EIE4512 - Digital Image Processing
Image Compression

Zhen Li
lizhen@cuhk.edu.cn
School of Science and Engineering
The Chinese University of Hong Kong, Shen Zhen
April 16-18, 2019
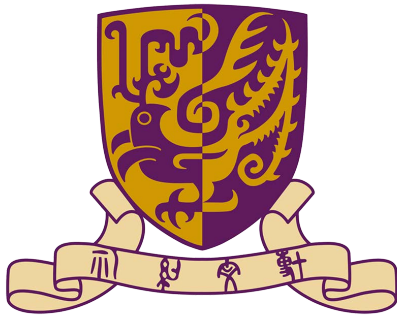
香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Image Compression

- The goal of image compression is to reduce the amount of data required to represent a digital image.
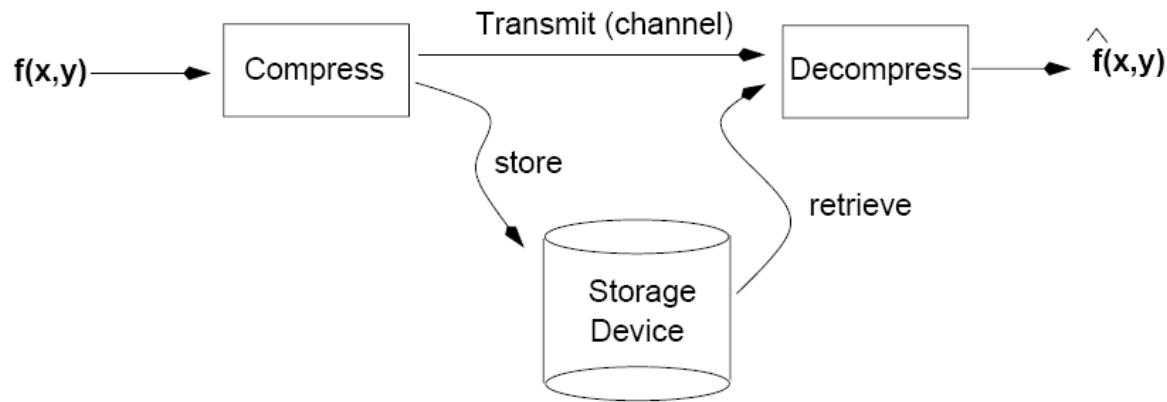
Image Compression (cont'd)

- **Lossless**
  - Information preserving
  - Low compression ratios

- **Lossy**
  - Information loss
  - High compression ratios

Trade-off: information loss **vs** compression ratio

Data ≠ Information

- Data and information are <u>not</u> synonymous terms!

- Data is the means by which information is conveyed.

- Data compression aims to <u>reduce</u> the amount of data while <u>preserving</u> as much information as possible.
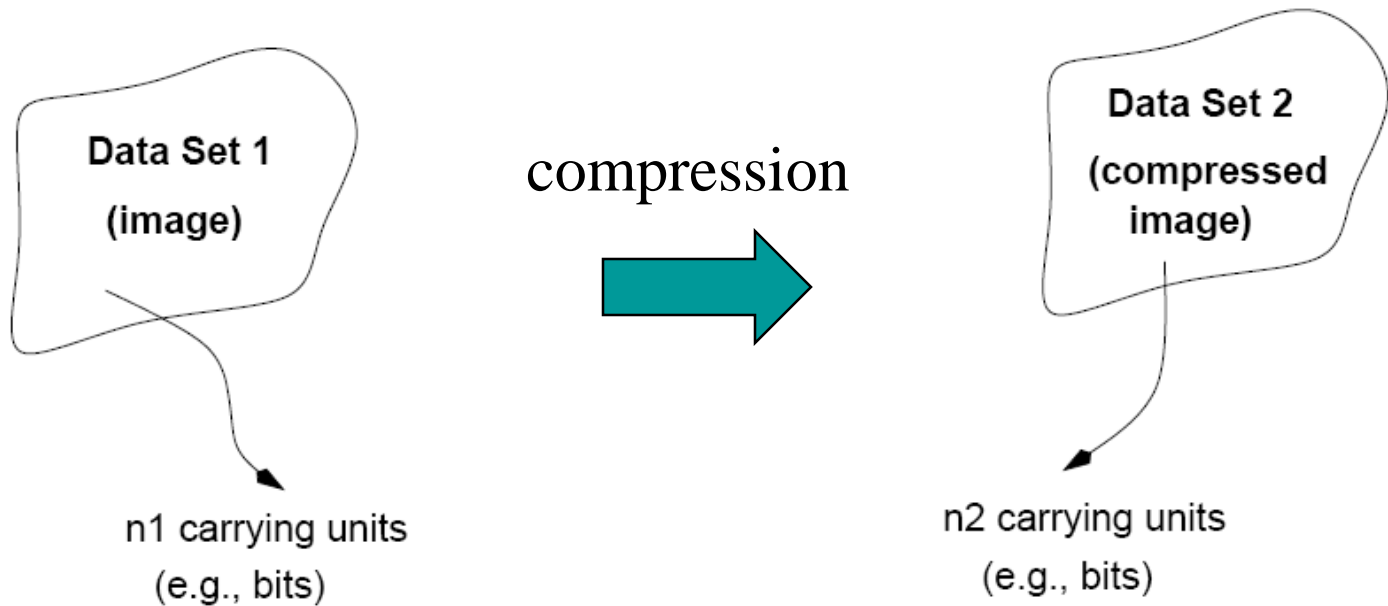
Data vs Information (cont'd)

- The same information can be represented by different amount of data – for example:

Ex1: *Your wife, Helen, will meet you at Logan Airport in Boston at 5 minutes past 6:00 pm tomorrow night*

Ex2: *Your wife will meet you at Logan Airport at 5 minutes past 6:00 pm tomorrow night*

Ex3: *Helen will meet you at Logan at 6:05 pm tomorrow night.*

Compression Ratio

Data Set 1
(image)

compression

Data Set 2
(compressed image)

n1 carrying units
(e.g., bits)

n2 carrying units
(e.g., bits)

Compression ratio: $\quad C_R = \dfrac{n_1}{n_2}$

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 Data Redundancy

(1) Coding Redundancy

(2) Interpixel Redundancy

(3) Psychovisual Redundancy

- Data 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 - Definitions (cont'd)

N x M image

$\mathbf{r_k}$: k-th gray level
$\mathbf{l(r_k)}$: # of bits for r_k
$\mathbf{P(r_k)}$: probability of r_k

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}

Expected value: $E(X) = \sum_{x} xP(X = x)$

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

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

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

Table 6.1 Variable-Length Coding Example — variable length

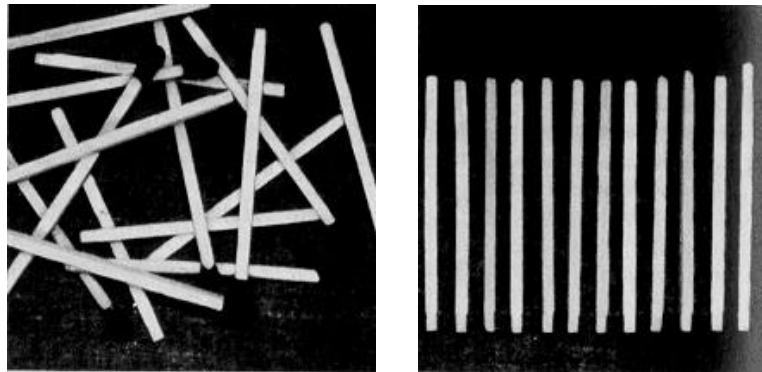$$L_{avg} = \sum_{k=0}^{7} l(r_k)P(r_k) = 2.7 \text{ bits}$$

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

Total number of bits: 2.7NM
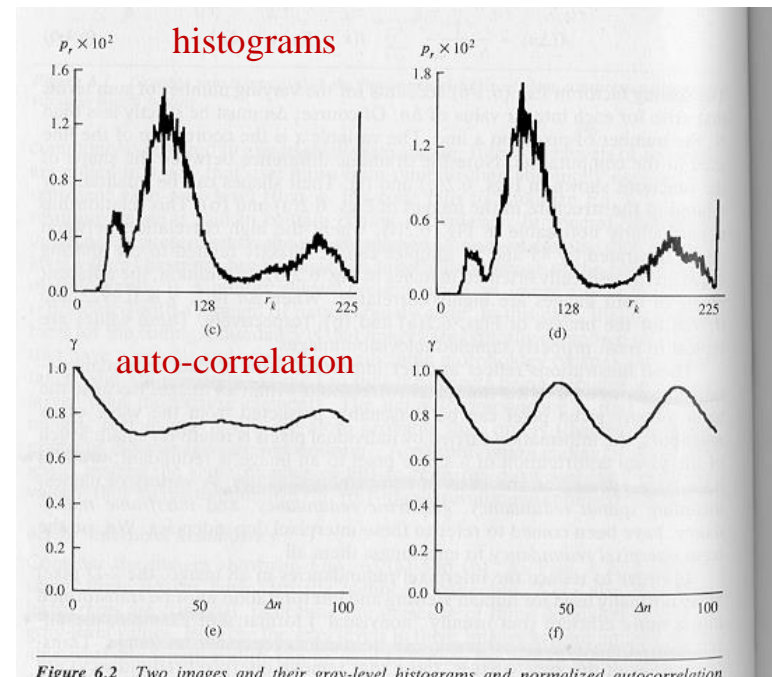
$$R_D = 1 - \frac{1}{1.11} = 0.099$$

Interpixel redundancy

- Interpixel redundancy implies that pixel values are correlated (i.e., a pixel value can be reasonably predicted by its neighbors).

histograms

auto-correlation

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

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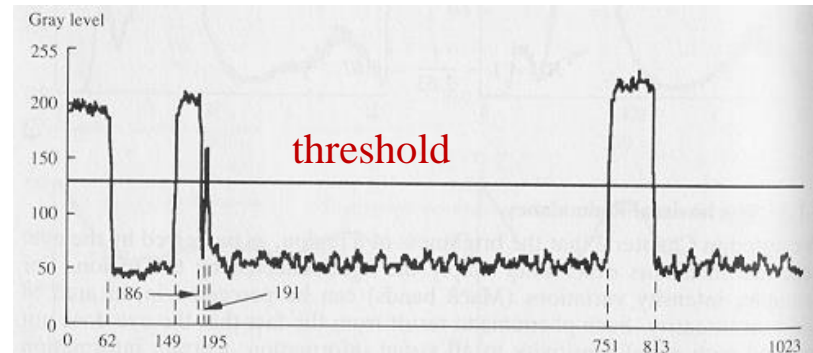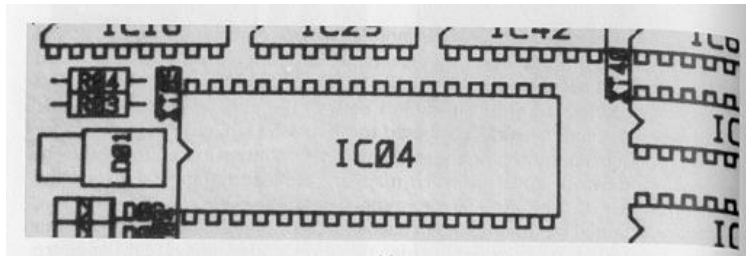
auto-correlation: f(x)=g(x)

Interpixel redundancy (cont'd)

- To reduce interpixel redundancy, some kind of transformation must be applied on the data (e.g., thresholding, DFT, DWT)

Example:

original

thresholded

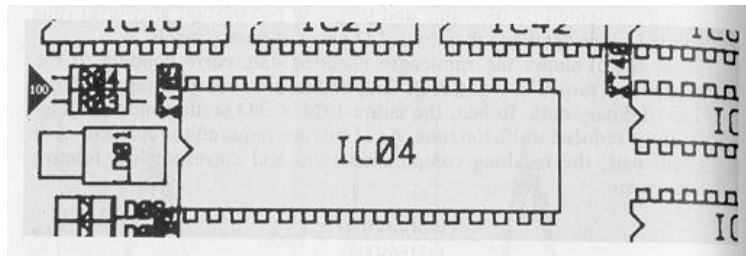11 ……………0000…………………...11…...000…..

Run-length encoding:

(1,63) (0,87) (1,37) (0,5) (1,4) (0, 556) (1,62) (0,210)

Using 11 bits/pair: (1+10) bits/pair

88 bits are required (compared to 1024 !!)

Psychovisual redundancy

- The human eye is more sensitive to the lower frequencies than to the higher frequencies in the visual spectrum.

- <u>Idea:</u> discard data that is perceptually insignificant!

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

Measuring Information

- A key question in image compression is:

    "What is the minimum amount of data that is sufficient to describe completely an image without loss of information?"

- How do we measure the information content of an image?

Measuring Information (cont'd)

- We assume that information generation is a probabilistic process.

- <u>Idea</u>: associate information with probability!

  A random event E with probability $P(E)$ contains:

  $$I(E) = log(\frac{1}{P(E)}) = -log(P(E)) \text{ units of information}$$

  <u>Note</u>: I(E)=0 when P(E)=1

How much information
does a pixel value contain?

- Suppose that gray level values are generated by a random process, then r_k contains:

$$I(r_k) = -log(P(r_k))$$ units of information!

(assume statistically independent random events)

How much information does an image contain?

- Average information content of an image:

$$E = \sum_{k=0}^{L-1} I(r_k) P(r_k)$$

using $I(r_k) = -log(P(r_k))$

Entropy: $H = -\sum_{k=0}^{L-1} P(r_k) log(P(r_k))$ units/pixel
(e.g., bits/pixel)

Redundancy

- **Redundancy:** $$R = L_{avg} - H$$
  (data vs info)

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

Note: if $L_{avg} = H$, then R=0 (no redundancy)

Entropy Estimation

- It is not easy to estimate H reliably!

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

Gray Level	Count	Probability
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8

Entropy Estimation (cont'd)

- First order estimate of H:

$$H = -\sum_{k=0}^{3} P(r_k) log(P(r_k)) = 1.81 \text{ bits/pixel}$$

$$L_{avg} = 8 \text{ bits/pixel} \quad R = L_{avg} - H$$

The first-order estimate provides only a <u>lower-bound</u> on the compression that can be achieved.

Estimating Entropy (cont'd)

- Second order estimate of H:
  - Use relative frequencies of <u>pixel blocks</u> :

image

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

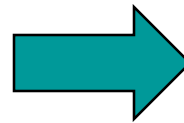| Gray Level Pair	Count	Probability
(21, 21)	8	1/4
(21, 95)	4	1/8
(95, 169)	4	1/8
(169, 243)	4	1/8
(243, 243)	8	1/4
(243, 21)	4	1/8

$$H = 2.5/2 = 1.25 \text{ bits/pixel}$$

Differences in Entropy Estimates

- Differences between higher-order estimates of entropy and the first-order estimate indicate the presence of interpixel redundancy!

- Need to apply some transformation to deal with interpixel redundancy!

Differences in Entropy Estimates (cont'd)

- Example: consider pixel differences

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0

Gray Level or Difference	Count	Probability
0	16	1/2
21	4	1/8
74	12	3/8

Differences in Entropy Estimates (cont'd)

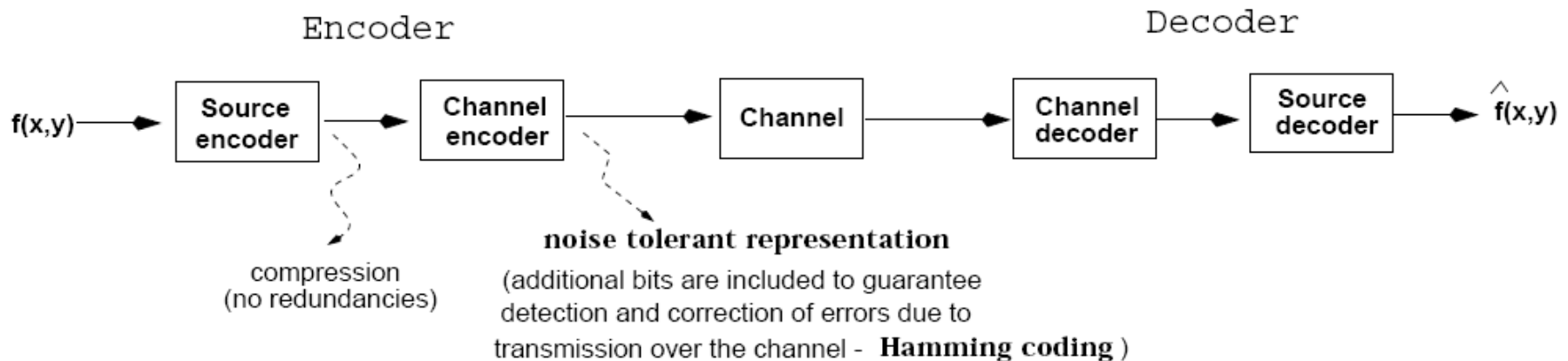- What is the entropy of the <u>pixel differences</u> image?

$$H = -\sum_{k=0}^{2} P(r_k) log(P(r_k)) = 1.41 \text{ bits/pixel}$$

(better than the entropy of the original image H=1.81)

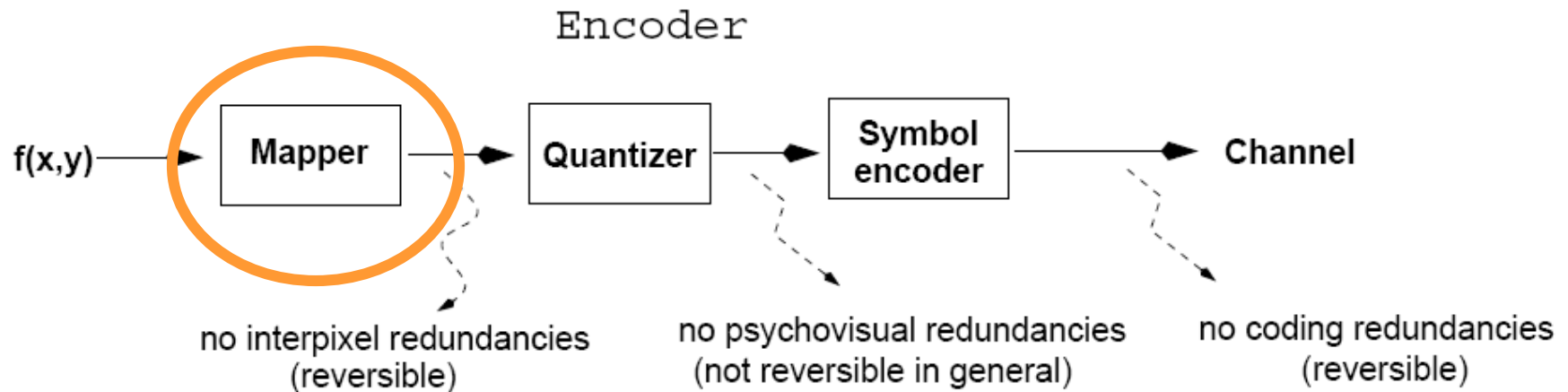- An even better transformation should be possible since the second order entropy estimate is lower:

1.41 bits/pixel > 1.25 bits/pixel

Image Compression Model

We will focus on the Source Encoder/Decoder only.

Image Compression Model (cont'd)

Encoder

f(x,y) → **Mapper** → **Quantizer** → **Symbol encoder** → **Channel**

no interpixel redundancies (reversible) no psychovisual redundancies (not reversible in general) no coding redundancies (reversible)
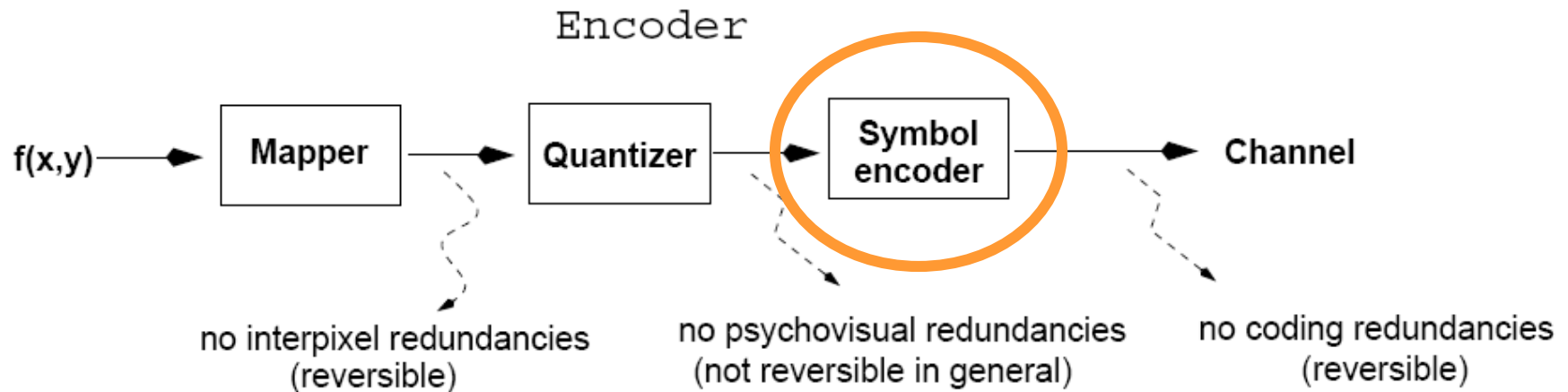
•

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

Image Compression Model (cont'd)
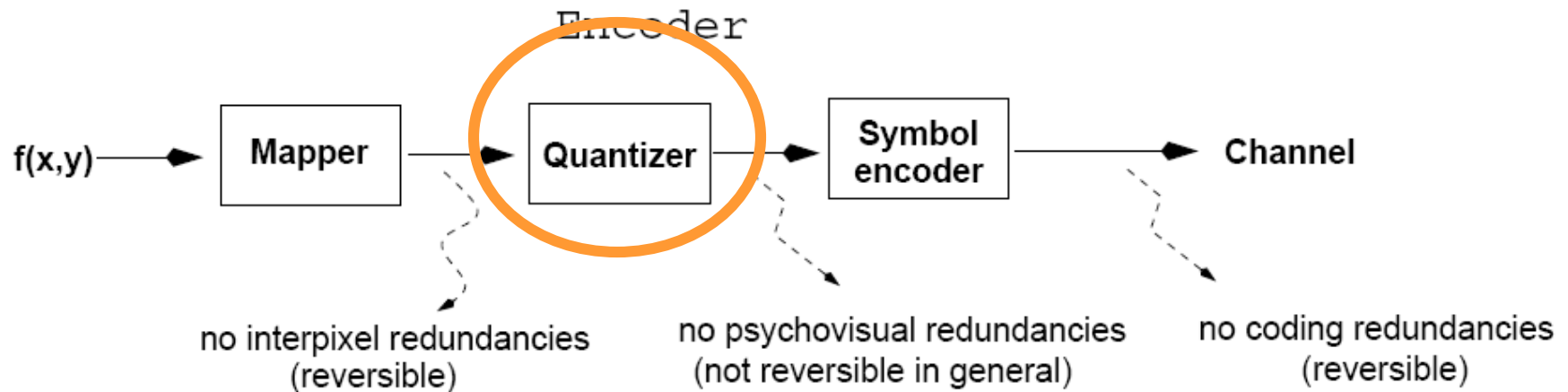
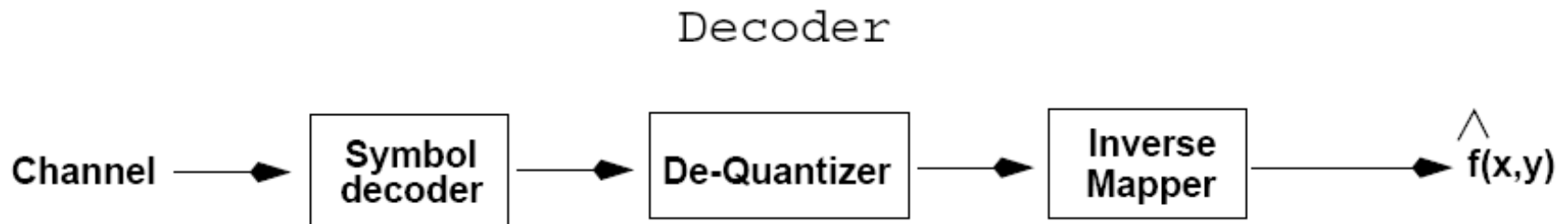- **Quantizer:** quantizes the data to account for psychovisual redundancies.

Image Compression Model (cont'd)

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

Image Compression Models (cont'd)

Decoder

Channel → **Symbol decoder** → **De-Quantizer** → **Inverse Mapper** → $\hat{f}(x,y)$

- The decoder applies the inverse steps.

- Note that quantization is irreversible in general.

Fidelity Criteria

f(x,y) ──→ [Compress] ──→ g(x,y) ──→ [Decompress] ──→ $\hat{f}(x,y)$
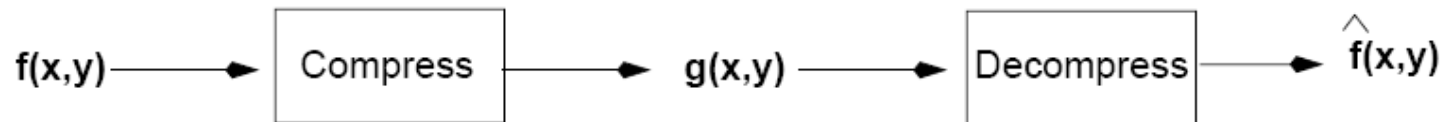
- How close is $f(x,y)$ to $\hat{f}(x,y)$?

- Criteria
  - <u>Subjective:</u> based on human observers
  - <u>Objective:</u> mathematically defined criteria

Subjective Fidelity Criteria

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

Objective Fidelity Criteria

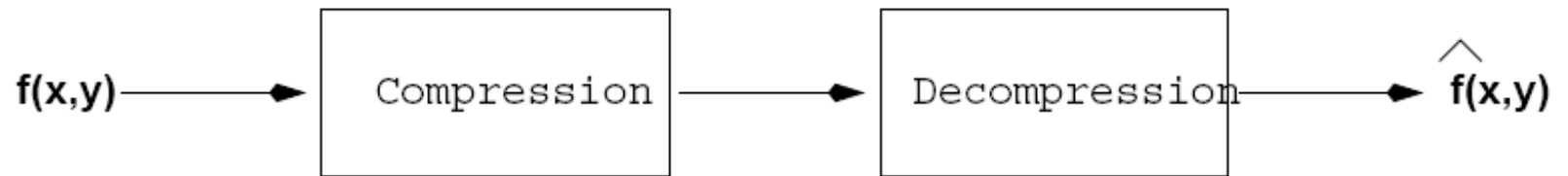- Root mean square error (RMS)

$$e_{rms} = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y) - f(x,y))^2}$$

- Mean-square signal-to-noise ratio (SNR)

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y))^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x,y) - f(x,y))^2}$$

Lossless Compression

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

Taxonomy of Lossless Methods
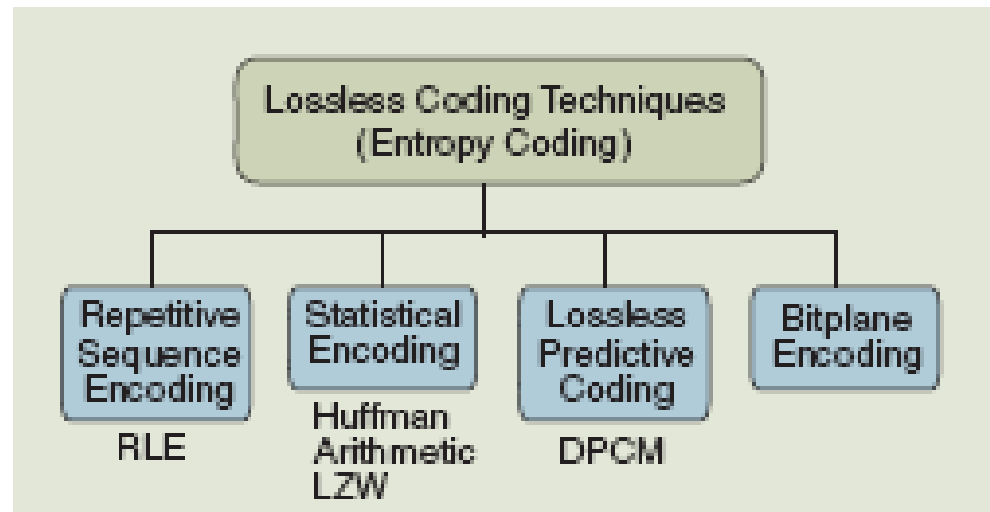
Huffman Coding
(addresses coding redundancy)
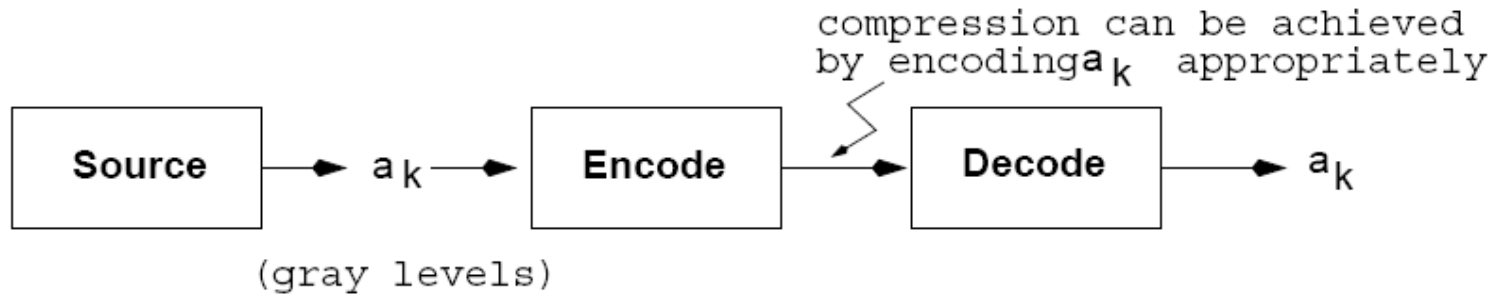
- A variable-length coding technique.
- Source symbols are encoded one at a time!
  - There is a one-to-one correspondence between source symbols and code words.
- Optimal code - minimizes code word length per source symbol.

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				

Huffman Coding (cont'd)

- *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:

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

$3 \times 0.1 + 1 \times 0.4 + 5 \times 0.06 + 4 \times 0.1 + 5 \times 0.04 + 2 \times 0.3 = 2.2 \text{ bits/symbol}$

- L_{avg} assuming binary coding:

6 symbols, we need a 3-bit code

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

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

Huffman Coding/Decoding

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

- Decoding can be done unambiguously.
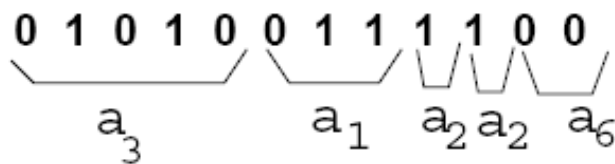
Original source		
Sym.	Prob.	Code
a_2	0.4	1
a_6	0.3	00
a_1	0.1	011
a_4	0.1	0100
a_3	0.06	01010
a_5	0.04	01011

Arithmetic (or Range) Coding
(addresses coding redundancy)

- Huffman coding encodes source symbols one at a time.

- Arithmetic coding encodes sequences of source symbols.

  – Slower than Huffman coding but can achieve better compression.

  – There is no one-to-one correspondence between source symbols and code words.

Arithmetic Coding (cont'd)

- Represent a sequence of source symbols by a sub-interval in $[0,1)$ which can be encoded using an arithmetic code.

<div align="center">

sub-interval   arithmetic code

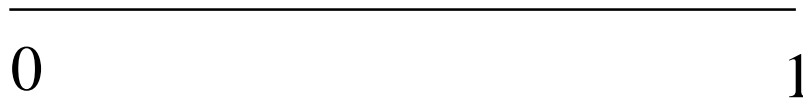$\alpha_1\ \alpha_2\ \alpha_3\ \alpha_3\ \alpha_4$ ➡ $[0.06752, 0.0688)$ ➡ 0.068

</div>

- Start with the interval $[0, 1)$
- As more symbols are encoded, a sub-interval is chosen to represent the message which keeps shrinking as the message increases.

Arithmetic Coding (cont'd)

Suppose there are four symbols α_1 α_2 α_3 α_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)

```
_____
0                                 1
```

2) Subdivide [0, 1) based on the probabilities of α_i
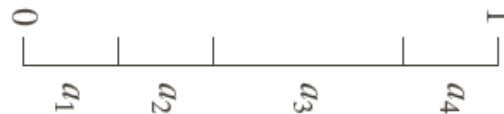
Initial Subinterval
[0.0, 0.2)
[0.2, 0.4)
[0.4, 0.8)
[0.8, 1.0)

3) Update interval by processing message

Example

Encode

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

[0.06752, 0.0688)

code: 0.068

(must be inside sub-interval)

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)
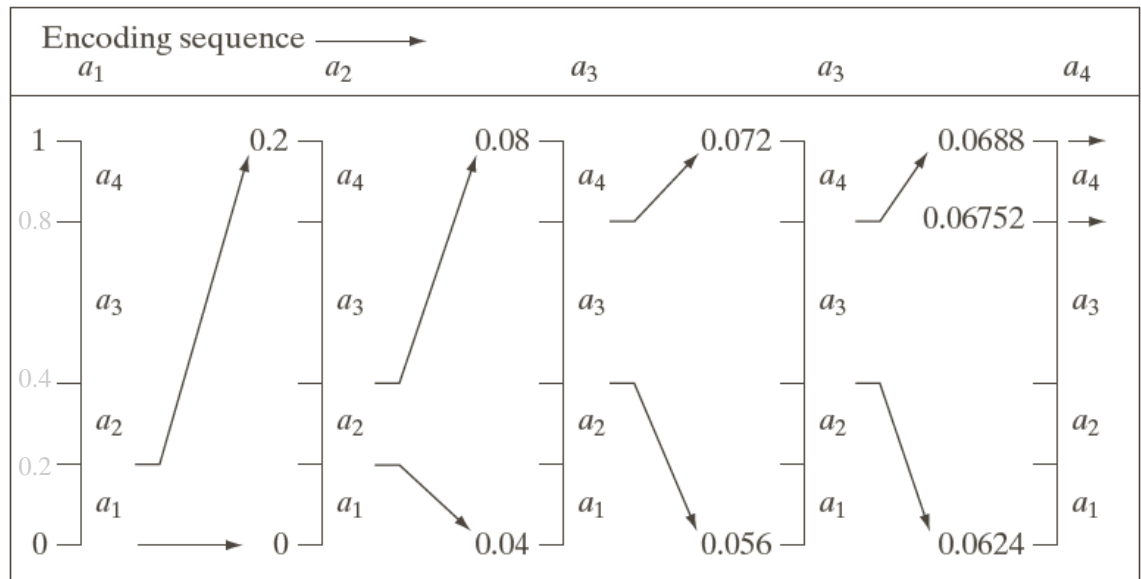
Example (cont'd)

- The message $\alpha_1\ \alpha_2\ \alpha_3\ \alpha_3\ \alpha_4$ is encoded using 3 decimal digits or $3/5 = 0.6$ decimal digits per source symbol.

- The entropy of this message is: $H = -\sum_{k=0}^{3} P(r_k)log(P(r_k))$

  $-(3 \times 0.2log_{10}(0.2)+0.4log_{10}(0.4))=0.5786$ digits/symbol

  <u>Note:</u> finite precision arithmetic might cause problems due to truncations!

Arithmetic Decoding

α_4

α_3

α_2

α_1

1.0

0.8

0.4

0.2

0.0

0.8

0.72

0.56

0.48

0.4

0.72

0.688

0.624

0.592

0.56

0.592

0.5856

0.5728

0.5664

0.56

0.5728

0.57152

0.56896

0.56768

0.5664
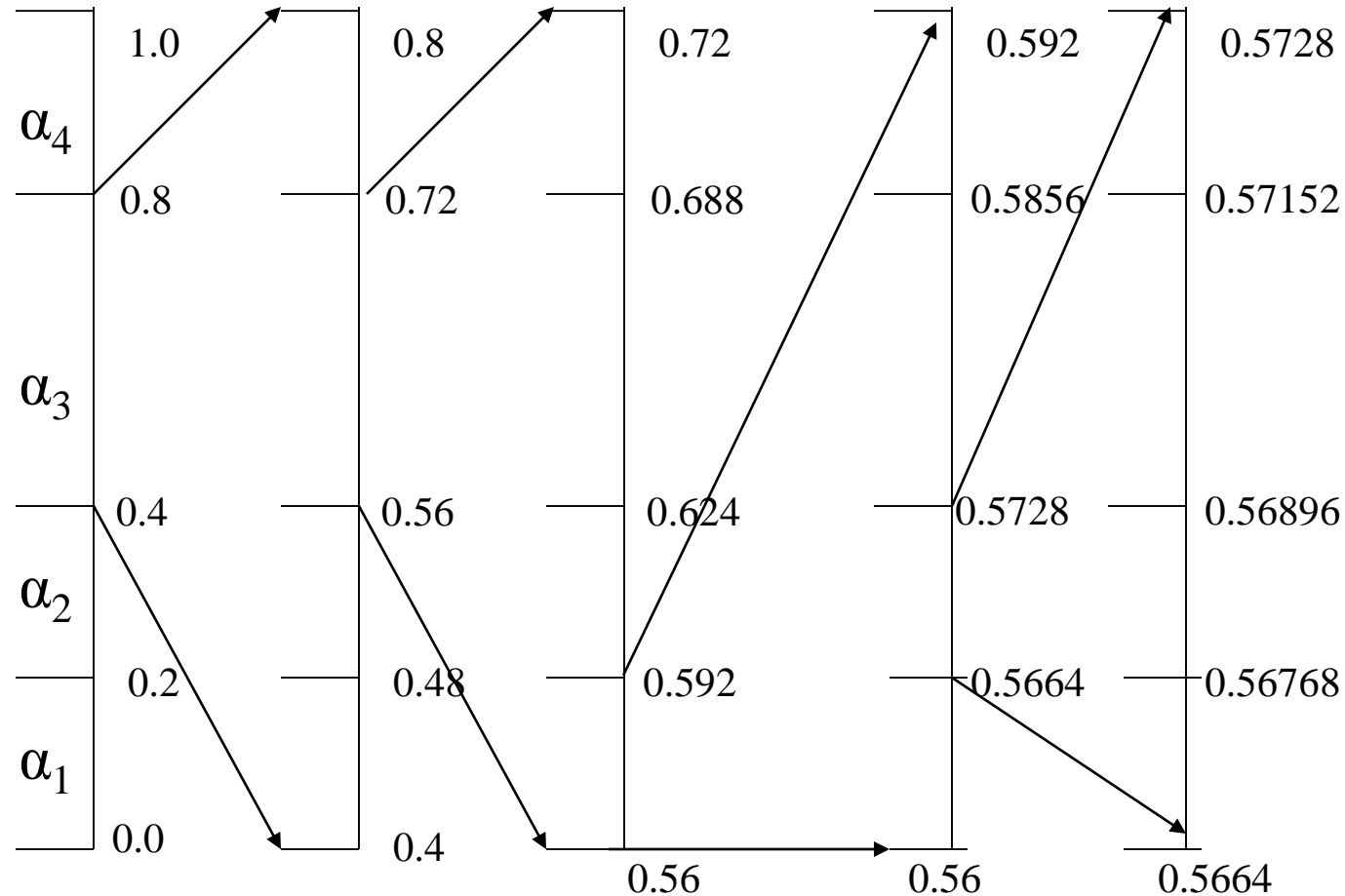
Decode 0.572

(sequence length=5)

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

LZW Coding
(addresses interpixel redundancy)

- Requires <u>no prior knowledge</u> 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)

Initial Dictionary

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

LZW Coding (cont'd)

Example:

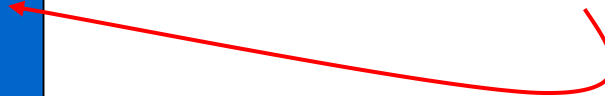39 39 126 126
39 39 126 126
39 39 126 126
39 39 126 126

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

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

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

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		

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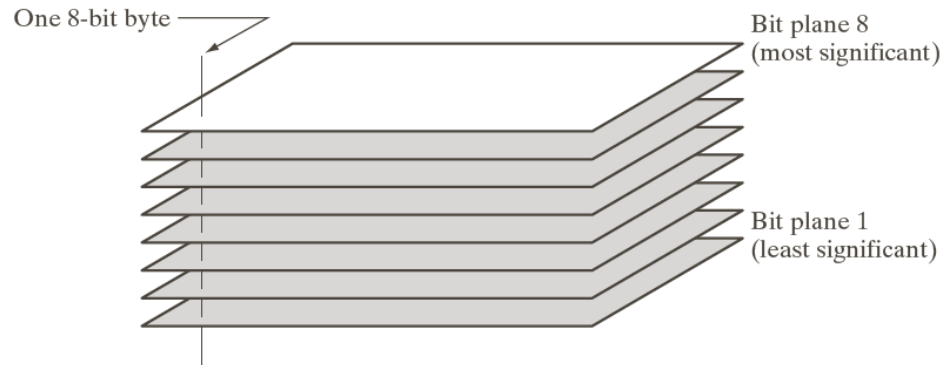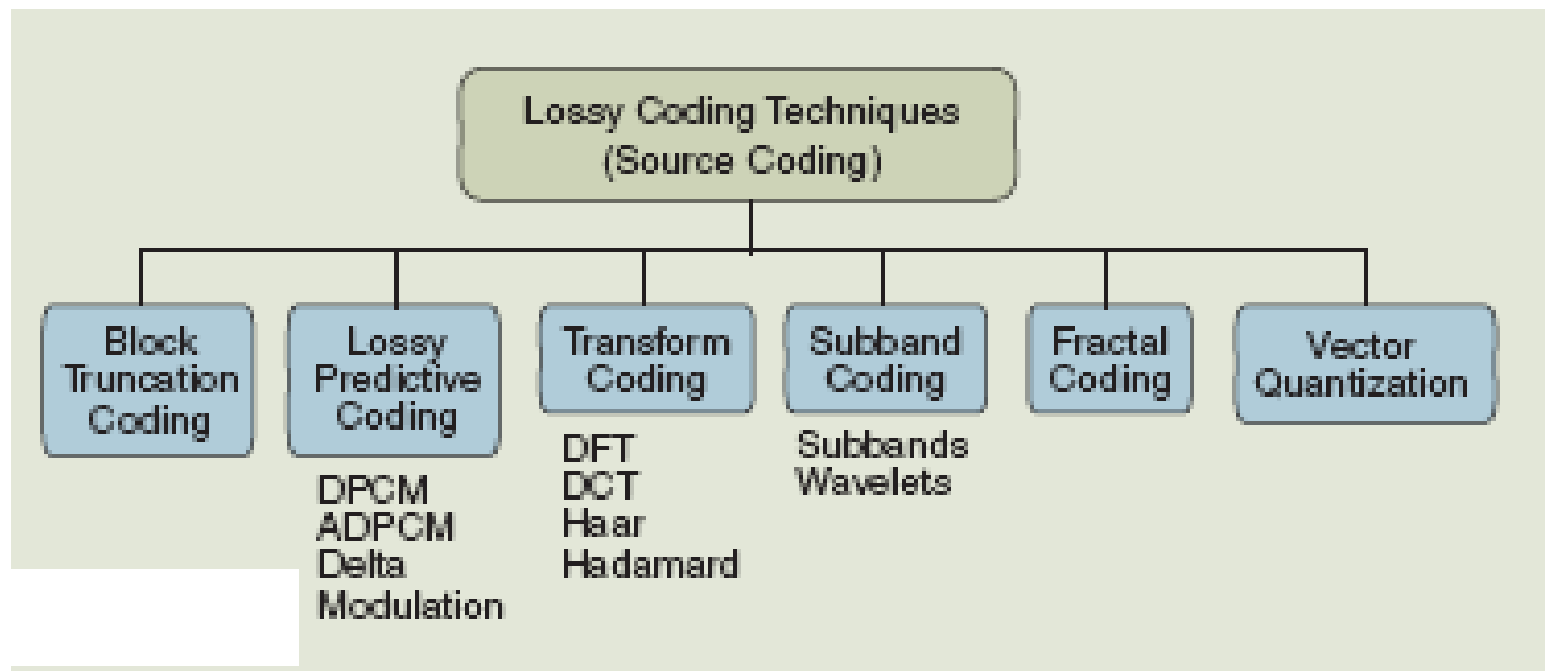
e.g., (0,1)(1,1)(0,1)(1,0)(0,2)(1,4)(0,2)

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.
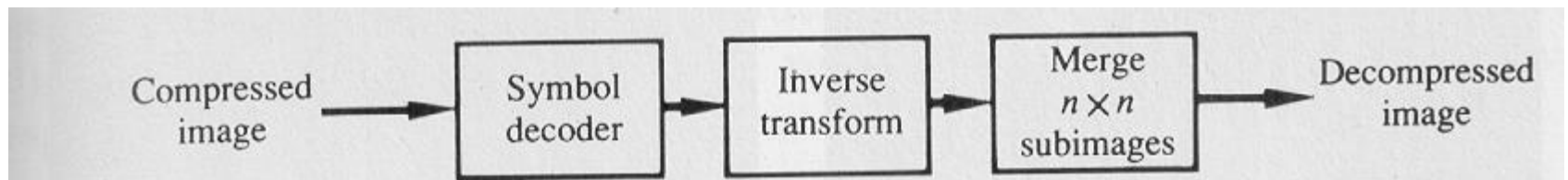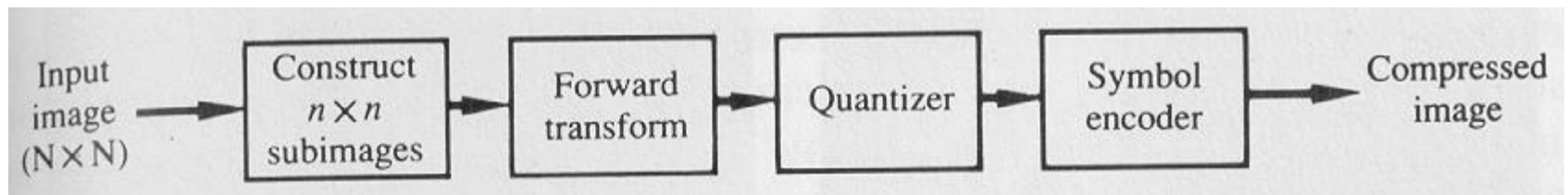
Example: Fourier Transform

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) e^{\frac{j2\pi(ux+vy)}{N}}, \quad x,y=0,1,...,N-1$$
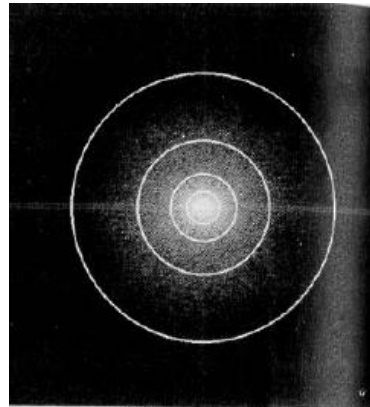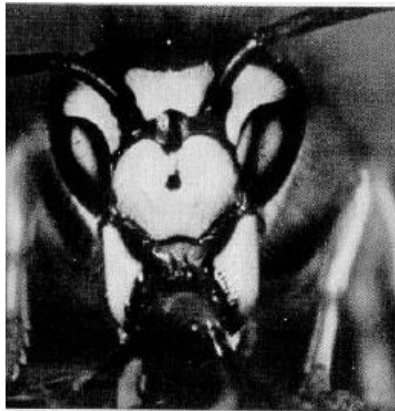
Note that the magnitude of the FT decreases, as u, v increase!

K << N

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

$\sum_{x,y} (\hat{f}(x, y) - f(x, y))^2$ 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 uses DCT for handling interpixel redundancy.

DCT (Discrete Cosine Transform)

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

$$u, v = 0, 1, ..., N\text{-}1$$

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

$$x, y = 0, 1, ..., N\text{-}1$$

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

DCT (cont'd)

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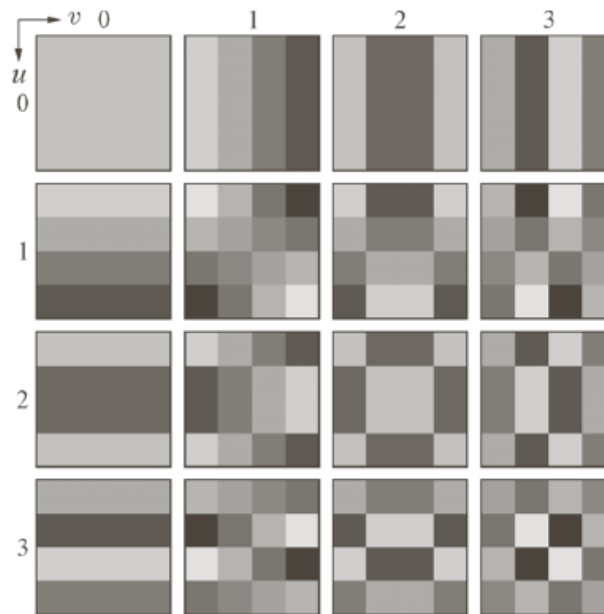
DCT (cont'd)

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

Reconstructed images
by truncating
50% of the
coefficients

DCT is a more compact
transformation!

DFT

WHT
(Walsh–Hadamard transform)

DCT
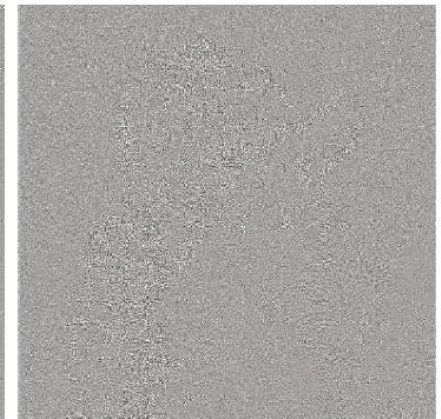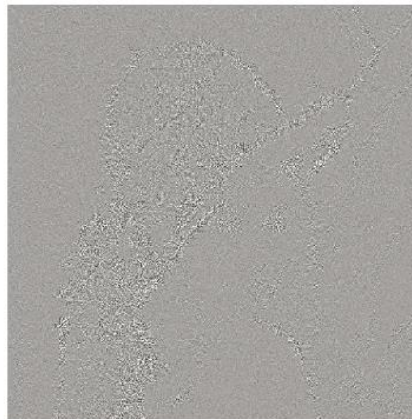
RMS error: 2.32   1.78   1.13

DCT (cont'd)

- Sub-image size selection:

Reconstructions
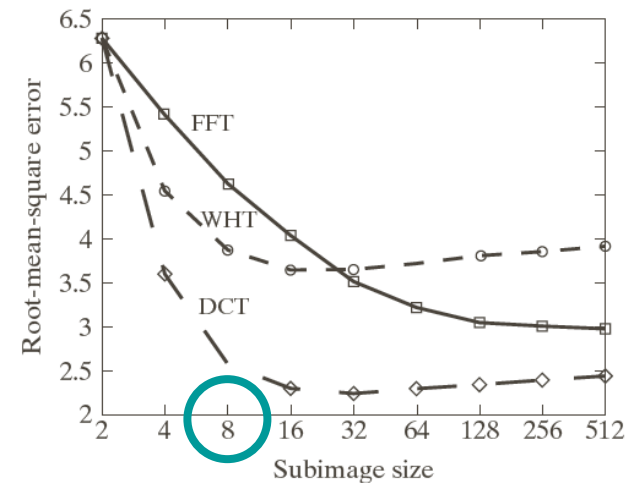
original   2 x 2 sub-images   4 x 4 sub-images   8 x 8 sub-images
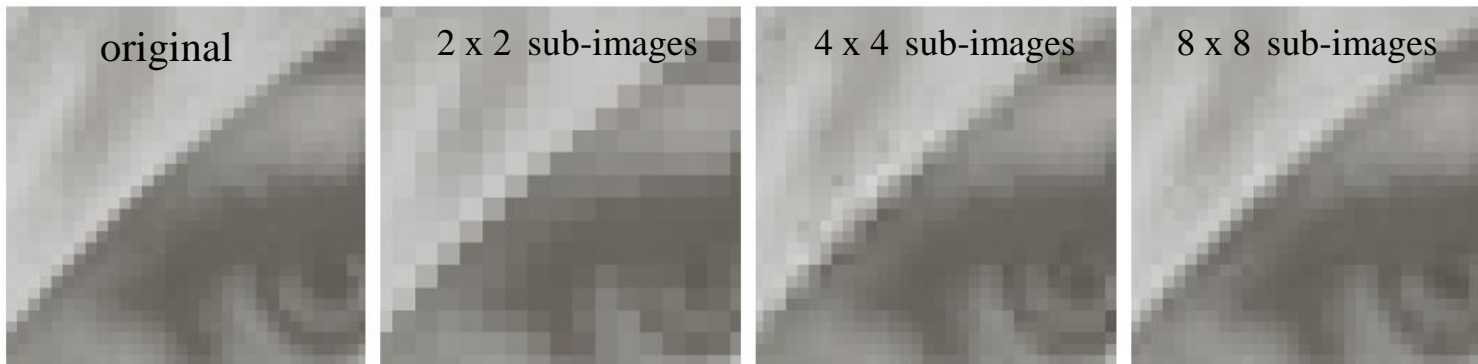
DCT (cont'd)

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

DFT

has n-point periodicity
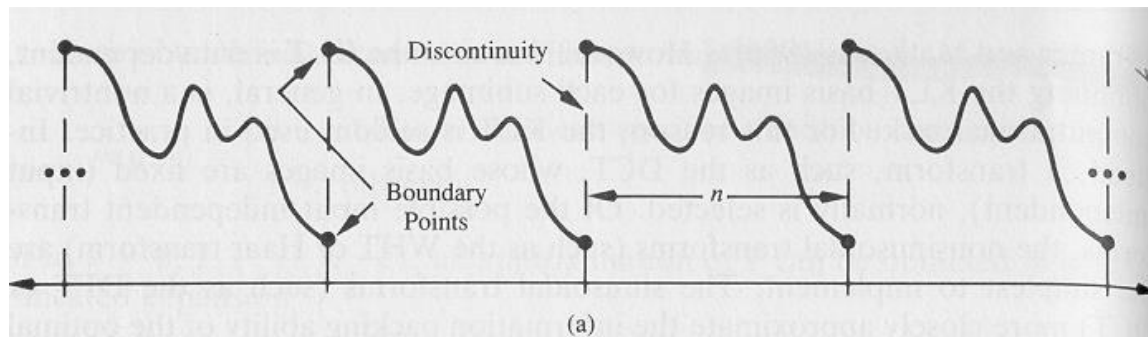
DCT

has 2n-point periodicity
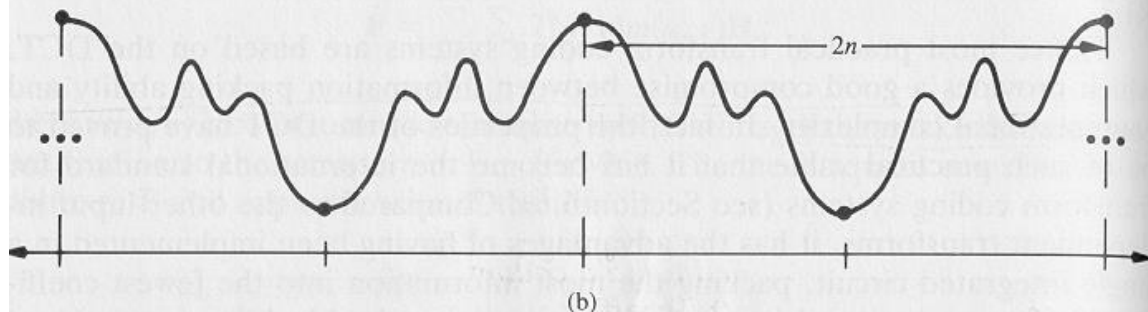
JPEG Compression

Accepted as an international image compression standard in 1992.
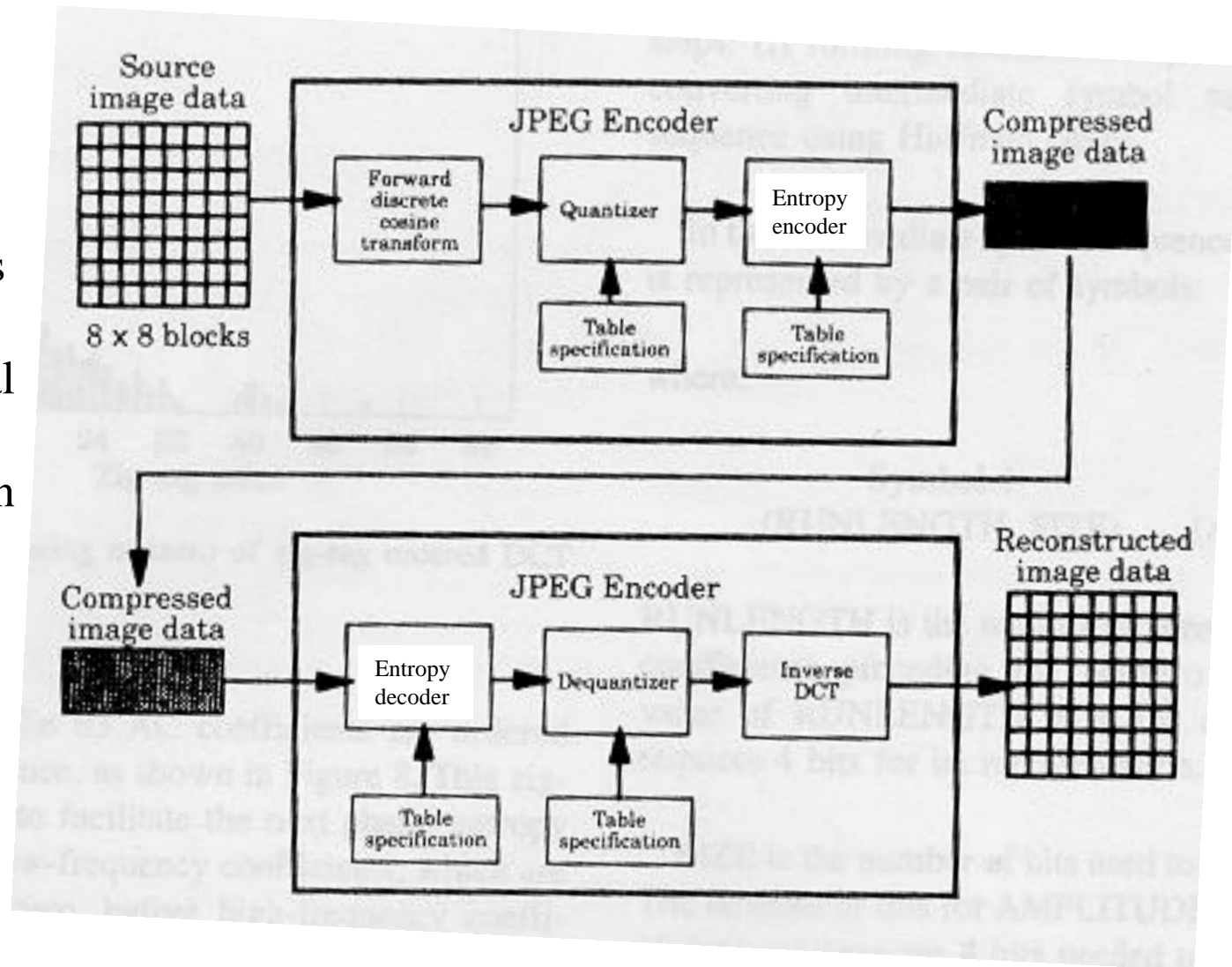
JPEG - Steps

1. Divide image into 8x8 subimages.

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	140	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

(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

[-128, 127]

(DCT - non-centered spectrum)

JPEG Steps

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

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

Q(u,v): quantization table

Example

(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 Table Q[i][j]

```
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

Horizontal frequency

DC AC_{01} AC_{07}

Vertical frequency

AC_{70} AC_{77}
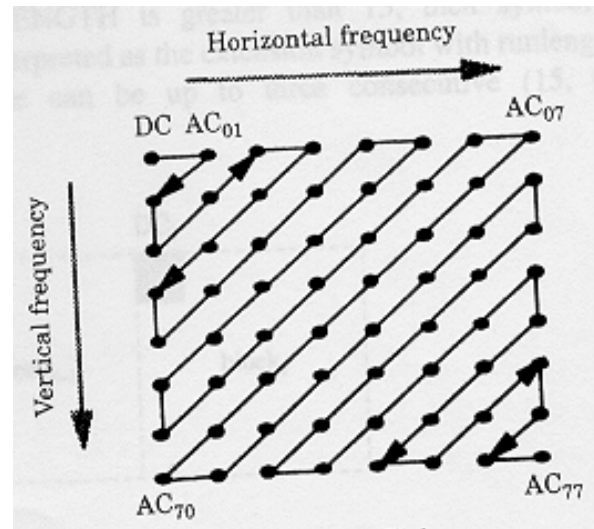
(f) Zig-zag sequence

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:

    6.1 Form "intermediate" symbol sequence.

    6.2 Encode "intermediate" symbol sequence into
       a binary sequence.

Intermediate Symbol Sequence – DC coeff

(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

(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),
(5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

	symbol_1 (SIZE)	symbol_2 (AMPLITUDE)
DC	(6)	(61)

SIZE: # bits need to encode the coefficient

DC Coefficient Encoding

symbol_1
(SIZE)

symbol_2
(AMPLITUDE)

predictive
coding:
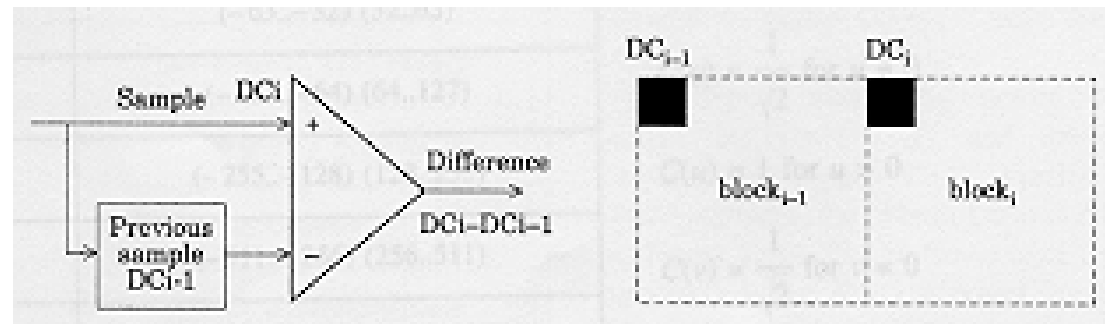
Intermediate Symbol Sequence – AC coeff

(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

(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),
(5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

end of block
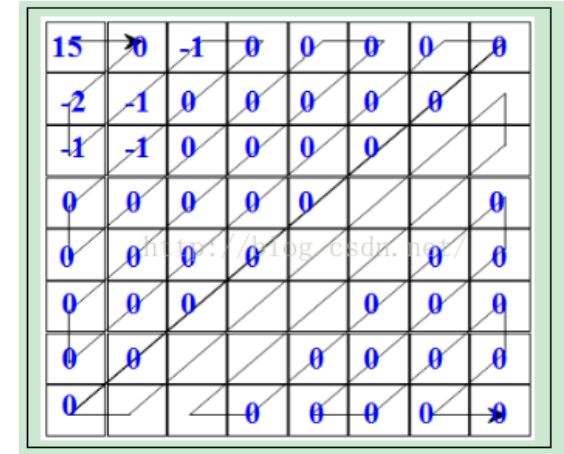
(1,-2),(0,-1),(0,-1),(0,-1),(2,-1), (0,0)   --EOB
(continuous zeros, next non-zero)

symbol_1 (RUN-LENGTH, SIZE)   symbol_2 (AMPLITUDE)

AC   (0, 2)   (-3)

RUN-LENGTH: run of zeros preceding coefficient
SIZE: # bits for encoding the amplitude of coefficient

Note: If RUN-LENGTH > 15, use symbol (15,0) ,

Example: JPEG compression

Y 分量 DC 系数 Huffman 表

Size	Code length	code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Y 分量 AC 系数 Huffman 表

Run/size	Length of code	code
0/0(EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
...	...	...
1/1	4	1100
1/2	5	11011
...	...	...
2/1	5	11100
3/1	6	111010
...	...	...
6/1	7	1111011
...	...	...
F/0(ZRL)	11	11111111001

15	0	−1	0	0	0	0	0
−2	−1	0	0	0	0	0	0
−1	−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	0	0	0	0	0	0	0

Assume previous DC=12，the coefficient after quantization is :

15-12, (1,-2), (0,-1), (0,-1), (0,-1), (2,-1), EOB

DC encoder: 3 ->(2,3) ->(011,11)

AC encoder:
(1,-2)->(1/2，-2) -T2-> 11011, (-2-1)mod2 -> 11011,01
(0,-1)->(0/1，-1) -T2->　00, (-1-1)mod2 -> 00,0
(0,-1)->(0/1，-1) -T2->　00, (-1-1)mod2 -> 00,0
(0,-1)->(0/1，-1) -T2->　00, (-1-1)mod2 -> 00,0
(2,-1)->(2/1，-1) -T2-> 11100, (-1-1)mod2 -> 11100,0
EOB->1010

After compression:
5+7+3+3+3+6+4 = 31 bits; (previous 8*8*8=512 bits)

What is the effect of the "Quality" parameter?

(58k bytes) (21k bytes) (8k bytes)

lower compression

higher compression

$$1 \leq quality \leq 25$$

What is the effect of the "Quality" parameter? (cont'd)

Table 6. Results of JPEG Compression for Grayscale Image 'Lisa' (320 ×240 pixels)

Quality factors	Original number of bits	Compressed number of bits	Compression ratio (Cr)	Bits/pixel (Nb)	RMS error
1	512,000	48,021	10·66	0·75	2·25
2	512,000	30,490	16·79	0·48	2·75
4	512,000	20,264	25·27	0·32	3·43
8	512,000	14,162	36·14	0·22	4·24
15	512,000	10,479	48·85	0·16	5.36
25	512,000	9,034	56·64	0·14	6·40

Effect of Quantization: homogeneous 8 x 8 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

$F(u, v)$

Fig. 9.2: JPEG compression for a smooth image block.

Effect of Quantization: homogeneous 8 x 8 block (cont'd)

Quantized

```
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
```

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

De-quantized

```
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
```

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

Effect of Quantization: homogeneous 8 x 8 block (cont'd)

Reconstructed

```
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
```

Original

```
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
```

Error is low!

```
 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
```

Effect of Quantization: non-homogeneous 8 x 8 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

$F(u,v)$

Effect of Quantization:
non-homogeneous 8 x 8 block (cont'd)

Quantized

De-quantized

```
 -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
```

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

Effect of Quantization:
non-homogeneous 8 x 8 block (cont'd)

Reconstructed

```
 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
```

Error is high!

```
  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
```

Original:

```
 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
```

Case Study: Fingerprint Compression

- FBI is digitizing fingerprints at 500 dots per inch with 8 bits of grayscale resolution.

- A single fingerprint card turns into about 10 MB of data!

A sample fingerprint image
768 x 768 pixels =589,824 bytes

WSQ Fingerprint Compression

- An image coding standard for digitized fingerprints employing the Discrete Wavelet Transform (*Wavelet/Scalar Quantization or WSQ).*

- Developed and maintained by:
  - FBI
  - Los Alamos National Lab (LANL)
  - National Institute for Standards and Technology (NIST)

Need to Preserve Fingerprint Details

The "white" spots in the middle of the black ridges are *sweat pores* and they are admissible points of identification in court.

<u>These details are just a couple pixels wide!</u>

What compression scheme should be used?

- Lossless or lossy compression?

- In practice lossless compression methods haven't done better than 2:1 on fingerprints!

- Does JPEG work well for fingerprint compression?

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
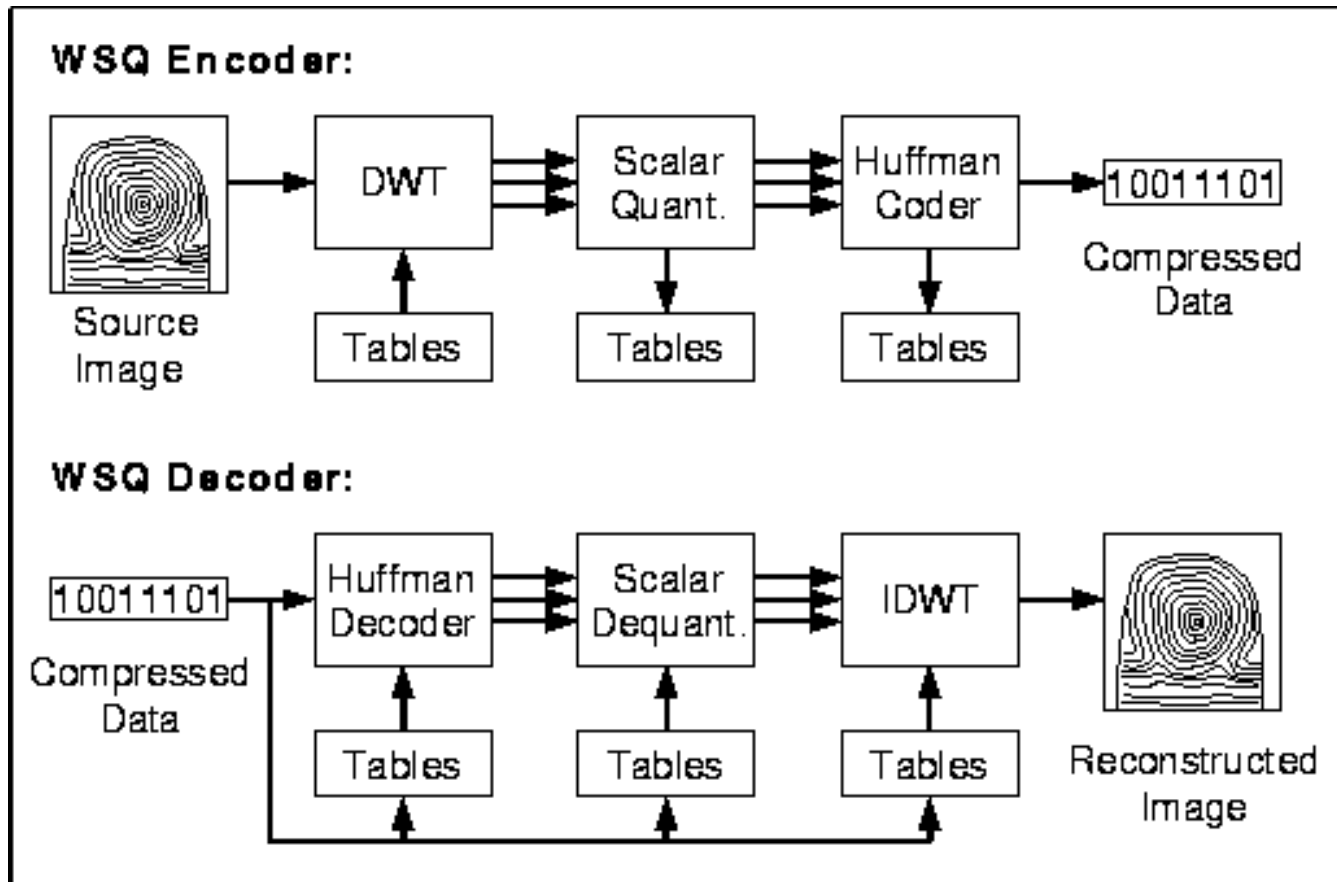
file size 45621 bytes
compression ratio: 12.9

Fine details are better preserved.

No "blocky" artifacts.

WSQ Algorithm

Compression ratio

- FBI's target bit rate is around 0.75 bits per pixel (bpp)

- This corresponds to a compression ratio of 8/0.75=10.7

- Target bit rate can set via a parameter, similar to the "quality" parameter in JPEG.

Varying compression ratio (cont'd)

Original image 768 x 768 pixels (589824 bytes)

Varying compression ratio (cont'd)
0.9 bpp compression

WSQ image, file size 47619 bytes, compression ratio 12.4

JPEG image, file size 49658 bytes, compression ratio 11.9

Varying compression ratio (cont'd)
0.75 bpp compression

WSQ image, file size 39270 bytes
compression ratio 15.0

JPEG image, file size 40780 bytes,
compression ratio 14.5

Varying compression ratio (cont'd)
0.6 bpp compression

WSQ image, file size 30987 bytes, compression ratio 19.0

JPEG image, file size 30081 bytes, compression ratio 19.6

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

Progressive JPEG (cont'd)

- Each scan encodes a subset of DCT coefficients.

- We'll examine the following algorithms:

  (1) Progressive spectral selection algorithm

  (2) Progressive successive approximation algorithm

  (3) Combined progressive algorithm

Progressive JPEG (cont'd)

(1) Progressive spectral selection algorithm

- Group DCT coefficients into several spectral bands
- Send low-frequency DCT coefficients first
- Send higher-frequency DCT coefficients next

Band 1: DC coefficient only
Band 2: AC_1 and AC_2 coefficients
Band 3: AC_3, AC_4, AC_5, AC_6, coefficients
Band 4: AC_7...AC_{63}, coefficients

Example

Table 8. Progressive spectral selection JPEG. (Image 'Cheetah': 320 × 240 pixels –> 512,000 bits)

Scan number	Bits transmitted	Compression ratio	Bits/pixel	RMS error
1	29,005	17·65	0·45	19·97
2	37,237	7·73	1·04	13·67
3	71,259	3·72	2·15	7·90
4	32,489	3·01	2·66	4·59
Sequential JPEG	172,117	2·97	2·69	4·59

Progressive JPEG (cont'd)

(2) Progressive successive approximation algorithm

- Send all DCT coefficients but with lower precision.
- Refine DCT coefficients in later scans.

Band 1: All DCT coefficients (divided by four)
Band 2: All DCT coefficients (divided by two)
Band 3: All DCT coefficients (full resolution)

Example

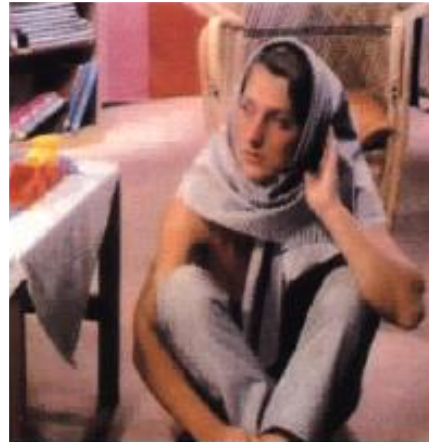**Table 9.** Progressive successive approximation JPEG. (Image 'Cheetah': 320 × 240 pixels –> 512,000 bits)

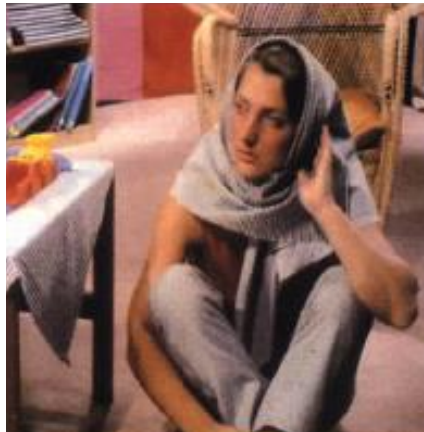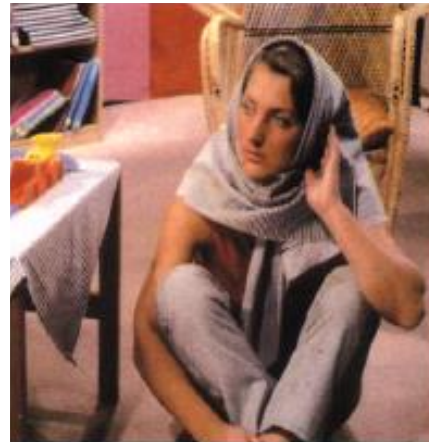| Scan number	Bits transmitted	Compression ratio	Bits/pixel	RMS error
1	26,215	19·53	0·41	22·48
2	34,506	8·43	0·95	12·75
3	63,792	4·11	1·95	7·56
4	95,267	2·33	2·43	4·59
Sequential JPEG	172,117	2·97	2·69	4·59

Example

after 0.9s

after 1.6s
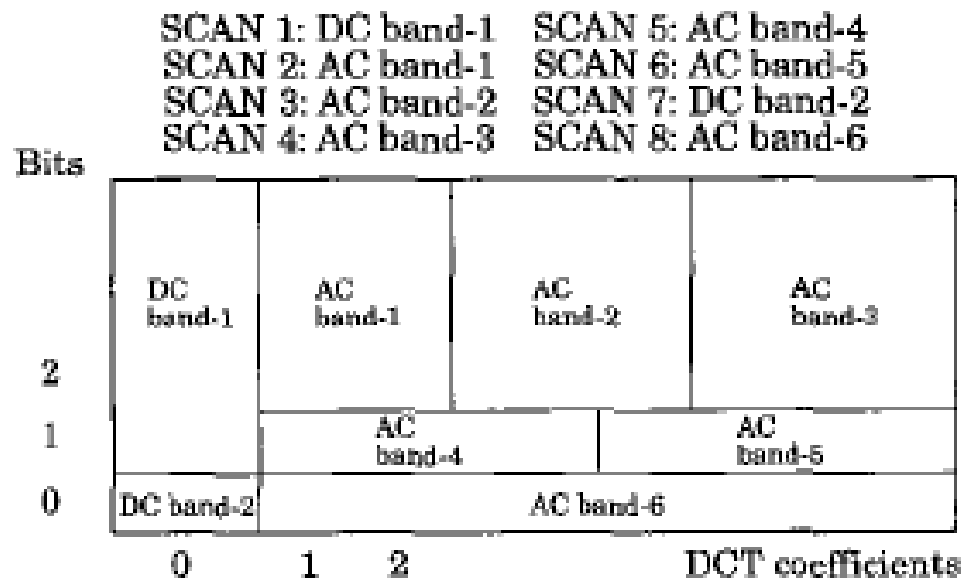

after 3.6s

after 7.0s

Progressive JPEG (cont'd)

(3) Combined progressive algorithm

– Combines spectral selection and successive approximation.

Hierarchical JPEG

- Hierarchical mode encodes the image at different resolutions.

- Image is transmitted in multiple passes with increased resolution at each pass.
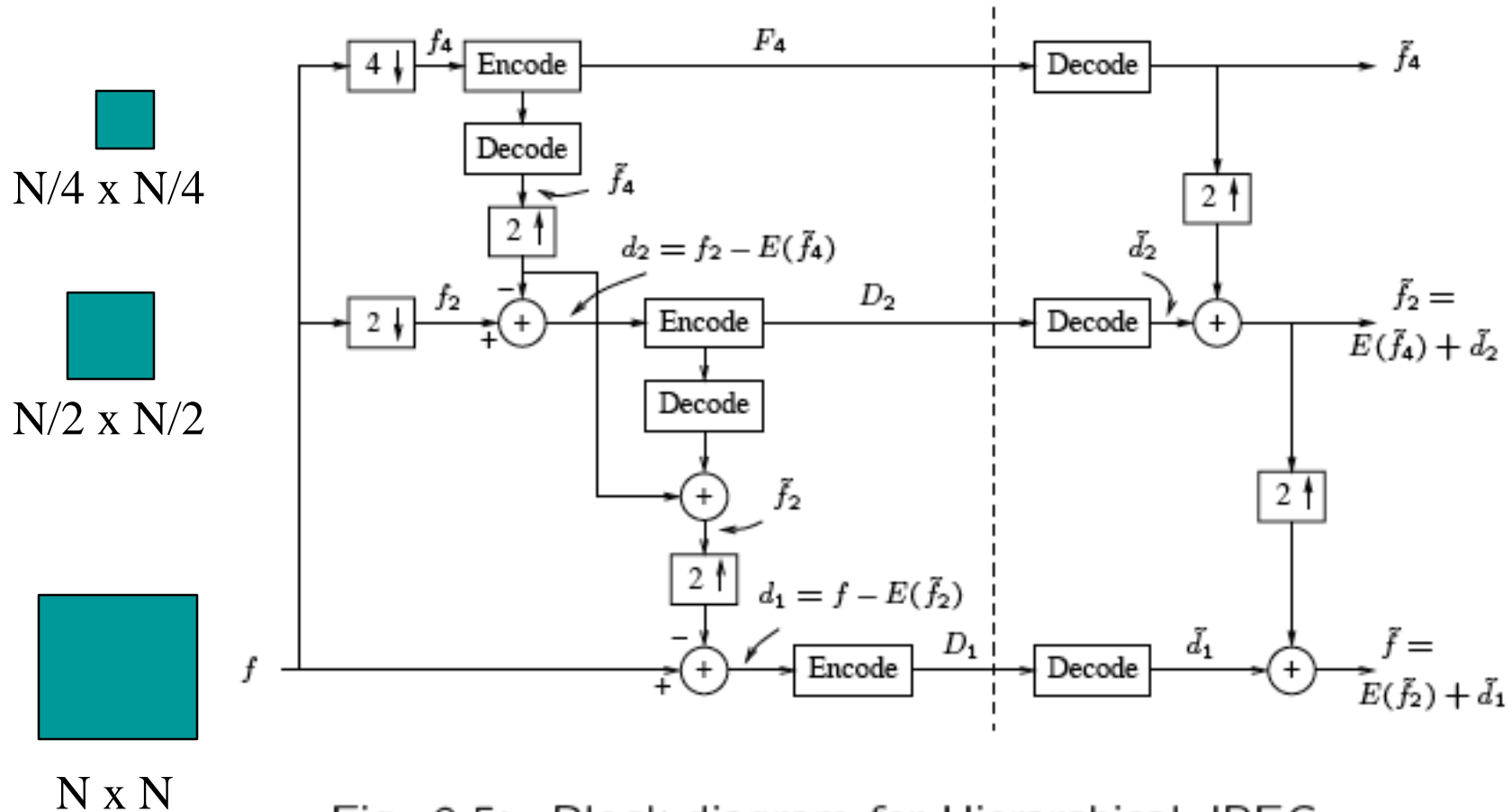
Hierarchical JPEG (cont'd)

N/4 x N/4

N/2 x N/2

N x N

Fig. 9.5: Block diagram for Hierarchical JPEG.

More Methods …

- See "Image Compression Techniques", IEEE Potentials, February/March 2001

Deep Learning Image Compression

- Learned-transform
  - Auto-encoders learn latent-space representation of images
- Differentiable approximation to quantization
  - Soft quantization
- Generative Codecs
  - O. Rippel and L. Bourdev, "Real-time adaptive image compression," ICML 2017, arXiv, 16 May 2017.
  - S. Santurkar, D. Budden, and N. Shavit, "Generative compression," arXiv, June 2017.

Soft Quantization

- Hard quantization for d-bits:
  - $$q = Q(z) = \left\lfloor z \times 2^d \right\rfloor$$

- However this function yields zero gradients except at decision boundaries. Therefore soft quantization is employed in training phase.

- Soft quantization for d-bits:
  - $$\tilde{z} = \sum_{i=0}^{2^d-1} \frac{\exp(-\|z \times 2^d - i\|)}{\sum_{j=0}^{2^d-1} \exp(-\|z \times 2^d - j\|)} \times i$$

E. Agustsson, et al. Soft-to-hard vector quantization for end-to-end learning compressible representations. NIPS, 2017

Enhancing Standard Codecs

- Deep networks learn free parameters of state of the art standards-based encoders
  - Block partitioning
  - Mode selection
  - Quantization parameter selection
  - In-loop filter
- Pre-processing and/or post-processing
  - Learned smoothing for pre-processing
  - Artifact removal

CVPR-CLIC 2018
Challenge on Learned Image Compression

- Rules
  - Compression rate of the whole test set must not exceed 0.15 bpp (average).
  - Participants are ranked according to
    - PSNR
    - Scores provided by human raters (MOS)
- Dataset
  - New: 1633 training, 102 validation, 286 test images.
    - DatasetP (professional)
    - DatasetM (mobile)

CLIC 2018 Winners

- Best MOS (also best MS-SSIM)
  - An Autoencoder-based Learned Image Compressor: Description of Challenge Proposal by NCTU
- Best PSNR
  - CNN-Optimized Image Compression with Uncertainty based Resource Allocation
- Fastest:
  - xvc codec

CLIC 2018 Results

- Only submissions which are evaluated for *MOS* scores are shown.

Entry	PSNR	MOS	MS-SSIM	Images Size (Bytes)	Decoding Time (ms)	Decoder Size (Bytes)
TucodecTNGcnn4p	27.67	3.587823941	0.964	13323808	15412641	97043988
xvc	27.09	3.580864198	0.954	13328076	535566	853416
iipTiramisu	30.89	3.56129165	0.955	13320299	111905135	9034988
TucodecTNG	30.76	3.552263374	0.955	13325574	46535029	2596395
yfan	30.43	3.531069959	0.951	13324840	74712678	4779153
tangzhimin	30.24	3.490740741	0.951	13308588	30311917	4984420
Amnesiack	30.30	3.450020568	0.950	13327732	179313414	7070429
AmnesiackLite	28.77	3.440510498	0.956	13325728	29297186	1220246
MVGL	30.14	3.3571723	0.948	13328219	219961195	25873249
BPG	29.64	3.175066859	0.943	13189634	333725	377858
JPEG	25.66	1.170987654	0.863	13326518	136074	166

Fastest

- xvc – A conventional codec
  - proprietary
  - Block based
  - Traditional approach for prediction, residual representation
- Originally developed for video compression.
- No machine learning is involved.

J.Samuelsson, P. Hermansson, Image compression with xvc, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

Best MOS and MS-SSIM
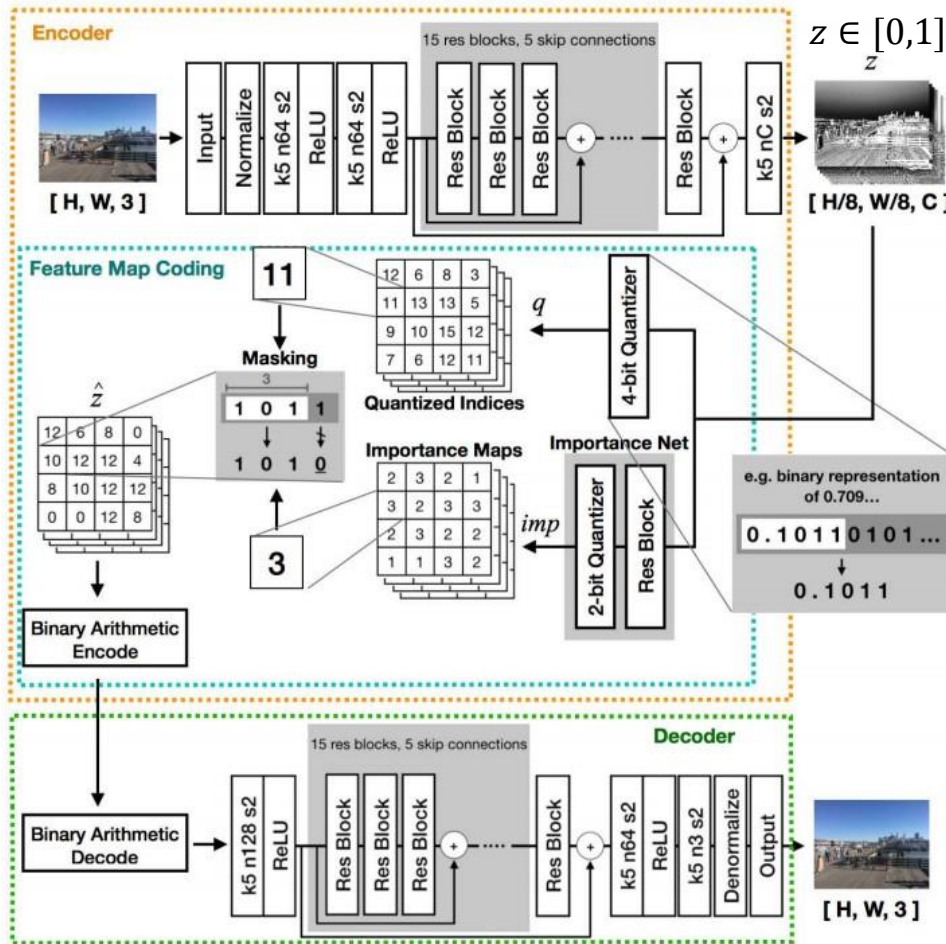
- Based on autoencoder
- 4-bit quantization
- Soft quantization
- Importance Network

D. Alexandre, et al., An autoencoder-based learned image compressor: Description of challenge proposal by NCTU, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

Importance Net

- Importance Net learns the important parts of the representation so that the system allocates more bits to complicated areas.

- It is made of residual blocks and another quantizer to select the number of bits

Optimization - Loss Function

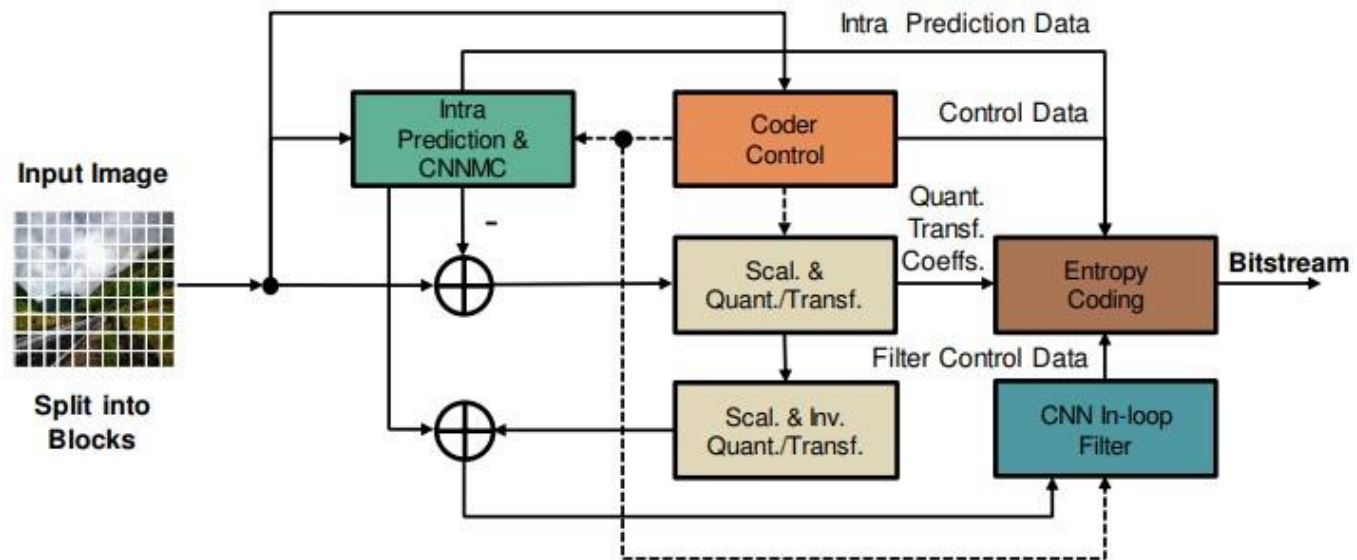- Loss function is a weighted sum of rate and distortion

  ◦ $L = \lambda \times H(imp) + L_d$

  where

  ◦ $L_d = \frac{MSE}{2\sigma_1^2} + \frac{MSSSIM}{2\sigma_2^2} + \log(\sigma_1^2) + \log(\sigma_2^2)$

- Rate loss H(imp) is estimated by summing up all values of the importance maps imp

- σ_1 and σ_2 are learnable parameters

Best PSNR

- Based on the JEM platform (the HEVC codec)
- Contributions:
  - CNN based in-loop filter (CNNIF) and
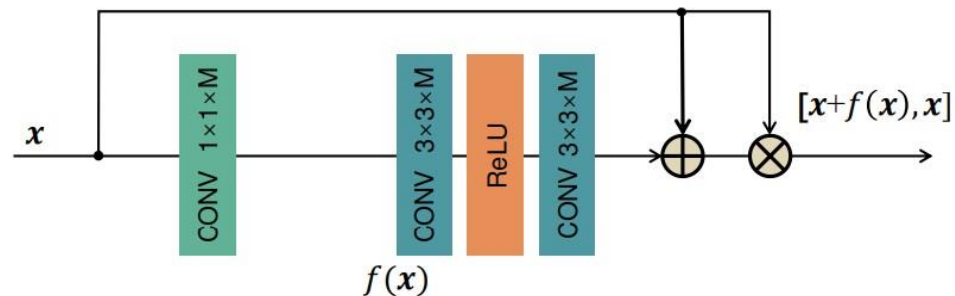 - CNN based mode coding (CNNMC)

Z. Chen, et al., CNN-optimized image compression with uncertainty based resource allocation, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

CNN based In-Loop Filter (CNNIF)

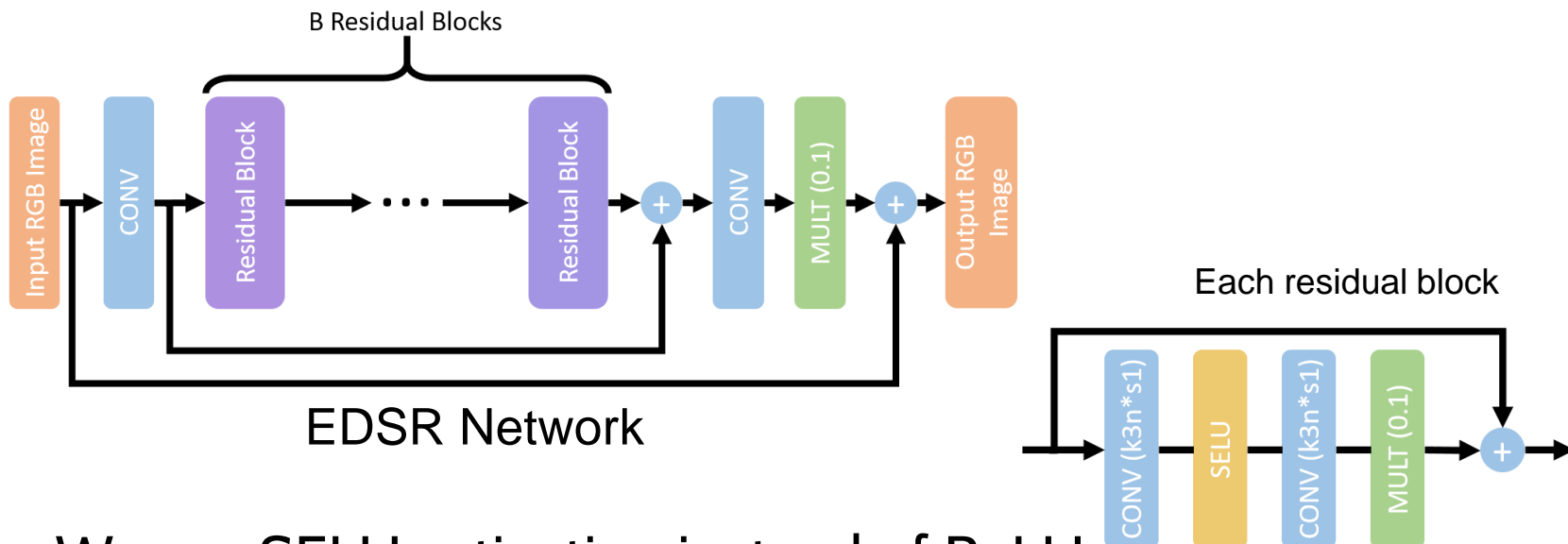- In-loop filter consists of stacked Dense Residual Units (DRU)
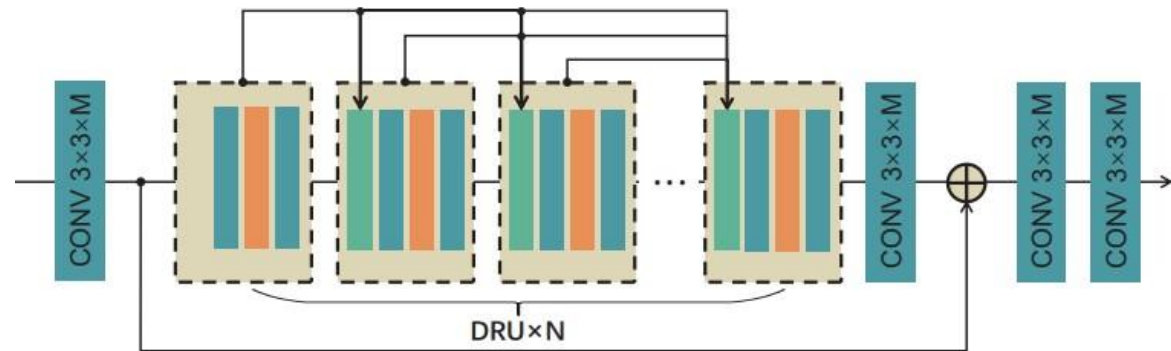
A Dense Residual Unit (DRU)

Learned Artifact Suppression

- Compression artifacts have structure that can be learned

EDSR Network

Each residual block

- We use SELU activation instead of ReLU

- It is trained to remove artifacts introduced by BPG codec.

- Although we trained our network with a single QP (40), it can improve images encoded by QP between 39 and 43.

O. Kirmemis, G. Bakar and A.M. Tekalp, Learned Compression artifact removal by deep residual networks, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2018

Residual GRU (RGRU)

- In-loop filter consists of stacked Dense Residual Units (DRU)

Encoder				
Input (floats) Size H×W×3	E-Conv Conv 3×3×64 Stride: 2×2	E-RNN#1 RNN Conv 3×3×256 Stride: 2×2	E-RNN#2 RNN Conv 3×3×512 Stride: 2×2	E-RNN#3 RNN Conv 3×3×512 Stride: 2×2

Encoded (floats) (H/16)×(W/16)×512

Binarizer
B-Conv Conv 1×1×32 Stride: 1×1

Binary Code (bits), Size (H/16)×(W/16)×32

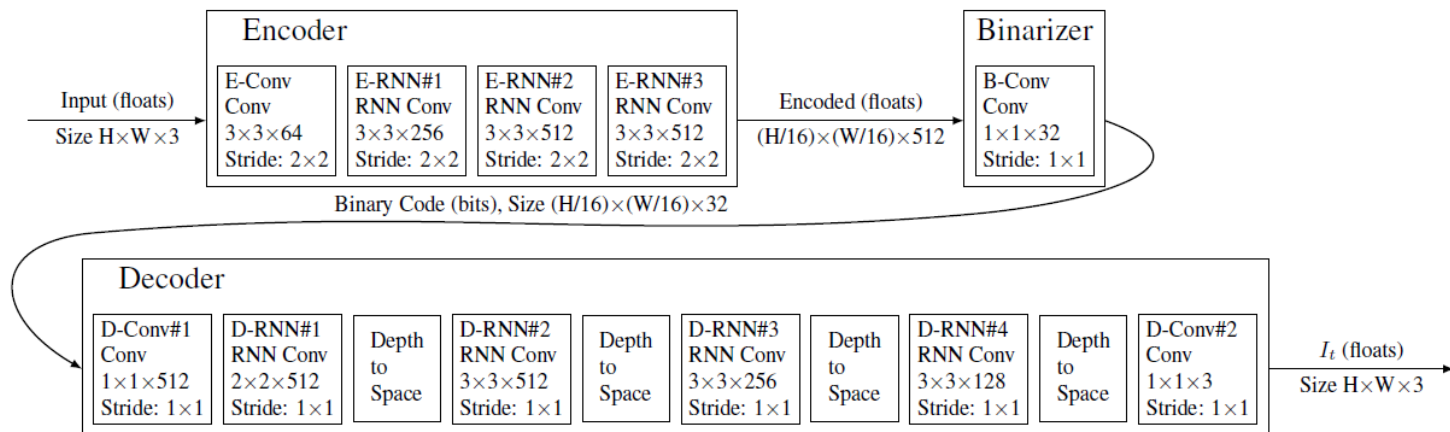| Decoder								
D-Conv#1 Conv 1×1×512 Stride: 1×1	D-RNN#1 RNN Conv 2×2×512 Stride: 1×1	Depth to Space	D-RNN#2 RNN Conv 3×3×512 Stride: 1×1	Depth to Space	D-RNN#3 RNN Conv 3×3×256 Stride: 1×1	Depth to Space	D-RNN#4 RNN Conv 3×3×128 Stride: 1×1	Depth to Space

D-Conv#2 Conv 1×1×3 Stride: 1×1

I_t (floats) Size H×W×3

Figure 1. A single iteration of our shared RNN architecture.

https://ai.googleblog.com/2016/09/image-compression-with-neural-networks.html
https://www.youtube.com/watch?v=RRPuXMyKIWw