Image **compression**

Theoretical size of a file representing

- a 5k x 4k colour photograph: 5000 x 4000 x 3 = 60 MB
- 1 min of UHD tv video: 3840 x 2160 x 3 x 24 x 60 = 36 GB

→ Compression is needed, either reversible (lossless) or irreversible (lossy)

APPROACHES:

A. Increase coding efficiency
B. Exploit spatial and temporal redundancy
C. Eliminate irrelevant or imperceptible information

→ Size can be reduced by a factor in the approx. range [5, 100]

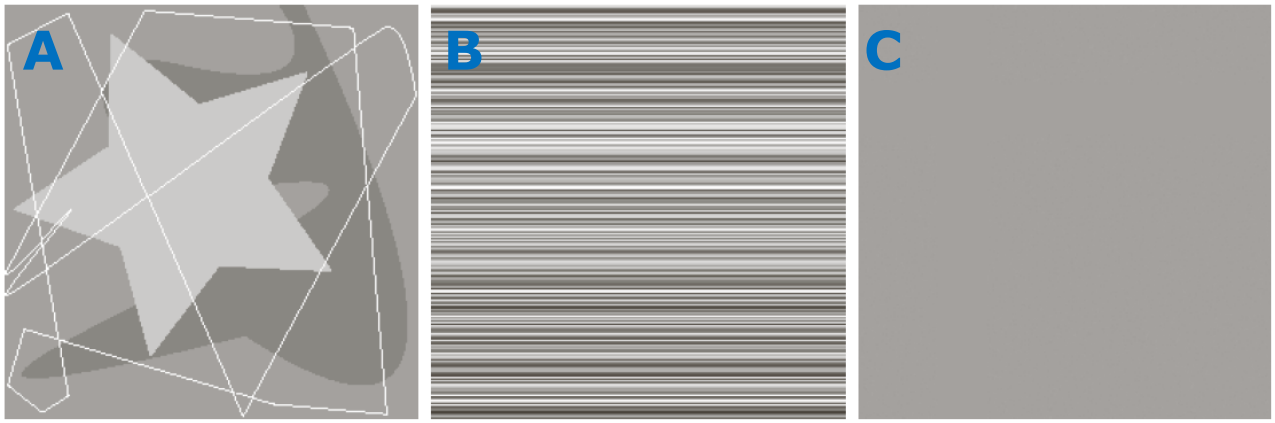


Image compression

Efficient coding: image **A** has four gray levels

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

Bit rate:

- Fixed-length code: $L_{avg} = \sum_k l(r_k)\, p_r(r_k) = 8$ bpp [bit per pixel]
- 4-levels fixed-length code: $L_{avg} = 2$ bpp
- "Code 2": $L_{avg} = 1.81$ bpp

Image compression

Spatial redundancy: in image **B**

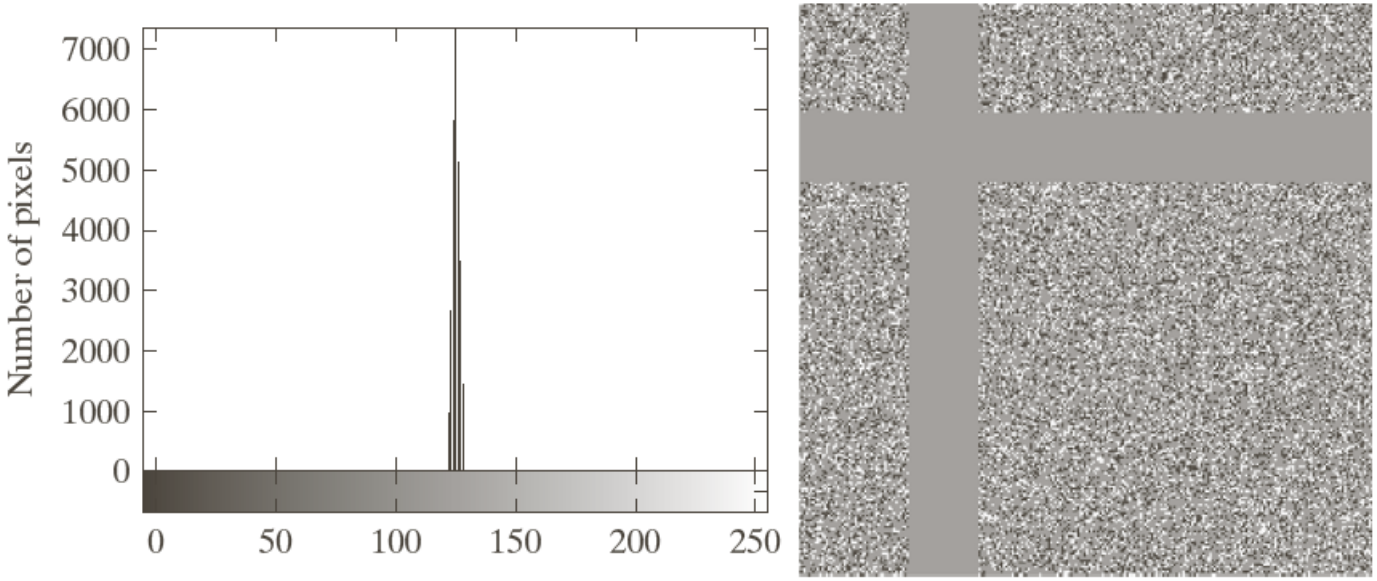
- rows are randomly ordered (vertical correlation = 0)
- columns are identical (horizontal correlation = 1)

→ A run-length code can be used, formed by a sequence of pairs *(graylevel, length)*

Image compression

Irrelevant or imperceptible information: image **C** could be represented by 1 byte only (average gray level)

But histogram equalization shows:



Its further coding becomes application-dependent

Note that if one decides to discard the "noise" and/or the "structure", the compression becomes **irreversible**

Image compression

Huffman coding (1952) [exploits *coding* redundancy]

For a given source, it yields the smallest number of code symbols per source symbols (among ***scalar*** coders; ***vector*** coders can do better)

Step 1: combine symbols into a tree of groups based on probability
Step 2: descend the tree and progressively generate each code

Example:

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				

Original source			Source reduction							
Symbol	Probability	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								

(assigning a '0' or a '1' to each level is irrelevant)

$L_{fixed\text{-}length}$ = 3 bpp

L_{avg} = 1x0.4 + 2x0.3 + 3x0.1 + 4x0.1 + 5x(0.06+0.04) = 2.2 bpp

while the entropy of a zero-memory source having this pdf is:

$$H = -\sum_k p_r(r_k) \log_2 p_r(r_k) = 2.14 \text{ bpp}$$

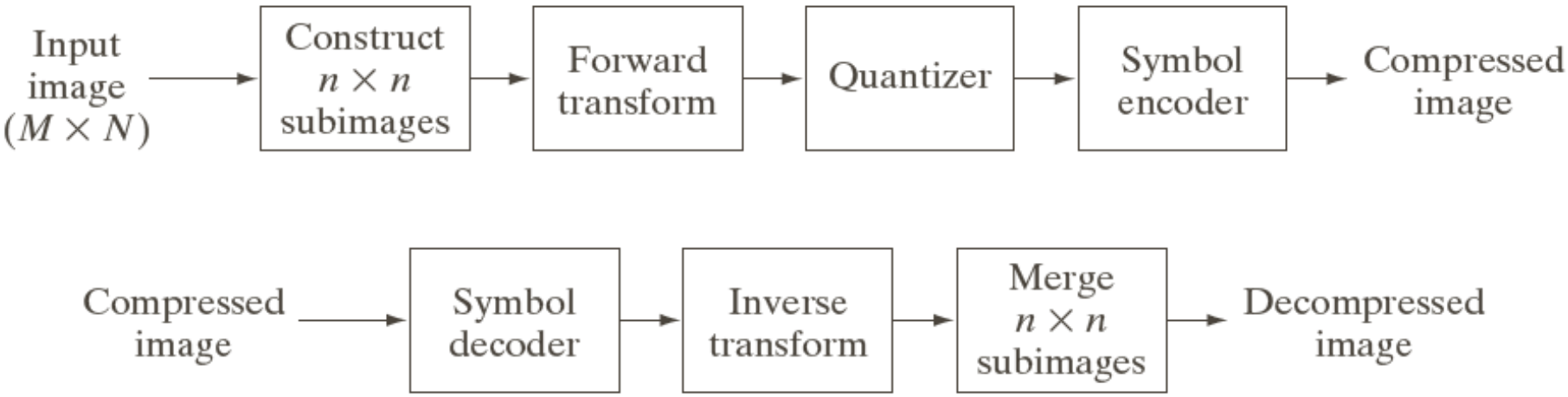
Note: code strings are *uniquely decodable*. E.g.: 010101000111011...

Note: The *code dictionary* has to be saved/sent too, or has to be fixed

Note: More sophisticated (but sometimes proprietary) coding tools exist, such as *Arithmetic coding*

Image compression

Block transform coding

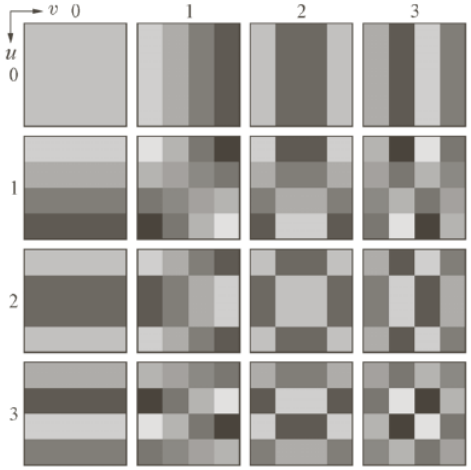


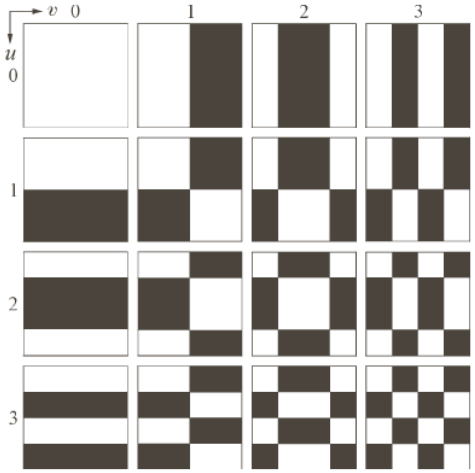
- Goal: pack most of the block information into the smallest number of transform coefficients
- Quantized transform → Intrinsically *lossy* coding
- Risk of *blocking artifacts*

Image compression

In principle, any unitary ($\boldsymbol{A}_N^{-1} = \boldsymbol{A}_N^{*T}$) (orthogonal if real) transform can be used: DFT, DCT, WHT, KLT...

e.g. Cosine and Walsh-Hadamard basis functions for n=4





- Periodicity artifact of DFT vs. DCT and WHT

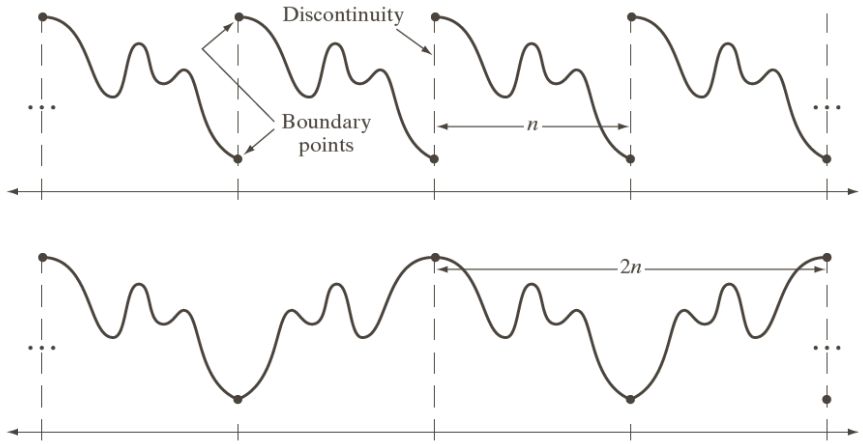


Image compression

Best transform?

Lena: reconstructed image and reconstruction error

- 8x8 blocks, largest 50% of the coefficients in each block are used

DFT (rmse=2.32) WHT (1.78) DCT (1.13)

Image compression

Best way to select the coefficients?

Reconstruction using

→

Largest magnitude subset of 12.5% of the coefficients

→

Maximum variance subset of 12.5% of the coefficients (ideal in the information-theoretic sense)

Image compression

Note: the *positions* of the selected coefficients must be stored/sent too →

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

Or, a standardized path can be followed for the selection, such as *zigzag ordering*

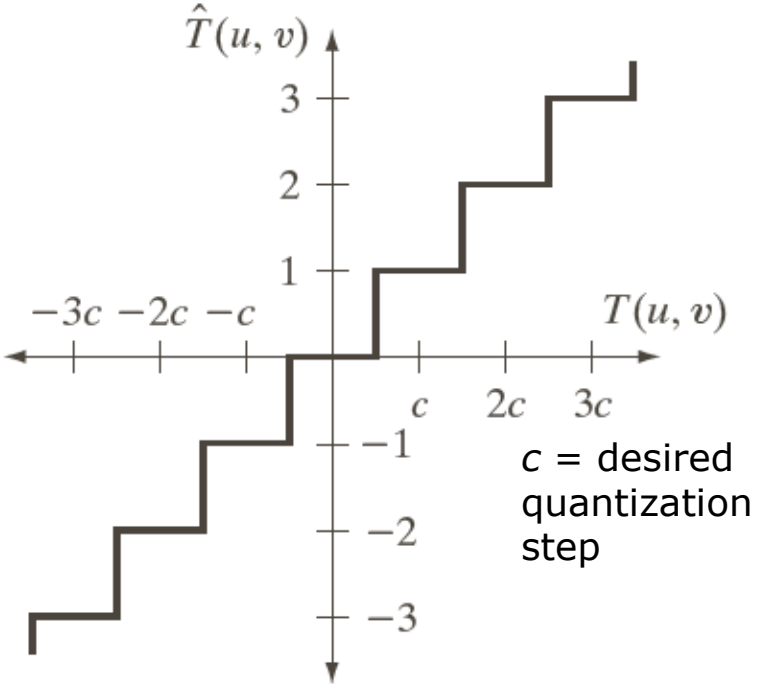
0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Image compression

Quantizer: coefficient selection in last slides was on/off

→ Can be refined using scaling + rounding

- Encoding: $\hat{T}(u,v) = \text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$
- Decoding: $\tilde{T}(u,v) = \hat{T}(u,v)Z(u,v)$



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

Typical quantization matrix $Z(u,v)$ for coefficients in the range [-128:127]

Image compression

The quantization matrix itself can be *scaled* to change its effects.
Compression ratios below are: 12, 19, 30; 49, 85, 182

FIGURE 8.31 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.30(b): (a) **Z**, (b) 2**Z**, (c) 4**Z**, (d) 8**Z**, (e) 16**Z**, and (f) 32**Z**.

Image compression: **JPEG**

JPEG baseline coding (lossy; a lossless version of JPEG exists too)

8 x 8 image block:

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

shifted by -128:

(to enable using signed integer format [-128,127] for all data)

−76	−73	−67	−62	−58	−67	−64	−55
−65	−69	−62	−38	−19	−43	−59	−56
−66	−69	−60	−15	16	−24	−62	−55
−65	−70	−57	−6	26	−22	−58	−59
−61	−67	−60	−24	−2	−40	−60	−58
−49	−63	−68	−58	−51	−65	−70	−53
−43	−57	−64	−69	−73	−67	−63	−45
−41	−49	−59	−60	−63	−52	−50	−34

Image compression: JPEG

JPEG baseline coding

DCT:

−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

Quantized DCT:

−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

Zig-zag ordered DCT:

[−26 −3 1 −3 −2 −6 2 −4 1 −4 1 1 5 0 2 0 0 −1 2 0 0 0 0 0 −1 −1 EOB]

Image compression: JPEG

JPEG baseline coding

- The difference between the DC term and the one of the previous block is calculated and Huffman-coded.
- For AC terms, a special Huffman coding allowing for runs of zeros is used

(-26 -*DCprev*) -3 1 -3 -2 -6 2 -4 ...

→ 1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001
001 100101 11100110 110110 0110 11110100 000 1010

Note: spaces are inserted here only for readability

Image compression: JPEG

JPEG baseline coding

(Was -415: risk of blocking artifact)

When decoding, the DCT block becomes (compare to original one)

−416	−33	−60	32	48	0	0	0
12	−24	−56	0	0	0	0	0
−42	13	80	−24	−40	0	0	0
−56	17	44	−29	0	0	0	0
18	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

and a reconstruction error is generated in the data domain

−6	−9	−6	2	11	−1	−6	−5
7	4	−1	1	11	−3	−5	3
2	9	−2	−6	−3	−12	−14	9
−6	7	0	−4	−5	−9	−7	1
−7	8	4	−1	6	4	3	−2
3	8	4	−4	2	6	1	1
2	2	5	−1	−6	0	−2	5
−6	−2	2	6	−4	−4	−6	10

Image compression: JPEG

JPEG baseline coding

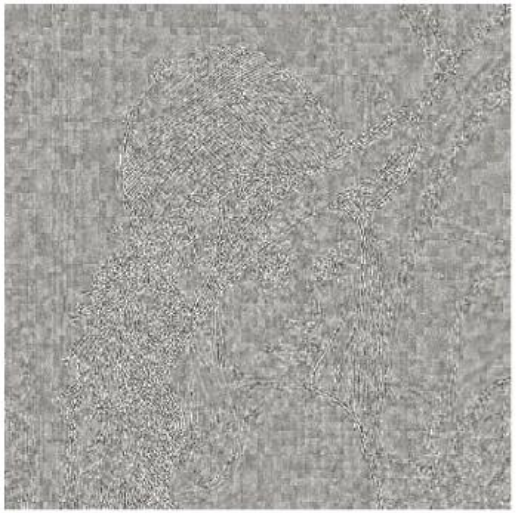
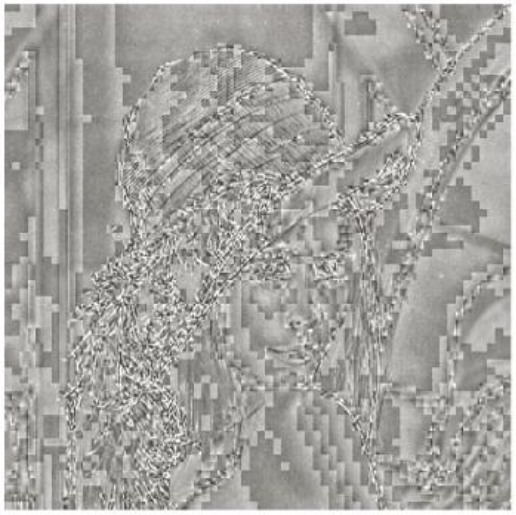
	reconstructed image	error	detail
25:1			
52:1			

Image compression: JPEG

JPEG baseline coding of **COLOR** images

- Avoid separate coding of RGB channels

(correlated data) → convert to *luma+chroma:* **YCbCr**

- Chroma components are often downsampled

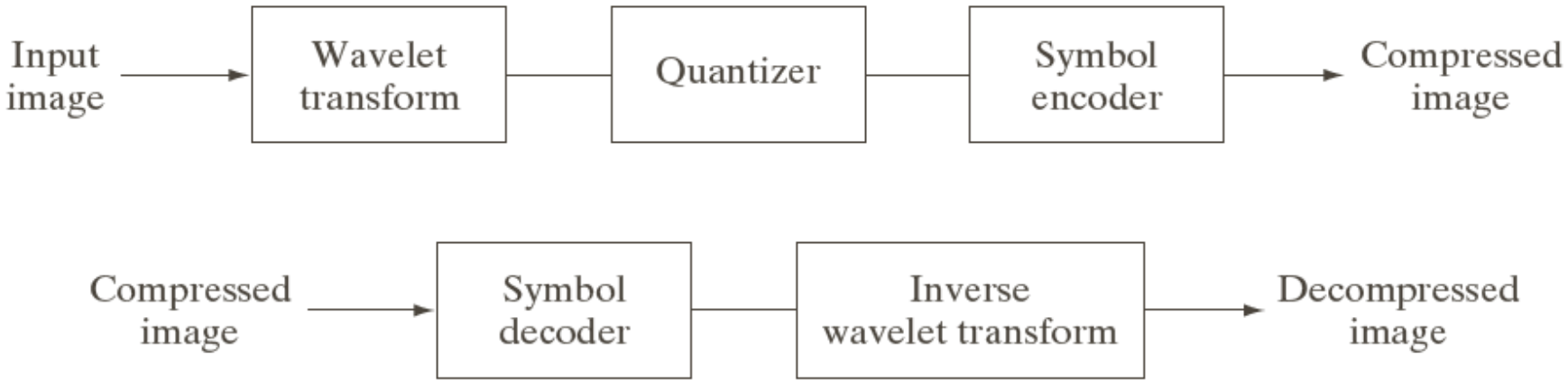
Quant=1; 4:4:4

Quant=1; 4:2:0

Quant=4; 4:2:0

Image compression

Wavelet coding (lossy)



- No need to operate on blocks, since wavelet transform is inherently local

  → no blocking artifacts
- Images can however be split into a few *tiles*, to permit independent coding/decoding of image portions

Image compression

Wavelet coding (lossy)

Haar

Daubechies

JPEG 2000
(lossy)

25:1

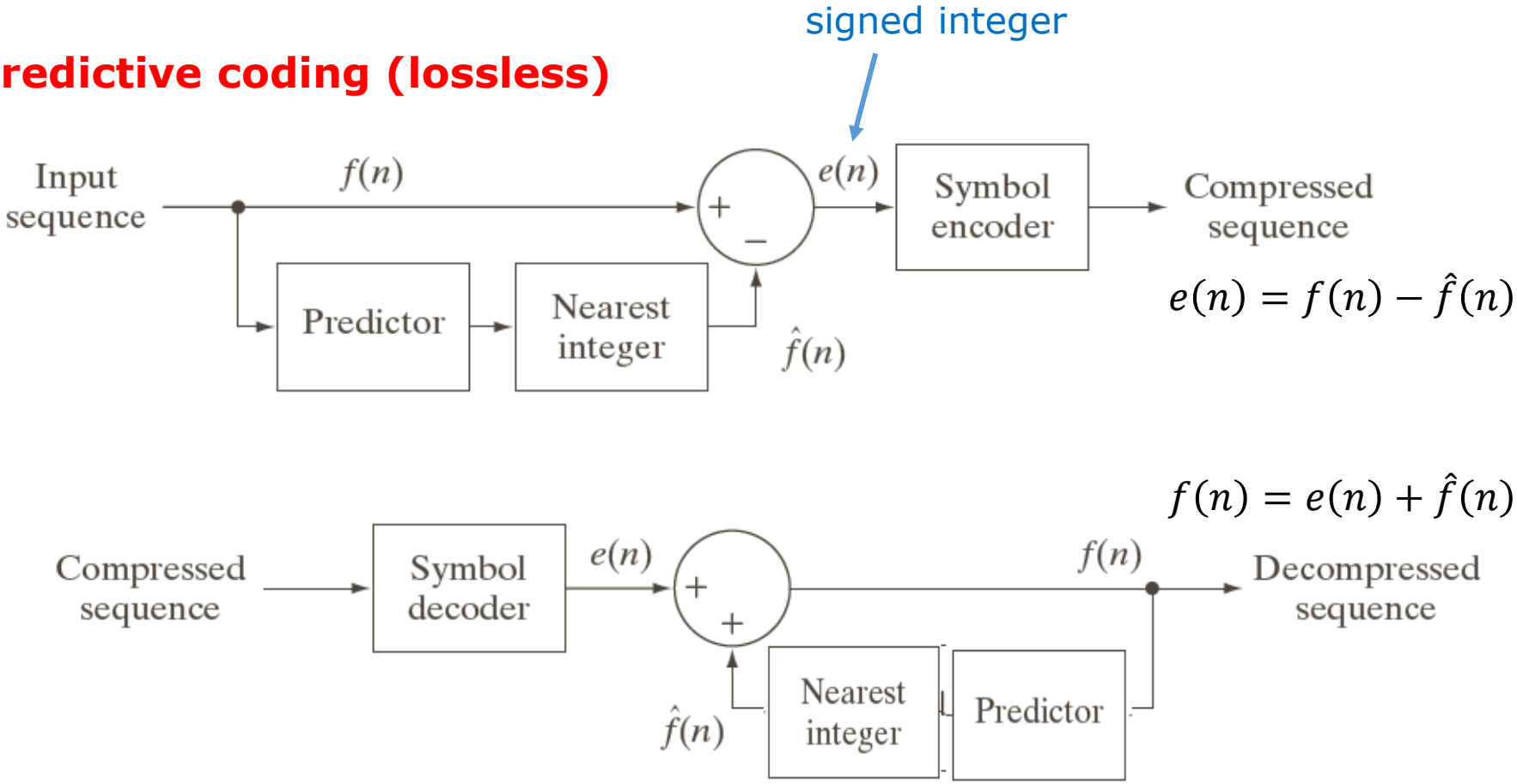
52:1

75:1

105:1

Image compression

Predictive coding (lossless)



- 1-D signals; easily extended to 2-D or 3-D data, provided a causality description (ordering) is defined
- *Optimal* predictors can be derived basing on the correlation matrix of the signal

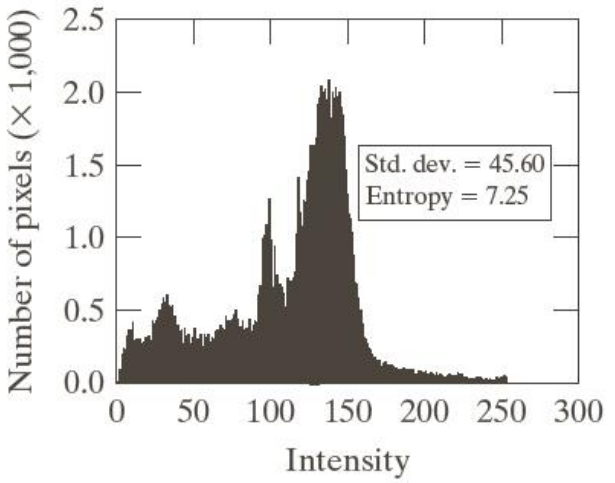
Image compression

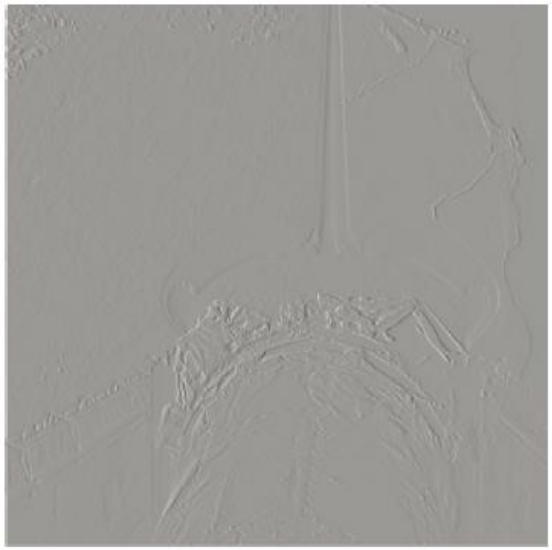
Simplest 2-D predictor

$$\hat{f}(x,y) = \text{round}[\alpha\ f(x,y-1)]$$

- Note: prediction error has 1 more bit than original data. However...
- Error distribution is approx. Laplacian

$\alpha = 1$





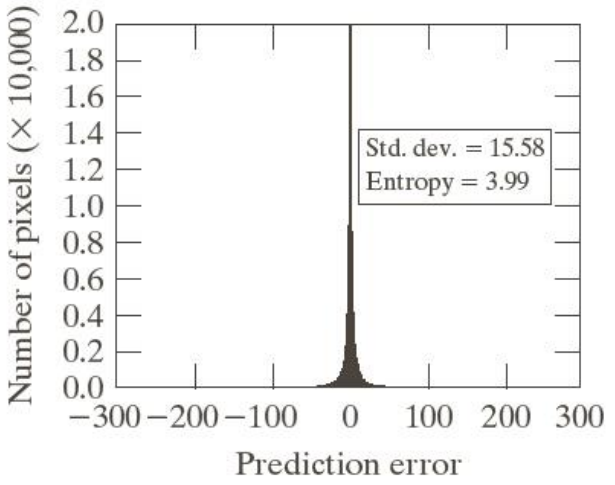
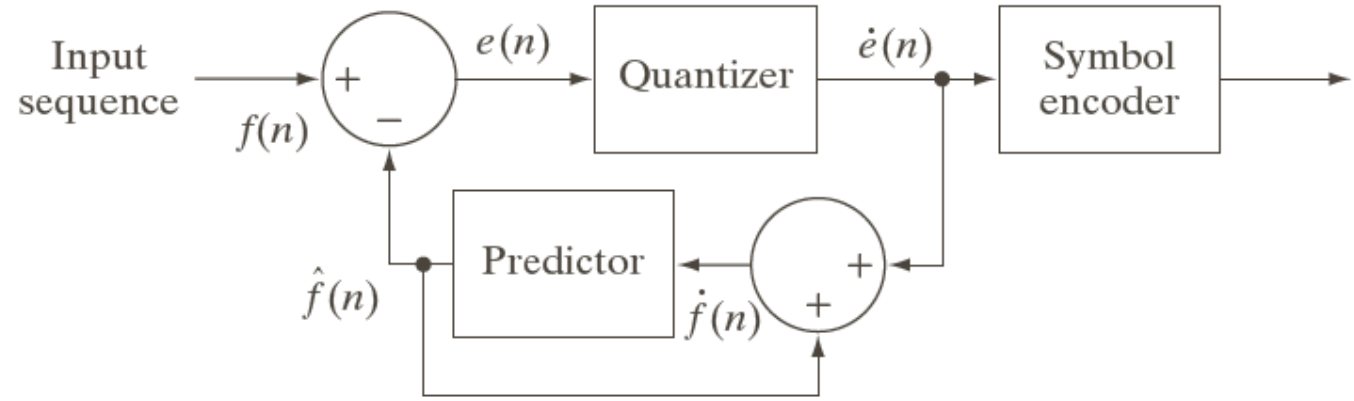


Image compression

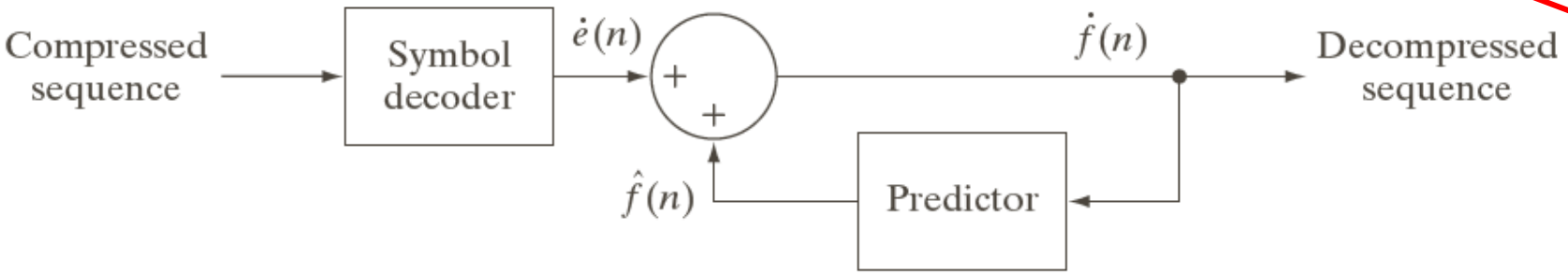
→ dip14_2_LossyPredCodec.m

Predictive coding (lossy)



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

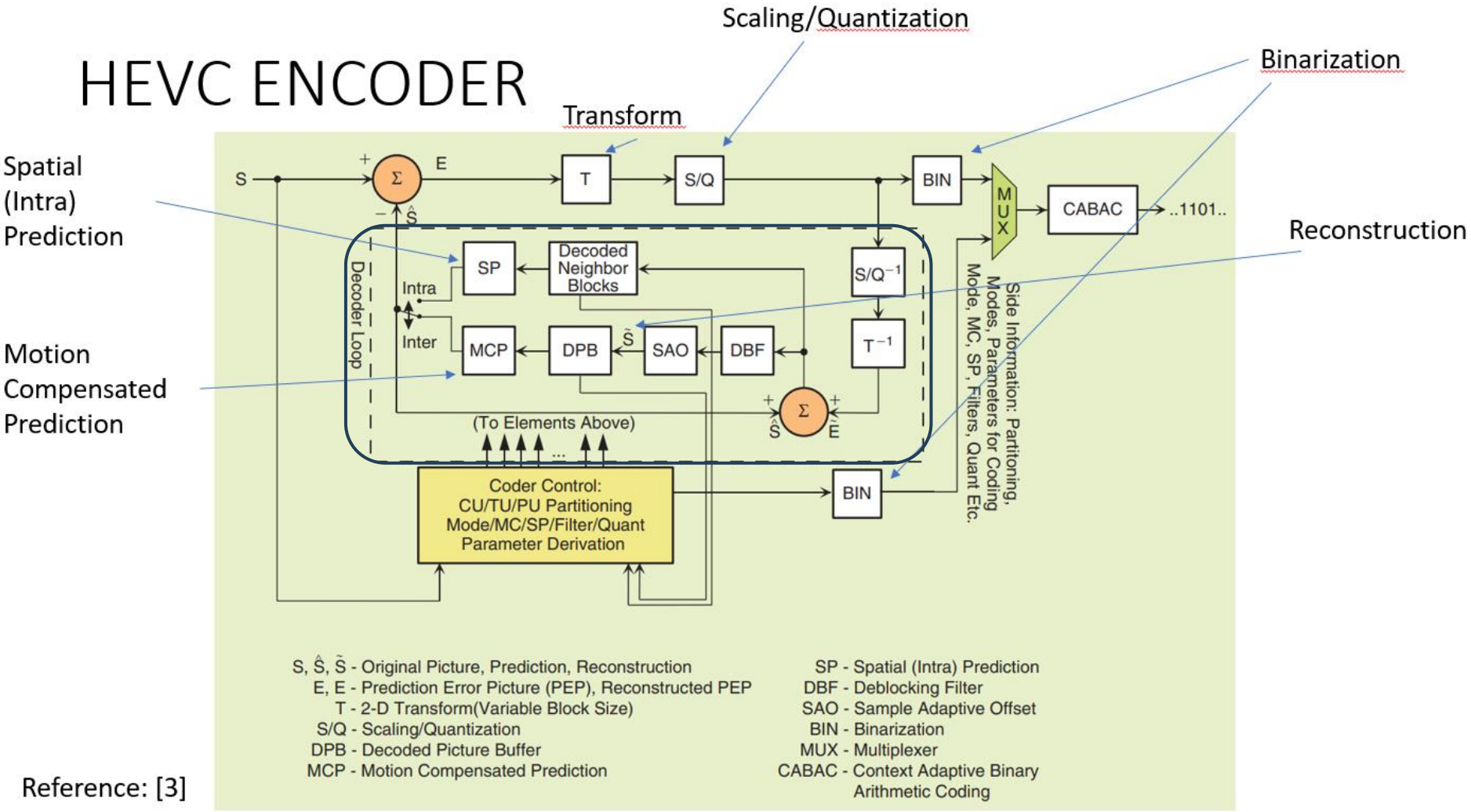
$$f(n) = e(n) + \hat{f}(n)$$



- Decoder is the same as in the lossless case, apart from rounding
- Predictor loop in the encoder avoids error buildup at the decoder

→ For both encoder and decoder, $\dot{f}(n) = \dot{e}(n) + \hat{f}(n)$

Image compression



Reference: [3]

Image compression

Standard and **popular** image and video formats

Image Compression
Standards, Formats, and Containers

Still Image

Binary

CCITT Group 3
CCITT Group 4
JBIG (or JBIG1)
JBIG2

TIFF

Continuous Tone

JPEG
JPEG-LS
JPEG-2000

BMP
GIF
PDF
PNG
TIFF

Video

DV
H.261
H.262
H.263
H.264
MPEG-1
MPEG-2
MPEG-4
MPEG-4 AVC

AVS
HDV
M-JPEG
QuickTime
VC-1 (or WMV9)

High Efficiency Video Coding (HEVC) defined jointly by ISO/IEC (as ISO/IEC 23008-2) and ITU-T (as **H. 265**) and also referred to as **MPEG-H Part 2**.

- Jpeg XT (High Dynamic Range & Wide Color Gamut)
- Jpeg Pleno (Plenoptic aka Lightfield imaging)
- … and of course: Jpeg AI

--> https://jpeg.org