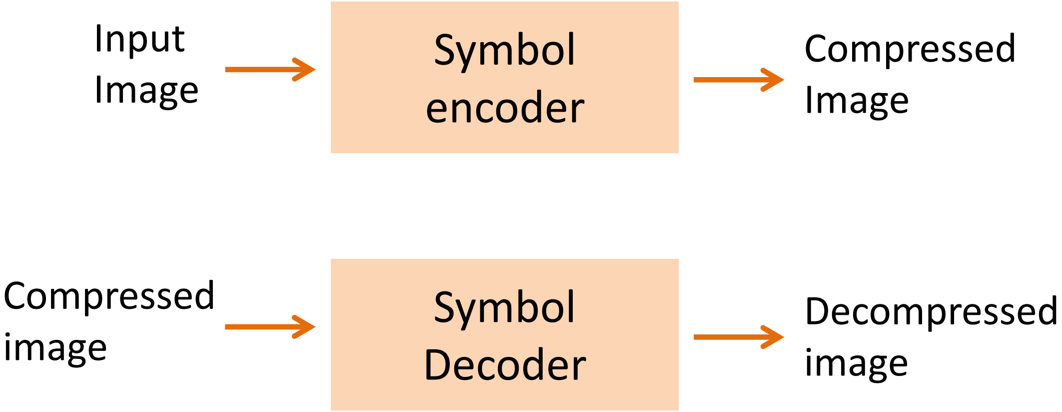


Image Compression: Overview

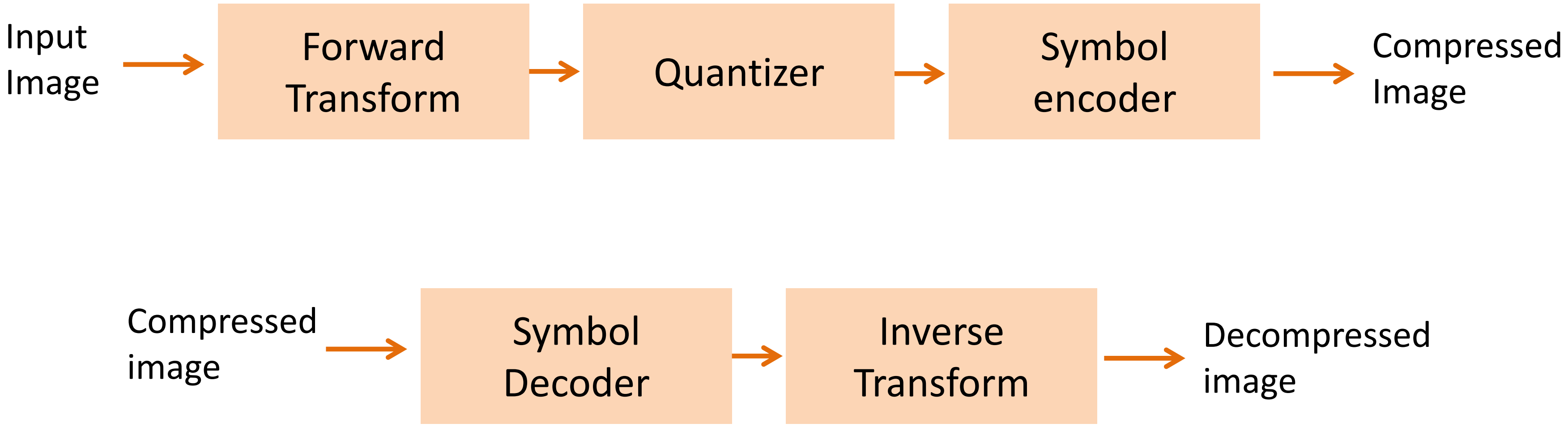


Image Compression: Overview

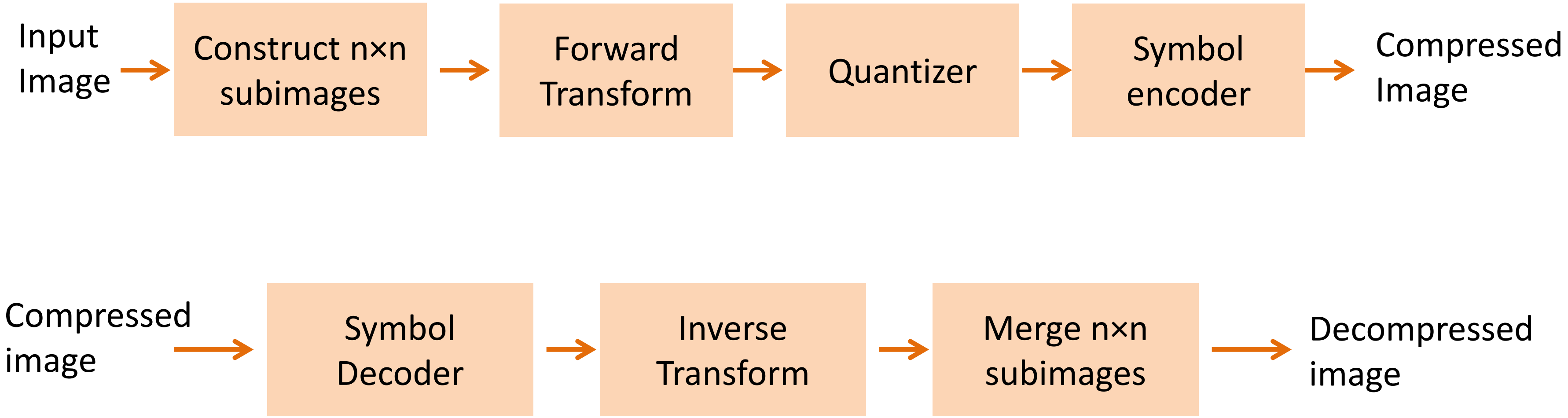
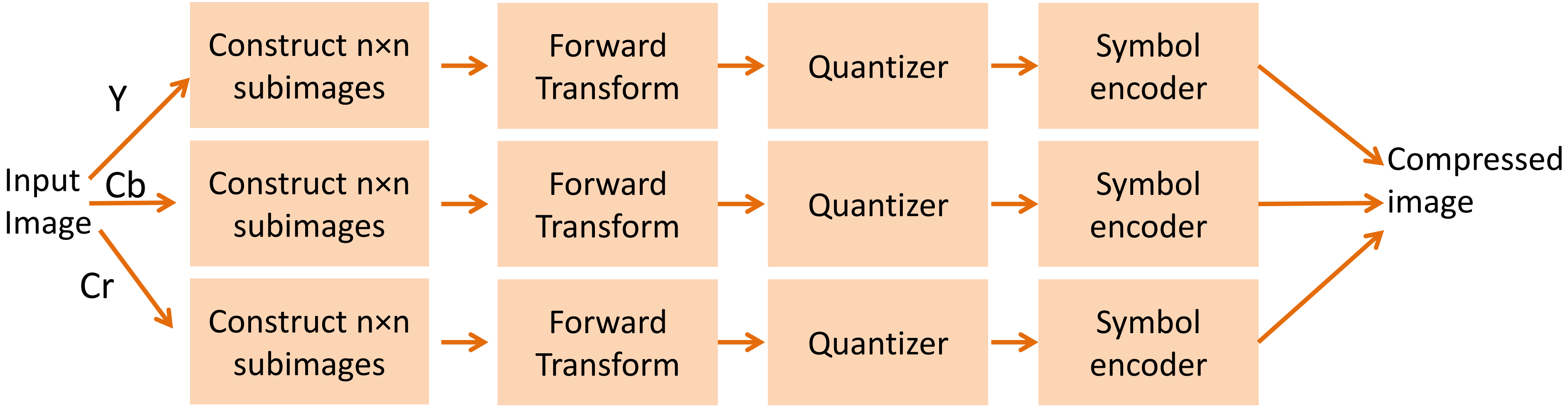
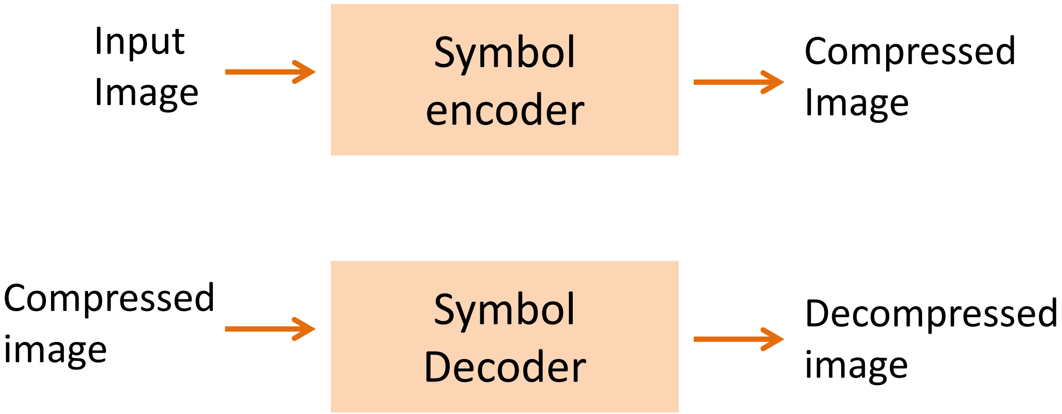


Image Compression: Overview



Lossless compression

Lets begin with simplest case: Lossless compression



Lossless compression : Huffman coding

- Already discussed in class
- Quick example : ABRAAKADABRAA

Lossless compression : Run Length coding

- Already discussed in class
- Quick example:

  0000000000000001111111111100000000111111 → 40 bits

- 15 0's, 11 1's , 8 0's, 6 1's
- 1111101110000110

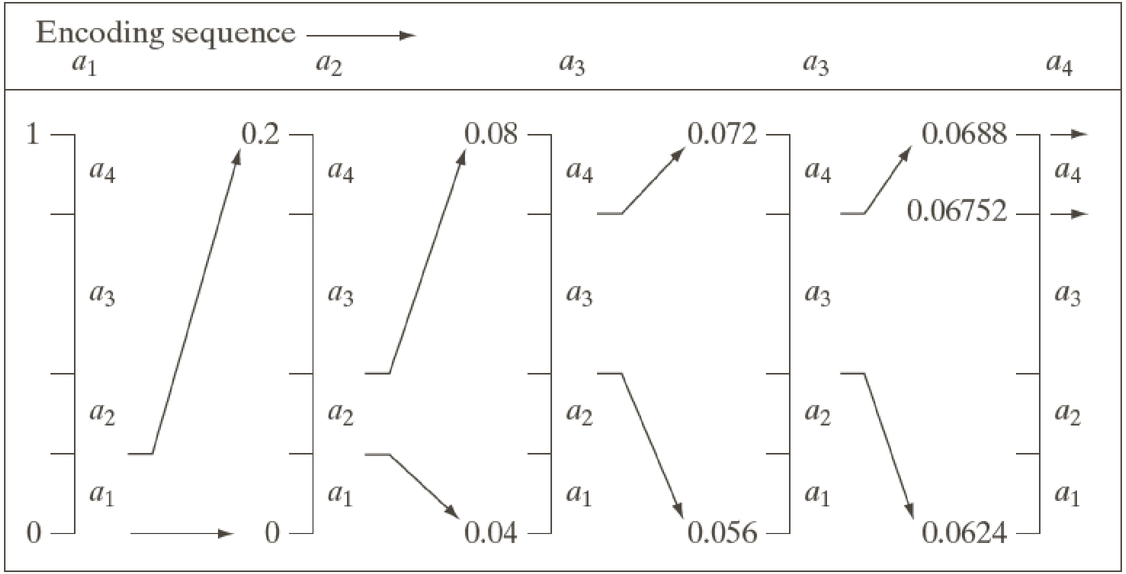
How many bits to store the count?

Give a scenario where run length coding will be extremely effective?

Lossless compression : Arithmetic coding

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)

Input sequence: $a_1 a_2 a_3 a_3 a_4$



Final code: 0.068 (could be anything between the computed range)

3 decimal digits for 5 symbols = 3/5 digits per symbol

How many bits per symbol?

Lossless compression : Arithmetic coding

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)

Another sequence: $a_1 a_1\ a_3$

Lossless compression: Dictionary coding

- Important in presence of recurring patterns
- Static and adaptive
- Static example: a b r a c a d a b r a

- Dynamic dictionary
  - Build during compression after observing the data
  - Rebuild at the decompression step
  - LZW is the commonly used algorithm

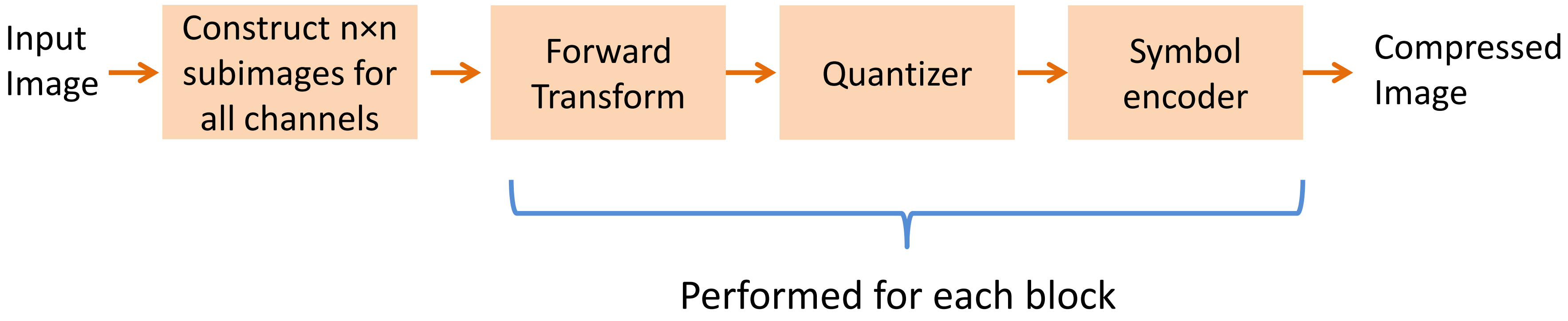
$$A = \{a, b, c, d, r\}$$

Code	Entry
000	a
001	b
010	c
011	d
100	r
101	ab
110	ac
111	ad

K. Sayood, Introduction to Data Compression, 2nd edition Morgan Kaufmann,2000

Lossy compression (with a case study of JPEG)

Lossy compression: JPEG



Block transform coding

- Partition the image into small non overlapping n×n blocks
  - 8×8 blocks in JPEG

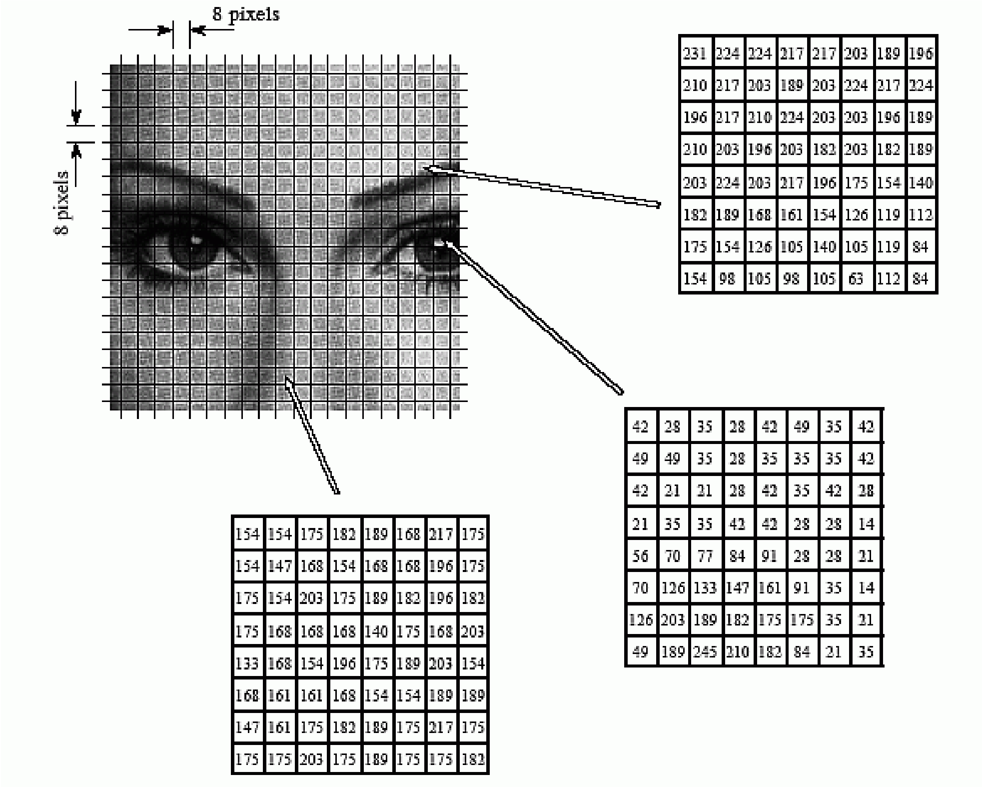


Image source: dspguide.com

Block Transform coding

- Process blocks using 2D transforms
- General forward transform of image g, size n×n:

$$T(u,v) = \sum_{x=0}^{n-1} \sum_{y=o}^{n-1} g(x,y)\, r(x,y,u,v)$$

- Inverse transform

$$g(x,y) = \sum_{x=0}^{n-1} \sum_{y=o}^{n-1} T(u,v)\, s(x,y,u,v)$$

- $r(x,y,u,v)$ and s(x,y,u,v) are basis functions or transformation kernels

Block Transform coding

$$T(u,v) = \sum_{x=0}^{n-1}\sum_{y=o}^{n-1} g(x,y)\, r(x,y,u,v) \qquad g(x,y) = \sum_{x=0}^{n-1}\sum_{y=o}^{n-1} T(u,v)\, s(x,y,u,v)$$

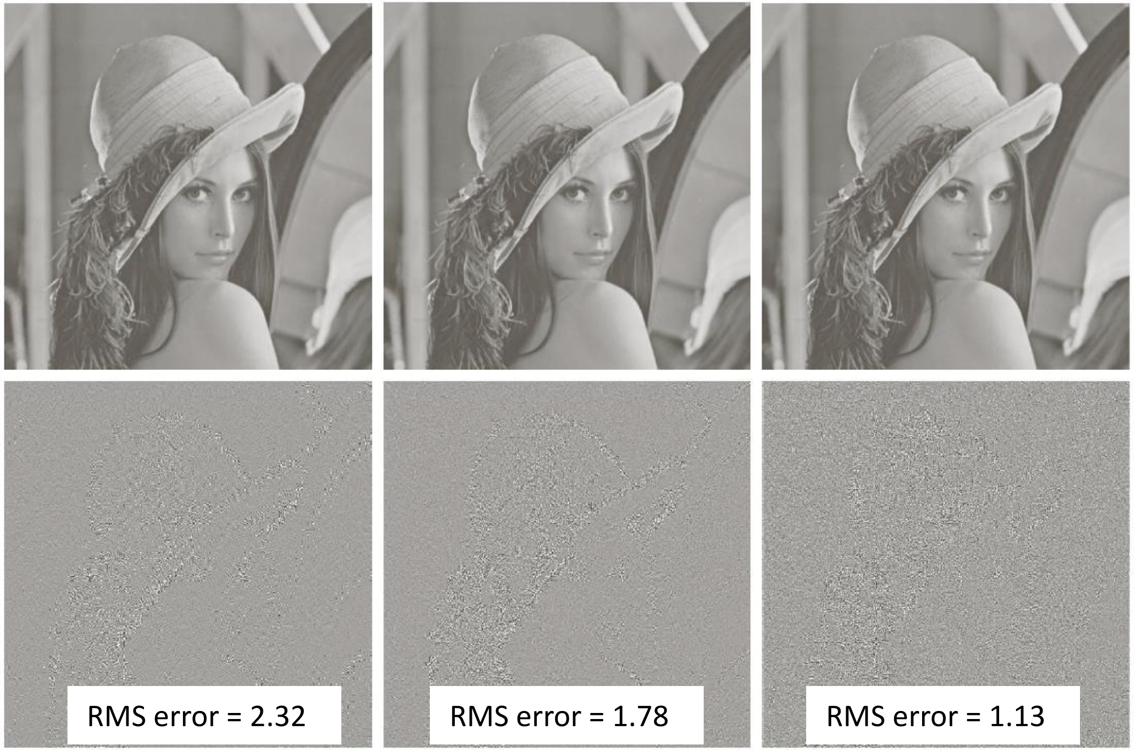
- $r(x,y,u,v) = e^{-j2\pi(ux+vy)/n}$ and $\mathrm{s}(x,y,u,v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n}$
- $r(x,y,u,v) = \mathrm{s}(x,y,u,v) = \frac{1}{n}(-1)^{\sum_{i=0}^{m-1}\left[b_i(x)p_i(u)+b_i(y)p_i(v)\right]}$
- $r(x,y,u,v) = \mathrm{s}(x,y,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$

Block Transform coding: which transform to use?

Apply transform to each 8×8 block

Keep highest 50% of the coefficients in each block

Reconstruct using the inverse transform on each block



a b c
d e f

FIGURE 8.24 Approximations of Fig. 8.9(a) using the (a) Fourier, (b) Walsh-Hadamard, and (c) cosine transforms, together with the corresponding scaled error images in (d)–(f).

Block Transform coding: which transform to use?

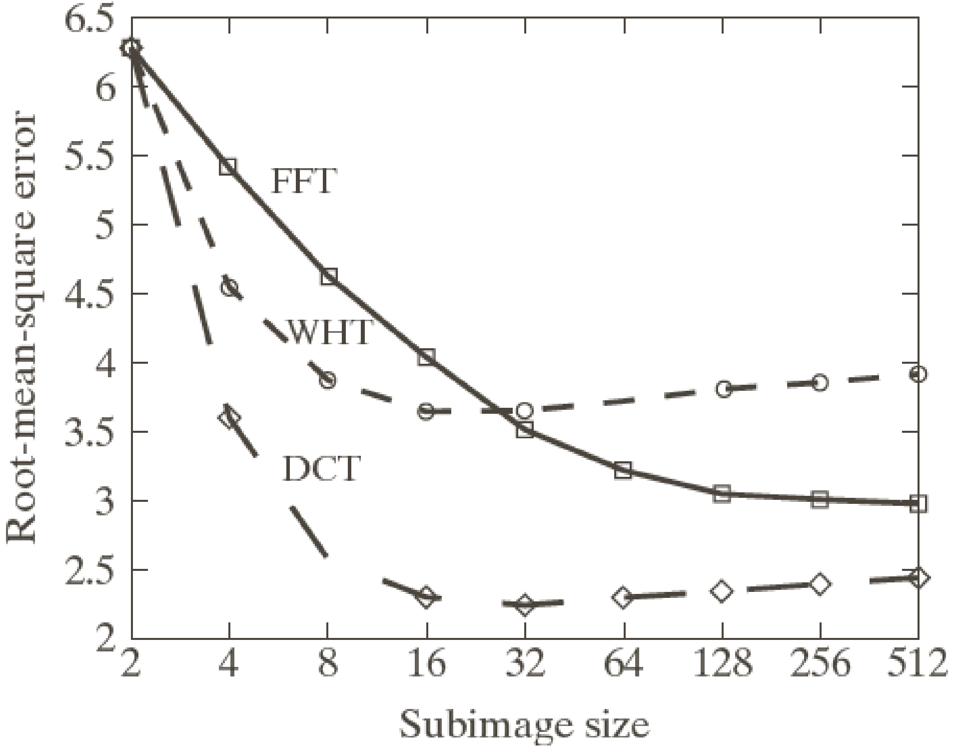
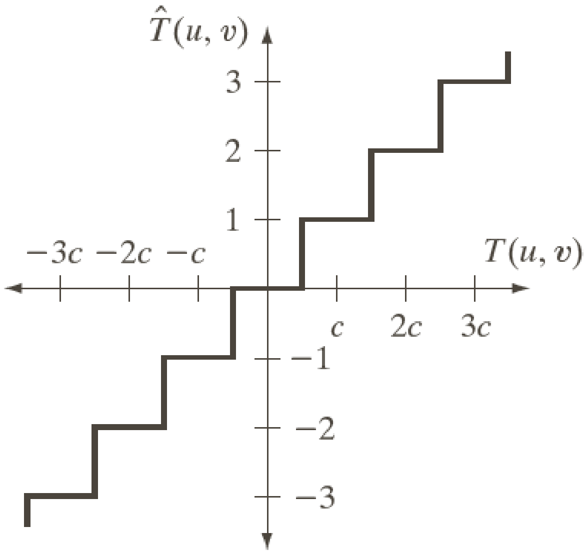


FIGURE 8.26 Reconstruction error versus subimage size.

Quantization



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

Different coefficients quantized with different step-size

Finally encode the quantized output!

Quantization (example)

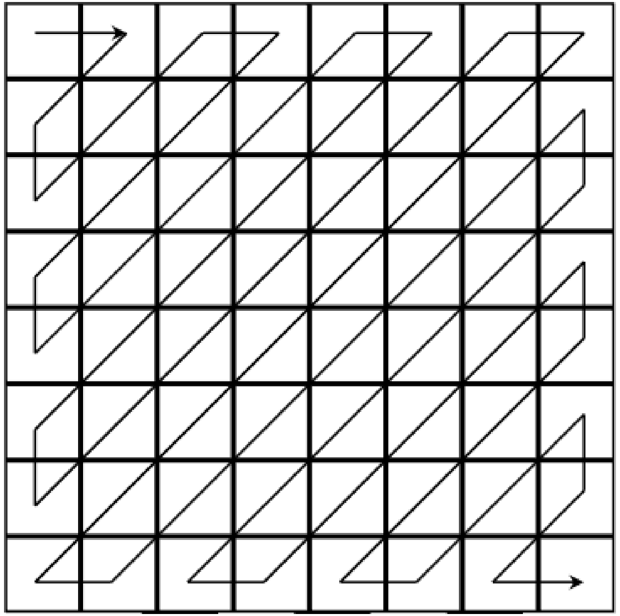
−415	−29	−62	25	55	−20	−1	3
7	−21	−62	9	11	−7	−6	6
−46	8	77	−25	−30	10	7	−5
−50	13	35	−15	−9	6	0	3
11	−8	−13	−2	−1	1	−4	1
−10	1	3	−3	−1	0	2	−1
−4	−1	2	−1	2	−3	1	−2
−1	−1	−1	−2	−1	−1	0	−1

$\xrightarrow{Q}$

−26	−3	−6	2	2	0	0	0
1	−2	−4	0	0	0	0	0
−3	1	5	−1	−1	0	0	0
−4	1	2	−1	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

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

Symbol encoding (Zigzag ordering)

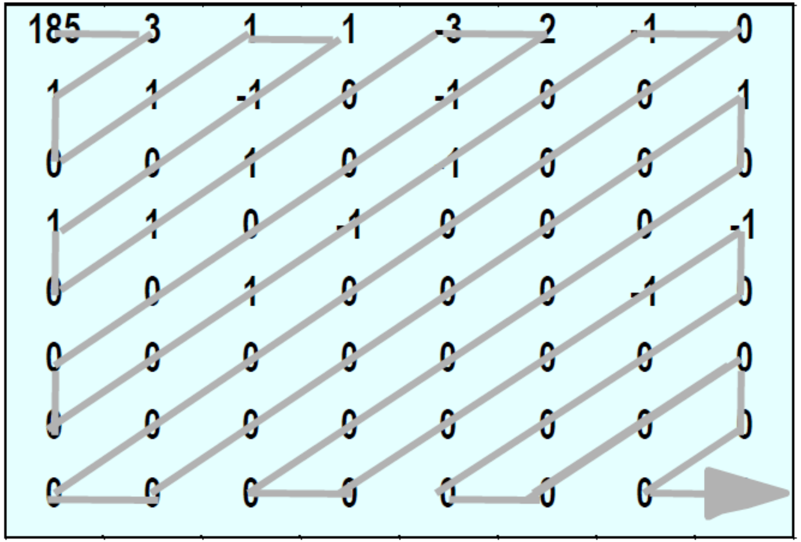


0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

JPEG uses run length encoding!

Symbol coding example

- Zigzag scan (additional example)

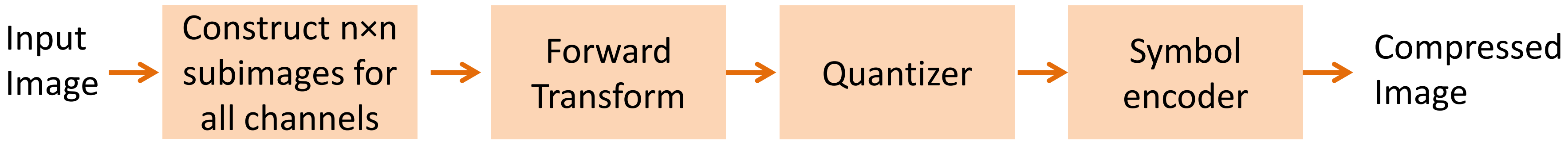


Run length coding

Mean of Block: 185

(0,3) (0,1) (1,1) (0,1) (0,1) (0,1) (0,-1) (1,1)
(1,1) (0,1) (1,-3) (0,2) (0,-1) (6,1) (0,-1) (0,-1)
(1,-1) (14,1) (9,-1) (0,-1) EOB

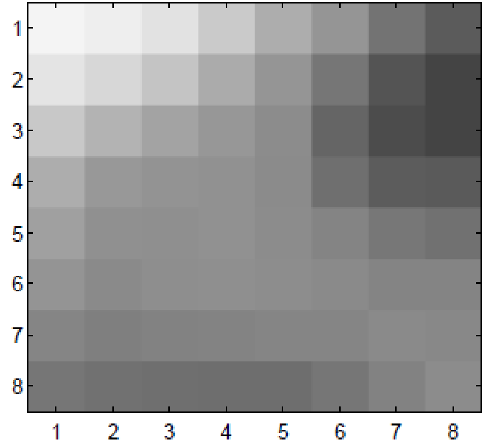
Lossy compression: JPEG



Lest understand the entire procedure with an example

- Consider a single 8×8 pixel block **B**:

$$\begin{pmatrix} 245 & 239 & 227 & 203 & 174 & 150 & 116 & 92 \\ 229 & 216 & 197 & 172 & 150 & 119 & 85 & 69 \\ 201 & 180 & 164 & 152 & 141 & 102 & 77 & 69 \\ 174 & 153 & 148 & 146 & 140 & 112 & 93 & 91 \\ 161 & 145 & 144 & 146 & 141 & 133 & 120 & 114 \\ 149 & 139 & 143 & 144 & 142 & 139 & 133 & 133 \\ 134 & 128 & 131 & 132 & 134 & 134 & 139 & 137 \\ 119 & 114 & 112 & 111 & 111 & 119 & 131 & 141 \end{pmatrix}$$



- Intensity range → [0 255]
- Subtract 127 from each entry and computer 2D DCT

Forward transform and quantization

- DCT of image block

$$\hat{\mathsf{B}} = \begin{pmatrix} 118.9 & 187.7 & -17.7 & 16.8 & 14.4 & 2.4 & 5.3 & 3.5 \\ 104.1 & 187.1 & -30.8 & 10.0 & -1.0 & -4.7 & 0.6 & 0.3 \\ 46.3 & 10.4 & 9.1 & -9.0 & -15.7 & 0 & -1.3 & -2.7 \\ 76.8 & -12.1 & -10.7 & -0.2 & -10.4 & 4.8 & 2.7 & -3.3 \\ 6.4 & -15.3 & 1.7 & -1.7 & -1.1 & 2.5 & 1.1 & -2.5 \\ 10.6 & -5.6 & -6.5 & -0.6 & 2.6 & 0.9 & -1.4 & 2.4 \\ 0.4 & -2.3 & 1.2 & -1.7 & 2.3 & -0.5 & 0.1 & -0.1 \\ 3.2 & -0.7 & -0.9 & 2.6 & -1.1 & 1.5 & -1.8 & 0.2 \end{pmatrix}$$

- Quantization and rounding

$$q(\hat{\mathsf{B}}) = \begin{pmatrix} 7 & 17 & -2 & 1 & 1 & 0 & 0 & 0 \\ 9 & 16 & -2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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

More than 75% entries are zero (notice their placement)

Encoding

- Zigzag scan

$$q(\hat{\mathrm{B}}) = \begin{pmatrix} 7 & 17 & -2 & 1 & 1 & 0 & 0 & 0 \\ 9 & 16 & -2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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

[**7** 17 9 3 16 -2 1 -2 1 5 0 -1 1 1 1 0 0 0 0 -1 EOB]

Lets try to reconstruct

Reconstruction: Decoding + Dequantization

[**7** 17 9 3 16 -2 1 -2 1 5 0 -1 1 1 1 0 0 0 0 -1 EOB]

$$\begin{pmatrix} 7 & 17 & -2 & 1 & 1 & 0 & 0 & 0 \\ 9 & 16 & -2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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

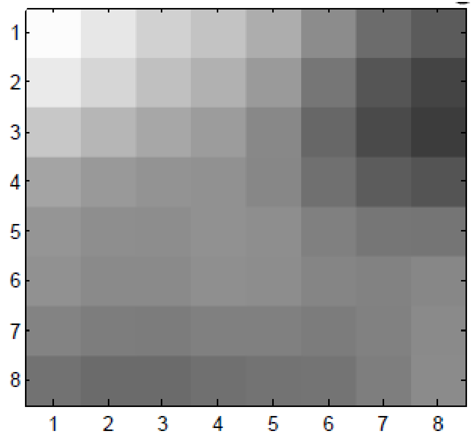
- Dequantization

$$\tilde{B} = \begin{pmatrix} 112 & 187 & -20 & 16 & 24 & 0 & 0 & 0 \\ 108 & 192 & -28 & 19 & 0 & 0 & 0 & 0 \\ 42 & 13 & 16 & 0 & 0 & 0 & 0 & 0 \\ 70 & -17 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Decoding

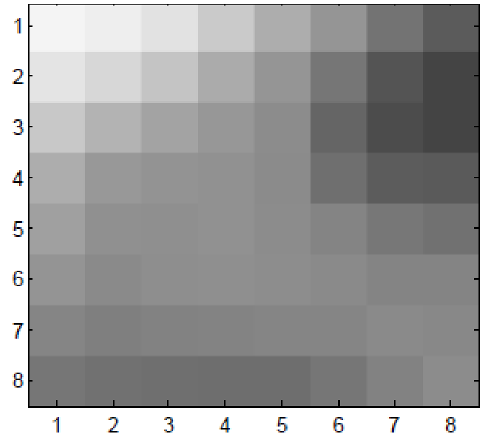
- Compute IDCT and add 127

$$\begin{pmatrix} 254 & 233 & 211 & 197 & 175 & 142 & 110 & 93 \\ 236 & 216 & 194 & 179 & 156 & 120 & 87 & 70 \\ 201 & 184 & 169 & 158 & 138 & 105 & 76 & 62 \\ 166 & 155 & 149 & 147 & 137 & 114 & 94 & 86 \\ 151 & 144 & 143 & 147 & 144 & 130 & 120 & 119 \\ 147 & 140 & 140 & 145 & 143 & 135 & 132 & 137 \\ 133 & 127 & 126 & 130 & 130 & 126 & 131 & 140 \\ 116 & 109 & 109 & 114 & 117 & 118 & 128 & 141 \end{pmatrix}$$



- Compare with original

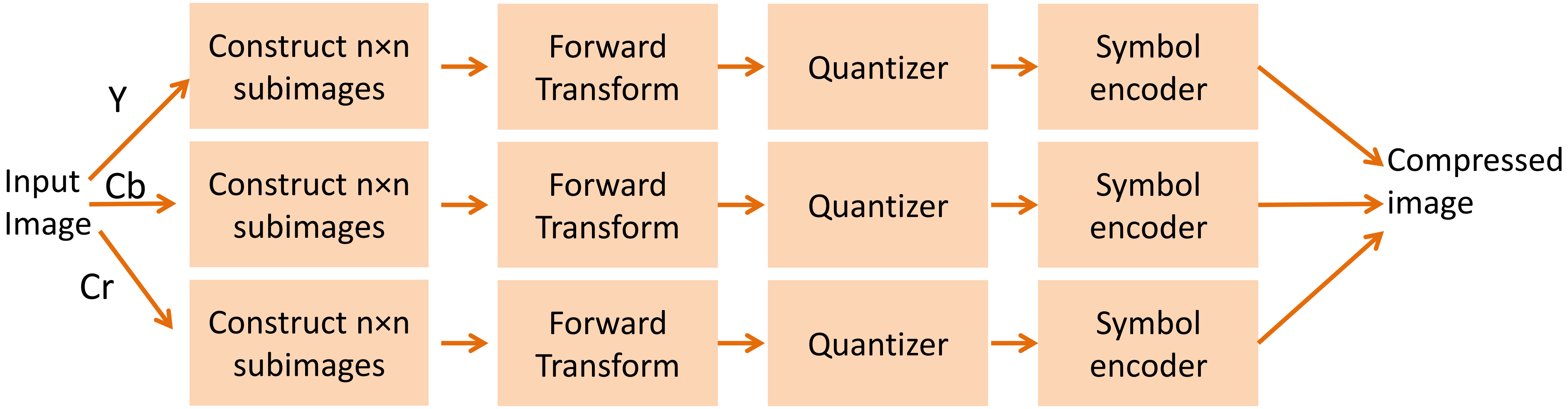
$$\begin{pmatrix} 245 & 239 & 227 & 203 & 174 & 150 & 116 & 92 \\ 229 & 216 & 197 & 172 & 150 & 119 & 85 & 69 \\ 201 & 180 & 164 & 152 & 141 & 102 & 77 & 69 \\ 174 & 153 & 148 & 146 & 140 & 112 & 93 & 91 \\ 161 & 145 & 144 & 146 & 141 & 133 & 120 & 114 \\ 149 & 139 & 143 & 144 & 142 & 139 & 133 & 133 \\ 134 & 128 & 131 & 132 & 134 & 134 & 139 & 137 \\ 119 & 114 & 112 & 111 & 111 & 119 & 131 & 141 \end{pmatrix}$$



Summary JPEG

- Divide into 8×8 subimages
- Compute DCT on each
- Quantize the coefficients
- Order coefficients in zigzag pattern
- Encode 1D sequence using run-length coding and Huffman coding

Color Images



Color Images

Y Cb Cr

- Different quantization matrices for chrominance and luminance
- Chroma subsampling (use reduced resolution of chroma channels)

Quantization matrices

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

Luminance

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

Chrominance

These matrices are scaled for higher compression!