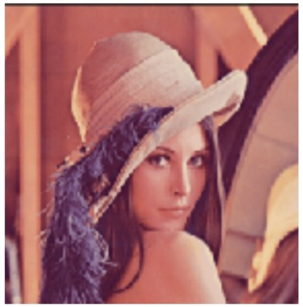
Compression II

---CS355: Digital Forensics---

Dr. Yu Guan,
Department of Computer Science
University of Warwick
Yu.Guan@warwick.ac.uk

JPEG compression steps

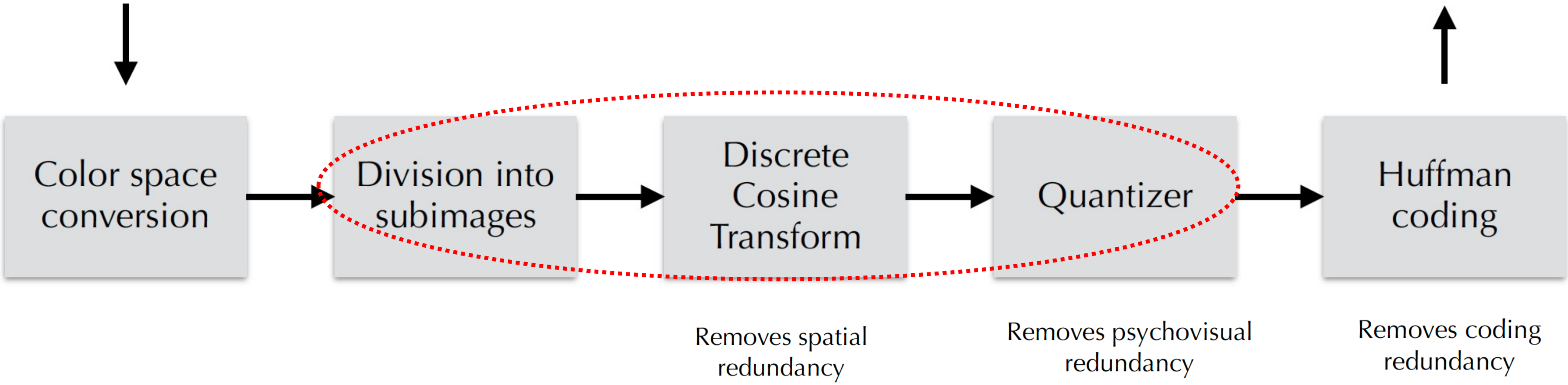


Original

Redundancy

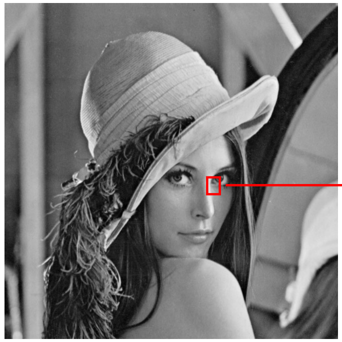
- Spatial redundancy
- Psychovisual redundancy
- Coding redundancy

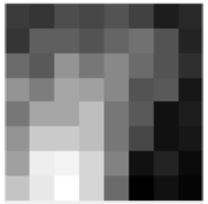
Compressed



Division into subimages

8×8 block

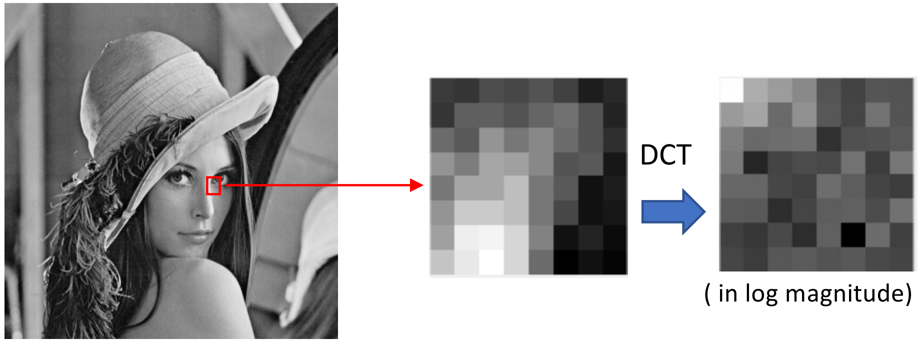




161	160	164	163	165	162	156	158
160	167	167	165	168	170	165	157
169	166	176	171	174	169	165	159
176	172	179	178	174	165	166	155
171	179	179	183	171	166	154	156
176	185	185	183	171	163	154	155
178	191	192	187	173	154	157	153
184	190	194	187	169	151	154	152

Spatial redundancy: neighboring pixels have similar values

DCT



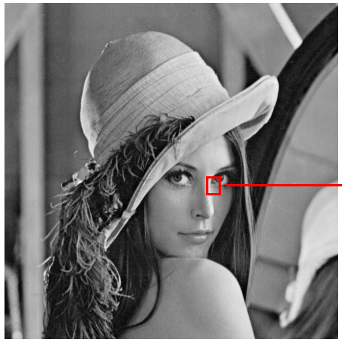
DCT coefficients: $F(u,v)$

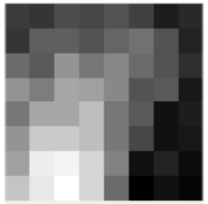
1352	58	−30	−14	−2	1	−2	2
−28	−43	4	19	−2	−1	6	2
−10	4	5	−6	0	−2	−2	2
−8	0	−1	−1	1	−6	1	−7
−4	4	1	−1	5	3	1	−1
2	2	0	−1	3	3	−1	6
−1	1	1	0	3	0	5	1
−1	−1	−2	−2	2	−2	1	−1

Removes spatial redundancy

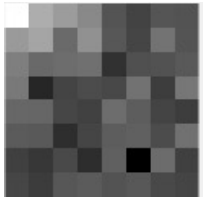
Only a few DCT basis can represent most of the information!

Quantization (Quality level 90)





DCT



(in log magnitude)

Quantization Q_{90}

(in log magnitude)

Removes psychovisual redundancy

Quantization will yield many 0s (i.e., mid/high frequency coefficients)

$F(u,v)$

$$\begin{matrix} 1352 & 58 & -30 & -14 & -2 & 1 & -2 & 2 \\ -28 & -43 & 4 & 19 & -2 & -1 & 6 & 2 \\ -10 & 4 & 5 & -6 & 0 & -2 & -2 & 2 \\ -8 & 0 & -1 & -1 & 1 & -6 & 1 & -7 \\ -4 & 4 & 1 & -1 & 5 & 3 & 1 & -1 \\ 2 & 2 & 0 & -1 & 3 & 3 & -1 & 6 \\ -1 & 1 & 1 & 0 & 3 & 0 & 5 & 1 \\ -1 & -1 & -2 & -2 & 2 & -2 & 1 & -1 \end{matrix}$$

$Q(u,v)$

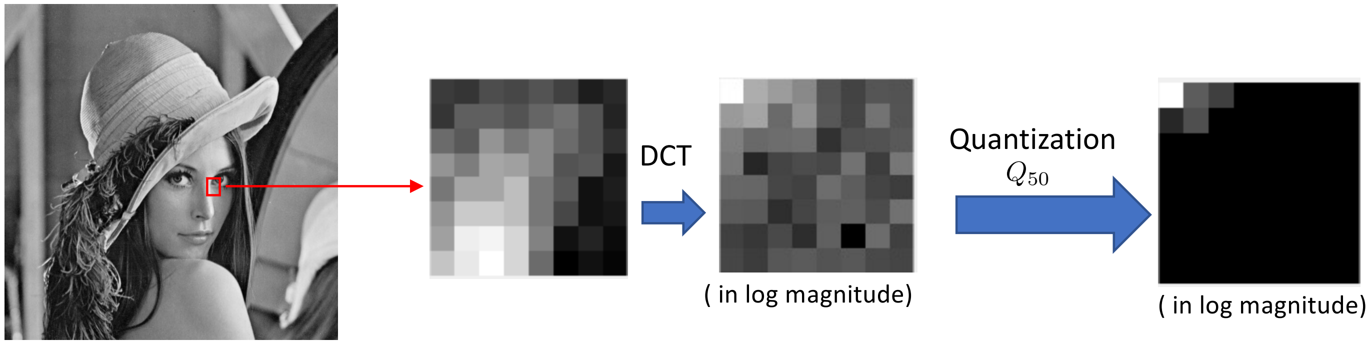
Quality level 90 (**high** quality)

$$Q_{90} = \begin{matrix} 3 & 2 & 2 & 3 & 5 & 8 & 10 & 12 \\ 2 & 2 & 3 & 4 & 5 & 12 & 12 & 11 \\ 3 & 3 & 3 & 5 & 8 & 11 & 14 & 11 \\ 3 & 3 & 4 & 6 & 10 & 17 & 16 & 12 \\ 4 & 4 & 7 & 11 & 14 & 22 & 21 & 15 \\ 5 & 7 & 11 & 13 & 16 & 12 & 23 & 18 \\ 10 & 13 & 16 & 17 & 21 & 24 & 24 & 21 \\ 14 & 18 & 19 & 20 & 22 & 20 & 20 & 20 \end{matrix}$$

$$\text{round}\left(\frac{F(u,v)}{Q(u,v)}\right) \longrightarrow \begin{matrix} 451 & 29 & -15 & -5 & 0 & 0 & 0 & 0 \\ -14 & -21 & 1 & 5 & 0 & 0 & 1 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

47 “0”s out of 64

Quantization (Quality level 50)



Removes psychovisual redundancy

Quantization will yield many 0s (i.e., mid/high frequency coefficients)

$$F(u,v) \quad \begin{matrix} 1352 & 58 & -30 & -14 & -2 & 1 & -2 & 2 \\ -28 & -43 & 4 & 19 & -2 & -1 & 6 & 2 \\ -10 & 4 & 5 & -6 & 0 & -2 & -2 & 2 \\ -8 & 0 & -1 & -1 & 1 & -6 & 1 & -7 \\ -4 & 4 & 1 & -1 & 5 & 3 & 1 & -1 \\ 2 & 2 & 0 & -1 & 3 & 3 & -1 & 6 \\ -1 & 1 & 1 & 0 & 3 & 0 & 5 & 1 \\ -1 & -1 & -2 & -2 & 2 & -2 & 1 & -1 \end{matrix}$$

$$Q(u,v)$$

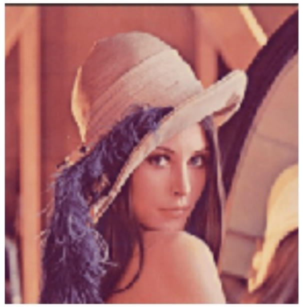
Quality level 50 (**medium** quality)

$$Q_{50} = \begin{matrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{matrix}$$

$$\text{round}\left(\frac{F(u,v)}{Q(u,v)}\right) \Rightarrow \begin{matrix} 85 & 5 & -3 & -1 & 0 & 0 & 0 & 0 \\ -2 & -4 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

56 “0”s out of 64

JPEG compression steps



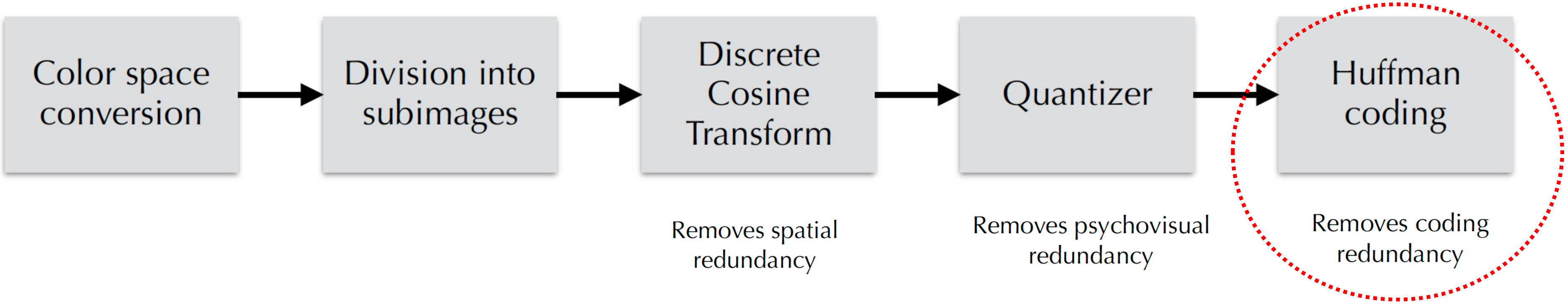
Original

Redundancy

- Spatial redundancy
- Psychovisual redundancy
- Coding redundancy

Quantized DCT coefficients

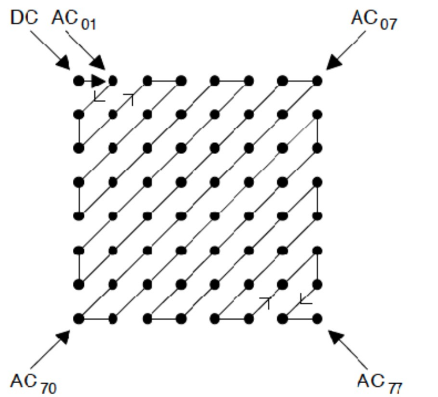
85	5	−3	−1	0	0	0	0
−2	−4	0	1	0	0	0	0
−1	0	0	0	0	0	0	0
−1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Quantized DCT coefficients

Quantized DCT coefficients

85	5	-3	-1	0	0	0	0
-2	-4	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	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Series of DCT coefficients
to be coded and stored
in this order:

$\mathrm{DC}, \mathrm{AC}_{01}, \mathrm{AC}_{10}, \mathrm{AC}_{20}, \mathrm{AC}_{11}, \mathrm{AC}_{02}, \mathrm{AC}_{03}, \mathrm{AC}_{12}, \mathrm{AC}_{21}, \mathrm{AC}_{30}, \ldots\ldots\ldots, \mathrm{AC}_{77}$

$F_Q(u=0, v=2)$

$$85, 5, -2, -1, -4, -3, -1, 0, 0, -1, 0, \ldots, \ldots 0, 0, 0, 0, 0, \ldots, 0, 0, 0, 0, 0, 0$$

with 56/64=87.5% of the entries 0s, store them direction?

Image storage (uncompressed example)

TIFF (**uncompressed**)
~786KB

lena512color.tiff
Available in Moodle (lab 2)

Resolution: 512×512 pixels

8-bit, RGB

Total entry number (pixels here): $512 \times 512 \times 3 \approx 786K\text{Bytes}$

(1 Byte = 8bits)

Or

$786K \times 8\text{bits}$

Each grayscale pixel is 8-bit here, with the values ranging from [0, 255], i.e.,

$$00000000$$
$$\cdots$$
$$11111111$$

Toy example

$$\mathrm{I} = \begin{array}{|ccccc|} \hline 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ \hline \end{array}$$

If we use 8-bit to code all the entries with the values range from [0, 255]:

$$\begin{array}{c} 00000000 \\ \cdots \\ 11111111 \end{array}$$

The size will be:

Total entry number: $4 \times 5 \times 8\text{bits}$

Toy example

$$\mathbf{I} = \begin{array}{|ccccc|} \hline 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ \hline \end{array}$$

Since there are only 4 different values (symbols), we may use 2-bit instead

Codewords: **00** (for 21), **01** (for 95), **10** (for 169) and **11** (for 243)

The codewords have to be communicated to the decoder.

The size will be: Total entry number: $4 \times 5 \times 2\text{bits}$

Fixed length code

$$
I = \begin{bmatrix} 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \end{bmatrix}
$$

8 bits/value

$$
\begin{vmatrix} 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \end{vmatrix}
$$

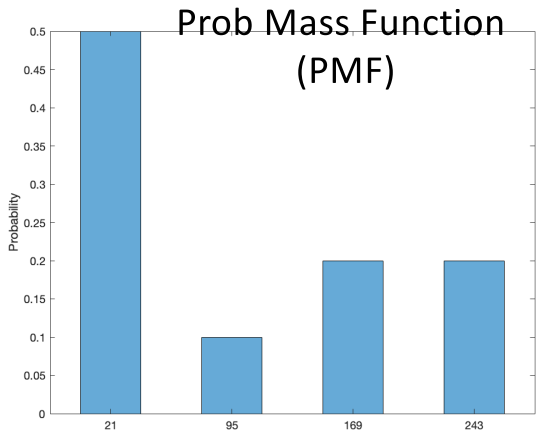
2 bits/value

$$
\begin{vmatrix} 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \end{vmatrix}
$$

Codewords:
00 (for 21), **01** (for 95),
10 (for 169) and **11** (for 243)

Can we do better?

Huffman coding



$$
I = \begin{bmatrix} 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \end{bmatrix}
$$

pixel value	frequency of occurrence	probability
21	10	0.5
95	2	0.1
169	4	0.2
243	4	0.2

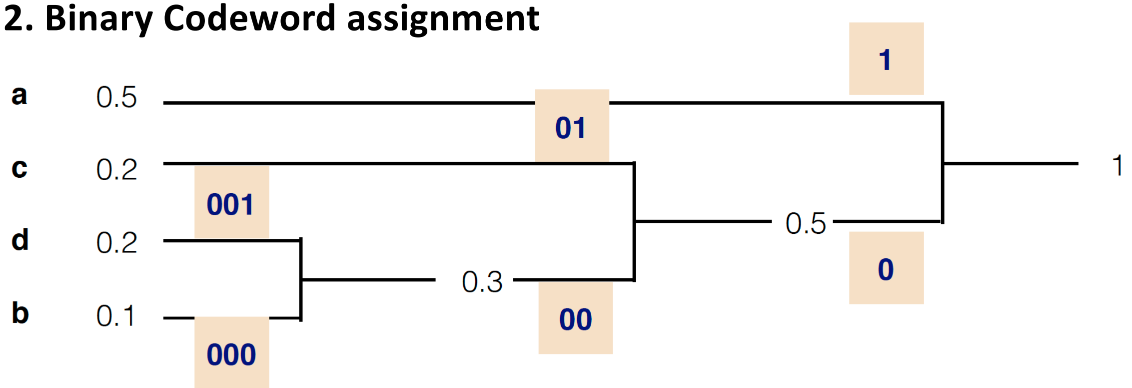
Basic idea:
Assign short (binary) codeword to numbers (symbols) with higher probability, and longer codeword to the ones with lower probability.

Huffman coding

1. Ranking probabilities

21	**a**	0.5		**a**	0.5
95	**b**	0.1	reorder →	**c**	0.2
169	**c**	0.2		**d**	0.2
243	**d**	0.2		**b**	0.1

2. Binary Codeword assignment

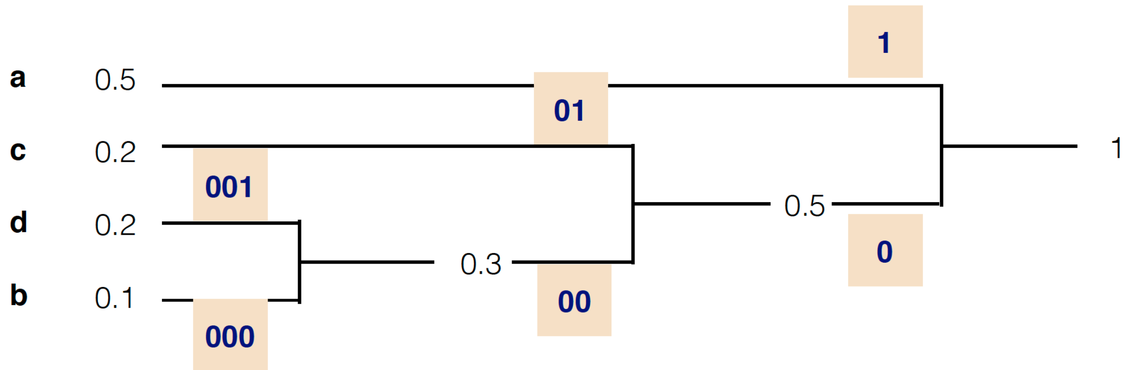


pixel value	frequency of occurrence	probability
21	10	0.5
95	2	0.1
169	4	0.2
243	4	0.2

Basic idea:
Assign short (binary) codeword to numbers (symbols) with higher probability, and longer codeword to the ones with lower probability.

Huffman coding

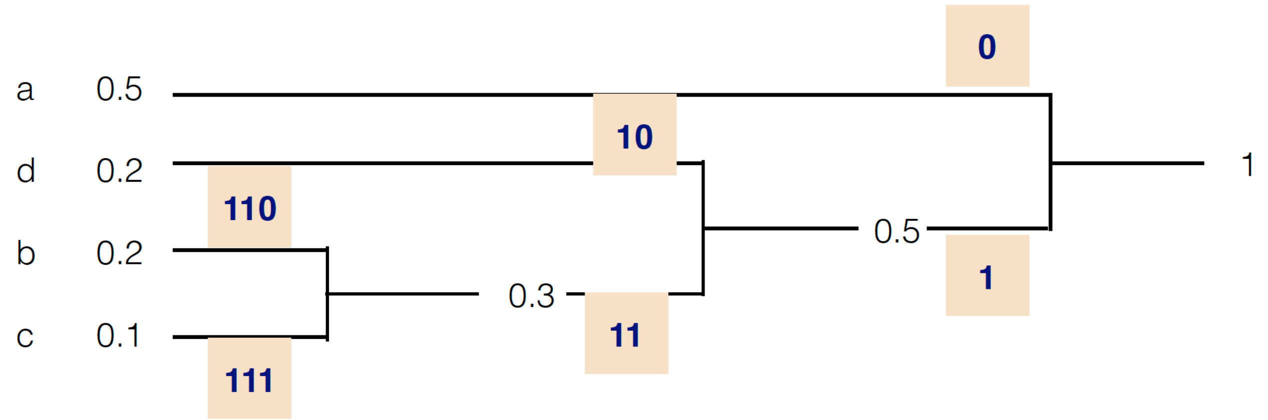
21	a	0.5	**1**	
95	b	0.1	**000**	
169	c	0.2	**01**	
243	d	0.2	**001**	codeword



Huffman coding

21	a	0.5	**1**		**0**
95	b	0.1	**000**	or	**111**
169	c	0.2	**01**		**10**
243	d	0.2	**001**		**110**

Huffman code is not unique



Message Encoding

Given a code, it is easy to encode the message by replacing the symbols by the **codewords**

$\Gamma = \{a, b, c, d\}$

Fixed length code:	$C_1\{a = 00,\ b = 01,\ c = 10,\ d = 11\}$
Variable length codes	$C_2 = \{a = 0,\ b = 110,\ c = 10,\ d = 111\}$ $C_3 = \{a = 1,\ b = 110,\ c = 10,\ d = 111\}$

Code the word: **bad**

- using C_1 **010011**
- using C_2 **1100111**
- using C_3 **1101111**

Are these codes uniquely decodable?

Message Encoding

Given a code, it is easy to encode the message by replacing the symbols by the **codewords**

$$\Gamma = \{a, b, c, d\}$$

Fixed length code:	$C_1\{a = 00,\ b = 01,\ c = 10,\ d = 11\}$
Variable length codes	$C_2 = \{a = 0,\ b = 110,\ c = 10,\ d = 111\}$ $C_3 = \{a = 1,\ b = 110,\ c = 10,\ d = 111\}$

Are these codes uniquely decodable?

- Decode **010011** using C_1 Yes, fixed length code, always uniquely decodable
- Decode **1100111** using C_2
- Decode **1101111** using C_3

Message Encoding

Given a code, it is easy to encode the message by replacing the symbols by the **codewords**

	$\Gamma = \{a, b, c, d\}$
Fixed length code:	$C_1\{a = 00,\ b = 01,\ c = 10,\ d = 11\}$
Variable length codes	$C_2 = \{a = 0,\ b = 110,\ c = 10,\ d = 111\}$
	$C_3 = \{a = 1,\ b = 110,\ c = 10,\ d = 111\}$

Are these codes uniquely decodable?

- Decode **010011** using C_1
- Decode **1100111** using C_2
- Decode **1101111** using C_3 No, it can be decoded as **bad**

1101111 or **acda**

1101111 or **acad**

Message Encoding

Given a code, it is easy to encode the message by replacing the symbols by the **codewords**

$$\Gamma = \{a, b, c, d\}$$

Fixed length code:	$C_1\{a = 00,\ b = 01,\ c = 10,\ d = 11\}$
Variable length codes	$C_2 = \{a = 0,\ b = 110,\ c = 10,\ d = 111\}$ $C_3 \;=\; \{a = 1,\ b = 110,\ c = 10,\ d = 111\}$

Are these codes uniquely decodable?

Huffman code is prefix-free, and it is uniquely decodable.

- Decode **010011** using C_1
- Decode **1100111** using C_2 Yes, no code is a prefix to another code (prefix-free code)
- Decode **1101111** using C_3

Fixed/Variable length coding

$$\mathbf{I} = \begin{bmatrix} 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \end{bmatrix}$$

Fixed length coding

$$\begin{vmatrix} 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \end{vmatrix}$$

$$4 \times 5 \times 8\text{bits} = 160\text{bits}$$

00 (for 21), **01** (for 95),
10 (for 169) and **11** (for 243)

$$\begin{vmatrix} 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \end{vmatrix}$$

$$4 \times 5 \times 2\text{bits} = 40\text{bits}$$

Can we do better?

Yes!

Variable length coding (Huffman coding)

21	a	0.5	**1**
95	b	0.1	**000**
169	c	0.2	**01**
243	d	0.2	**001**

$$\begin{vmatrix} 1 & 1 & 1 & 01 & 001 \\ 1 & 1 & 1 & 01 & 001 \\ 1 & 1 & 000 & 01 & 001 \\ 1 & 1 & 000 & 01 & 001 \end{vmatrix}$$

$$10\text{x}1 + 2\text{x}3 + 4\text{x}2 + 4\text{x}3 = 36\text{bits}$$

Fixed/Variable length coding

$$\mathbf{I} = \begin{array}{ccccc} 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \end{array}$$

Fixed length coding

$$\begin{vmatrix} 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 00010101 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \\ 00010101 & 00010101 & 01011111 & 10101001 & 11110011 \end{vmatrix}$$

$$4 \times 5 \times 8\text{bits} = 160\text{bits}$$

00 (for 21), **01** (for 95),
10 (for 169) and **11** (for 243)

$$\begin{vmatrix} 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 00 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \\ 00 & 00 & 01 & 10 & 11 \end{vmatrix}$$

$$4 \times 5 \times 2\text{bits} = 40\text{bits}$$

Variable length coding (Huffman coding)

Different **codewords**

21	a	0.5	**1**		**0**
95	b	0.1	**000**	or	**111**
169	c	0.2	**01**		**10**
243	d	0.2	**001**		**110**

$$\begin{vmatrix} 0 & 0 & 0 & 10 & 110 \\ 0 & 0 & 0 & 10 & 110 \\ 0 & 0 & 111 & 10 & 110 \\ 0 & 0 & 111 & 10 & 110 \end{vmatrix}$$

$$10\text{x}1 + 2\text{x}3 + 4\text{x}2 + 4\text{x}3 = 36\text{bits}$$

Fixed/Variable length coding

$$I = \begin{array}{|ccccc|} \hline 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ \hline \end{array}$$

	Codewords	Total bits	Average code length
Fixed length coding	**00** (for 21), **01** (for 95), **10** (for 169) and **11** (for 243)	$4 \times 5 \times 2\text{bits}$ $= 40\text{bits}$ (20 symbols)	2bits/symbol
Variable length coding (Huffman coding)	21 a 0.5 **1** or **0** 95 b 0.1 **000** or **111** 169 c 0.2 **01** or **10** 243 d 0.2 **001** or **110**	10x1 + 2x3 + 4x2 + 4x3 $= 36\text{bits}$ (20 symbols)	1.8bits/symbol

Assign short (binary) codeword to numbers (symbols) with higher probability, and longer codeword to the ones with lower probability.

Question: is Huffman coding the **optimal solution?**

Entropy coding

- Entropy coding is a lossless compression technique applicable to **any** data.
- Entropy is a fundamental concept in **information theory.**
- **Entropy** is a measure of **information**.

Entropy

Shannon (1948) proposed to measure information in terms of uncertainty or randomness in data

- **Entropy** is a measure of uncertainty in data.

A source $\mathbf{z}$ generates two symbols: a_1, a_2

Probabilities: $Prob(a_1) = p, Prob(a_2) = 1 - p$

Entropy is defined as

$$H(\mathbf{z}) = p\log\frac{1}{p} + (1-p)\log\frac{1}{(1-p)}$$

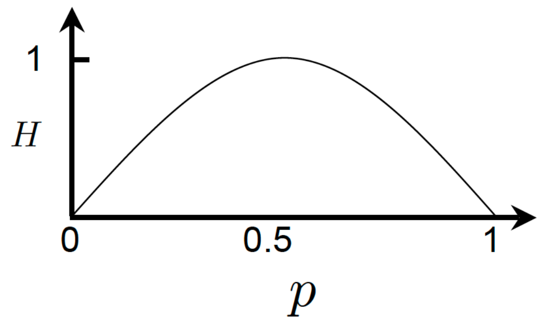
Higher the entropy, higher is the uncertainty in data.

Entropy

Shannon (1948) proposed to measure information in terms of uncertainty or randomness in data

- **Entropy** is a measure of uncertainty in data.

$$H(\mathbf{z}) = p\log\frac{1}{p} + (1-p)\log\frac{1}{(1-p)}$$



For example, toss a normal coin, with P(head)=P(tail)=0.5

$$H(\mathbf{z}) \;= -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 0.5 + 0.5 = 1$$

Largest uncertainty

Toss a double-headed coin, with P(head)=1, P(tail)=0

$$H(\mathbf{z}) \;= -1 \times \log_2 1 - 0 \times \log_2 0 = 0 + 0 = 0$$

No uncertainty

Higher the entropy, higher is the uncertainty in data.

Entropy coding

- Fundamental to **any** data compression. Entropy coding is a lossless compression technique.
- **Entropy** is a measure of uncertainty in data.

A source **z** generates $a_1, a_2, \ldots a_n$

Probabilities: $Prob(a_1) = p_1, Prob(a_2) = p_2, \ldots..$ such that $\sum_{j=1}^{n} p_j = 1$

Entropy is defined as

$$H(\mathbf{z}) = -\sum_{j=1}^{n} p_i \log p_i$$

Shannon's source coding theorem

- In any (uniquely decodable) coding scheme, the average codeword length of a source (of symbols) can at best be equal to the source entropy, and can not be less than it.

- Entropy is the **bound** on maximum compression that can be achieved using <u>entropy coding.</u>

Fixed/Variable length coding

$$\mathrm{I} = \begin{bmatrix} 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 21 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \\ 21 & 21 & 95 & 169 & 243 \end{bmatrix}$$

	Codewords	Total bits	Average code length
Fixed length coding	**00** (for 21), **01** (for 95), **10** (for 169) and **11** (for 243)	$4 \times 5 \times 2\text{bits} = 40\text{bits}$ (20 symbols)	2bits/symbol
Variable length coding (Huffman coding)	21 a 0.5 **1** or **0** 95 b 0.1 **000** or **111** 169 c 0.2 **01** or **10** 243 d 0.2 **001** or **110**	10x1 + 2x3 + 4x2 + 4x3 = 36bits (20 symbols)	1.8bits/symbol

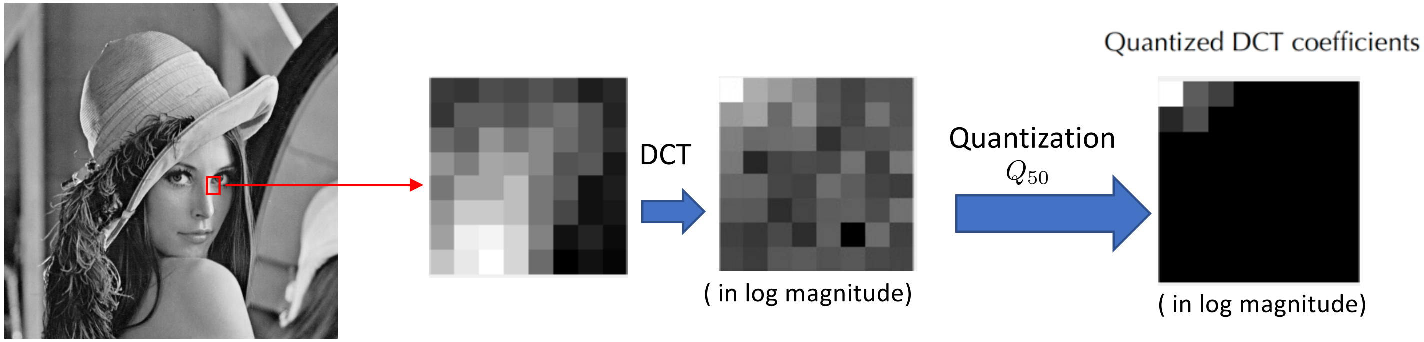
Entropy of the source:

$$H(\mathbf{z}) = -\sum_{j=1}^{n} p_i \log p_i$$

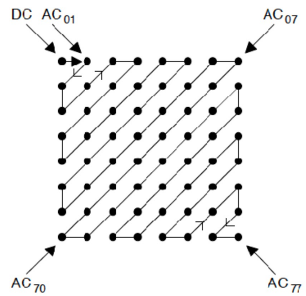
$$= -0.5\log 0.5 - 0.1\log 0.1 - 0.2\log 0.2 - 0.2\log 0.2 = 1.76(\text{bits/symbol})$$

Huffman coding can be deemed as the optimal solution

Back to Lena Example



85	5	−3	−1	0	0	0	0
−2	−4	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	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



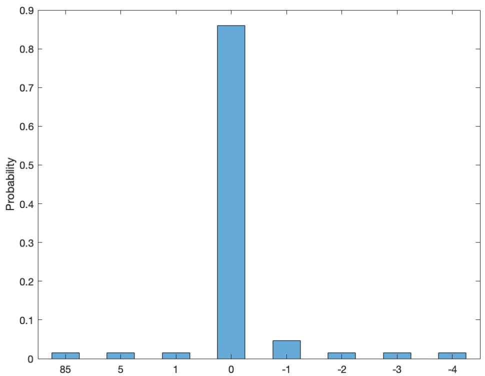
$$85, 5, -2, -1, -4, -3, -1, 0, 0, -1, 0, ..., ...0, 0, 0, 0, 0, ..., 0, 0, 0, 0, 0, 0$$

Huffman coding

Huffman coding (Quality level 50)

Quantized DCT coefficients

$$\begin{matrix} 85 & 5 & -3 & -1 & 0 & 0 & 0 & 0 \\ -2 & -4 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$



Huffman coding:

value	Freq.	Huffman code
0	56	1
-1	2	000
85	1	0010
5	1	0011
1	1	0100
-2	1	0101
-3	1	0110
-4	1	0111

Total bits: $1 \times 56 + 3 \times 2 + (4 \times 1) \times 6 = 86\text{bits}$

Average code length: $86/64 = 1.34 \text{ bits/symbol}$

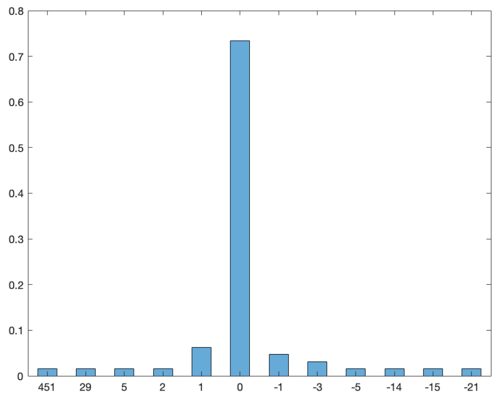
Entropy: (theoretical bound) $H(\mathbf{z}) = -\sum_{j=1}^{n} p_i \log p_i = 0.89 \text{ bits/symbol}$

Question: if we use fixed length coding, how many bits are required (for the 8*8 block)?

Huffman coding (Quality level 90)

Quantized DCT coefficients

451	29	−15	−5	0	0	0	0
−14	−21	1	5	0	0	1	0
−3	1	2	−1	0	0	0	0
−3	0	0	0	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	0	0
0	0	0	0	0	0	0	0



Huffman coding:

value	Freq.	Huffman code
0	47	1
1	4	000
-1	3	0111
-3	2	0010
29	1	00110
5	1	00111
2	1	01000
451	1	01001
-5	1	01010
-14	1	01011
-15	1	01100
-21	1	01101

Total bits: $1 \times 47 + 3 \times 4 + 4 \times 3 + 4 \times 2 + (5 \times 1) \times 8 = 119\text{bits}$

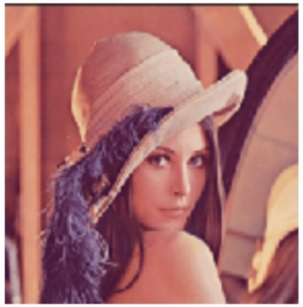
Average code length: $119/64 = 1.86 \text{ bits/symbol}$

Entropy:
(theoretical bound)

$$H(\mathbf{z}) = -\sum_{j=1}^{n} p_i \log p_i \;=\; 1.69 \text{ bits/symbol}$$

Question: if we use fixed length coding, how many bits are required (for the 8*8 block)?

JPEG compression steps



Original

Redundancy

- Spatial redundancy
- Psychovisual redundancy
- Coding redundancy

...

value	Freq.	Huffman code
0	56	1
-1	2	000
85	1	0010
5	1	0011
1	1	0100
-2	1	0101
-3	1	0110
-4	1	0111

...

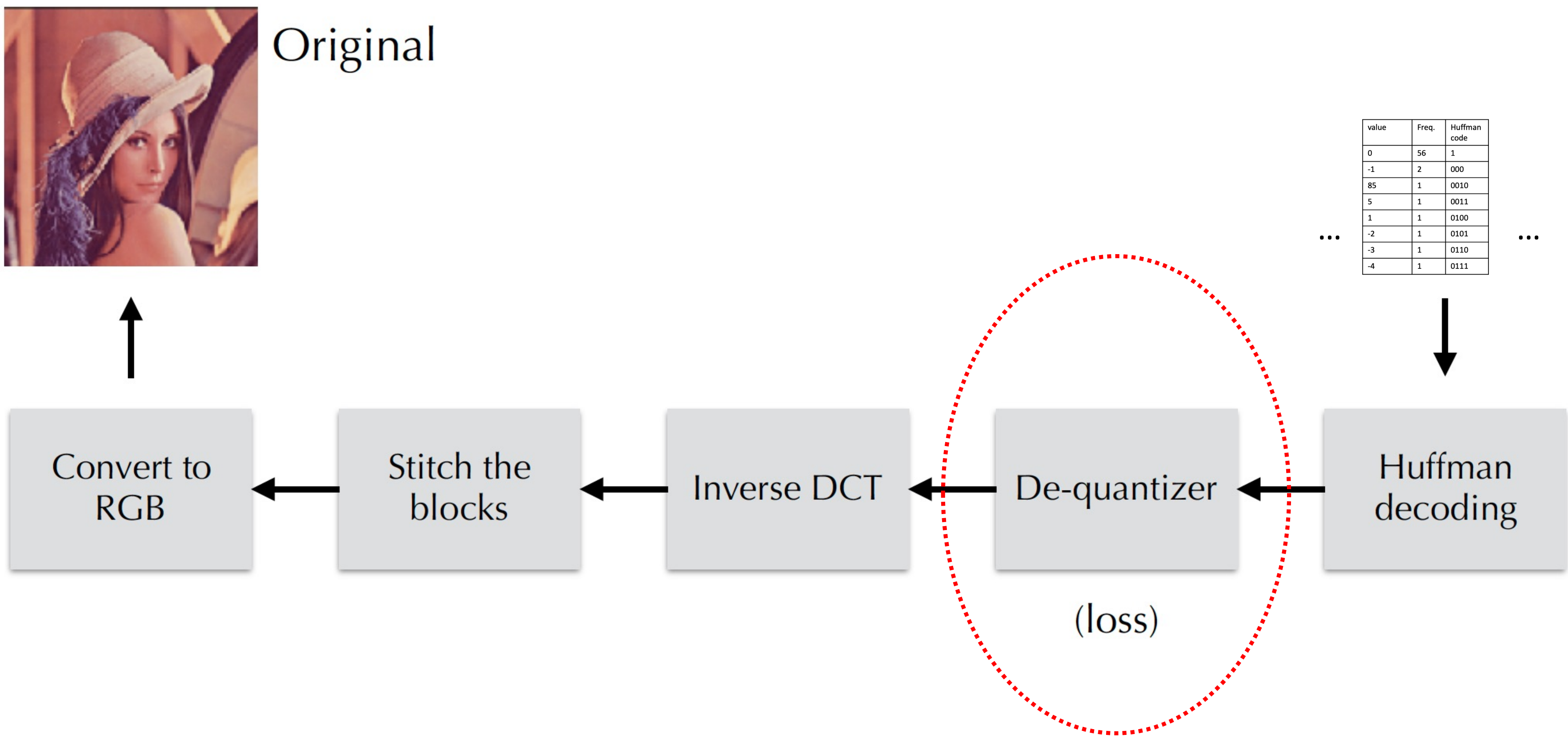
Color space conversion → Division into subimages → Discrete Cosine Transform → Quantizer → Huffman coding

Discrete Cosine Transform: Removes spatial redundancy

Quantizer: Removes psychovisual redundancy

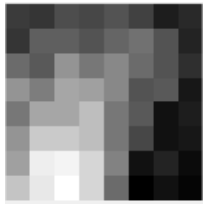
Huffman coding: Removes coding redundancy

JPEG de-compression steps

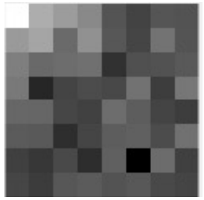


value	Freq.	Huffman code
0	56	1
-1	2	000
85	1	0010
5	1	0011
1	1	0100
-2	1	0101
-3	1	0110
-4	1	0111

$$Q_{50} = \big(Q(u,v)\big) \quad \begin{matrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{matrix}$$



DCT



(in log magnitude)

$$F(u,v) \quad \begin{matrix} 1352 & 58 & -30 & -14 & -2 & 1 & -2 & 2 \\ -28 & -43 & 4 & 19 & -2 & -1 & 6 & 2 \\ -10 & 4 & 5 & -6 & 0 & -2 & -2 & 2 \\ -8 & 0 & -1 & -1 & 1 & -6 & 1 & -7 \\ -4 & 4 & 1 & -1 & 5 & 3 & 1 & -1 \\ 2 & 2 & 0 & -1 & 3 & 3 & -1 & 6 \\ -1 & 1 & 1 & 0 & 3 & 0 & 5 & 1 \\ -1 & -1 & -2 & -2 & 2 & -2 & 1 & -1 \end{matrix}$$

Quantization
Q_{50}

(in log magnitude)

$$F_Q(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

$$\begin{matrix} 85 & 5 & -3 & -1 & 0 & 0 & 0 & 0 \\ -2 & -4 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

value	Freq.	Huffman code
0	56	1
-1	2	000
85	1	0010
5	1	0011
1	1	0100
-2	1	0101
-3	1	0110
-4	1	0111

Quantization error: $\epsilon = |F(u,v) - \hat{F}(u,v)|$

De-quantized DCT coefficients

$$\begin{matrix} 1360 & 55 & -30 & -16 & 0 & 0 & 0 & 0 \\ -24 & -48 & 0 & 19 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\hat{F}(u,v) = F_Q(u,v)Q(u,v)$$

De-Quantization
Q_{50}

(in log magnitude)

Next, we will take advantage of this quantization error for **forensic applications**

Further reading

Digital Image processing
By Gonzalez and Woods.

Chapter 8 Image Compression and Watermarking
8.1 Fundamentals
8.2 Huffman coding