Image Compression (1)

Qiufeng Wang

Qiufeng.Wang@xjtlu.edu.cn

Outline

- Introduction to compression (source coding)
  Why, what, how compression
- Concept of lossless and lossy compression
- Types of redundancy
- Fixed-length codes
- Variable length codes

Why image compression? ---Data storage

- Comparison of common broadcast resolutions

Format	Resolution	Display aspect ratio	Pixels
Ultra-high-definition television	3840 × 2160	1.78:1 (16:9)	8,294,400
Ultra-wide-television	5120 × 2160	2.37:1 (21:9)	11,059,200
DCI 4K (native resolution)	4096 × 2160	1.90:1 (256:135)	8,847,360
DCI 4K (CinemaScope cropped)	4096 × 1716	2.39:1 (1024:429)	7,028,736
DCI 4K (flat cropped)	3996 × 2160	1.85:1 (999:540)	8,631,360

From Wikipedia.org

Why image compression?
---Data storage

- Comparison of common broadcast resolutions

Format	Resolution	Display aspect ratio	Pixels
Ultra-high-definition television	3840 × 2160	1.78:1 (16:9)	8,294,400
Ultra-wide-television	5120 × 2160	2.37:1 (21:9)	11,059,200
DCI 4K (native resolution)	4096 × 2160	1.90:1 (256:135)	8,847,360
DCI 4K (CinemaScope cropped)	4096 × 1716	2.39:1 (1024:429)	7,028,736
DCI 4K (flat cropped)	3996 × 2160	1.85:1 (999:540)	8,631,360

One image : 4096 ×2160 ×3 = 26.5 M Byte
One hour video : 4096 ×2160 ×3 ×30 ×3600 = 2862 GByte

From Wikipedia.org

Why image compression? ---Data storage

- Image data can take a lot of space. An example of storage requirement for a full-frame camera (Nikon D600 - RAW 14 bit):

$$\underbrace{(6016 \times 4016)}_{\text{Spatial resolution}} \underbrace{\times 14}_{\text{bit/channel}} = 338243584 \text{ bits}$$

$$= 42\text{MB}$$

From Wikipedia.org

Why image compression?
---Data usage pattern

- Symmetric applications
  - Data is typically compressed and decompressed the same number of times
  - Compression and decompression involves roughly the same complexity
    - Example: video telephony

Telepresence system, By Fuelrefuel - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5146472

Why image compression?
---Data usage pattern

- Asymmetric applications
  - Data is typically compressed once and decompressed many times
  - Compression is computationally expensive, decompression is simple
    - Movie compression

Why Image Compression ---Benefits

- Image compression has the following benefits:
  - Less memory for storage.
  - Reduced transmission time and costs.
  - Reduced the required memory for computation.

Why image compression?
--- Redundancy

BMP: 258KB

JPG: 82KB

What is Compression

- The fundamental task of image compression is to reduce the amount of data (bytes) required to represent an image (**information**).
  - We do this by removing image data redundancies.

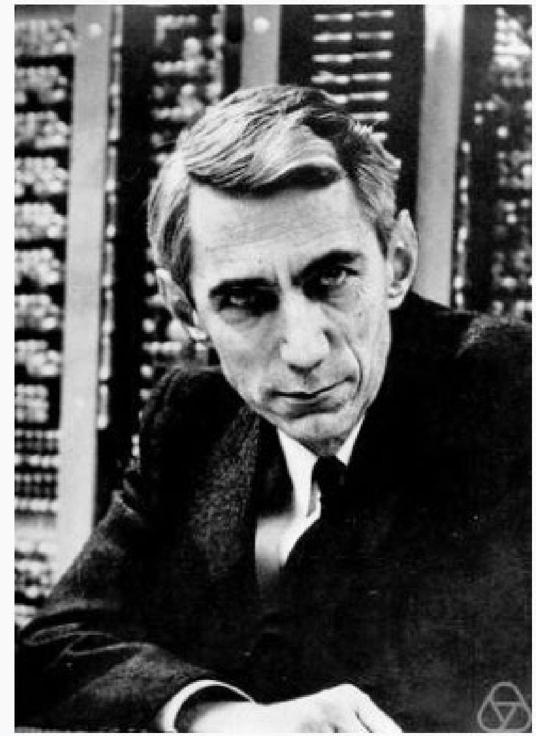
Data vs. information

- Data (bytes) is not equal to information
- Data is the means by which information is conveyed
  - The same story can be told with a different number of words if the teller is long-winded or short and to the point!
- There are different ways to represent the same information.

Data vs. information

- Information theory address one essential problem: How much data is needed in order to store a certain amount of information?

Claude Shannon



Born	April 30, 1916 Petoskey, Michigan, United States
Died	February 24, 2001 (aged 84) Medford, Massachusetts, United States
Nationality	American

Measuring information

- A discrete *memoryless* source generates symbols from a set X of M elements (alphabet); each symbol is characterized by its probability of occurrence p_i

$$X = \{x_i\}_{i=1}^{M} \qquad \{p_i\}_{i=1}^{M}$$

- How do we measure the amount of information carried by message x_i ?

Properties of information

- The amount of information carried by a message is inversely proportional to its probability

$$I(x_j) > I(x_i) \quad \text{if} \quad p_j < p_i$$

- Statistically independent messages:

$$P(x_i, x_j) = P(x_i)P(x_j) \;\; \Rightarrow \;\; I(x_i, x_j) = I(x_i) + I(x_j)$$

Definition of "information"

$$I(x_i) = \log_2 \frac{1}{p_i}$$

and is measured in bits

- Average amount of information carried by the memoryless source: *first order entropy* (bits/symbol)

$$H(X) = \sum_{i=1}^{M} p_i I(x_i) = \sum_{i=1}^{M} p_i \log_2 \frac{1}{p_i}$$

Examples

$$X = \begin{array}{ll} x_1 & p_1 = 1/2 \\ x_2 & p_2 = 1/4 \\ x_3 & p_3 = 1/8 \\ x_4 & p_4 = 1/8 \end{array}$$

H(X) = 1.75 bits/symbol

$$X = \begin{array}{ll} x_1 & p_1 = 1/4 \\ x_2 & p_2 = 1/4 \\ x_3 & p_3 = 1/4 \\ x_4 & p_4 = 1/4 \end{array}$$

H(X) = 2 bits/symbol

Equiprobable symbols carry more information, and are more difficult to compress

Bounds on Entropy

- ***Theorem:*** the first order entropy of a memoryless M-symbol alphabet is limited by

$$\boxed{H(X) \le \log_2 M}$$

- Example: 8 bit quantizer ($M=2^8$)

$$\boxed{H(X) \le 8 \text{ bit/symbol}}$$

with the equality if the symbols are equiprobable

Entropy …

- What is the name of this value:

$$-\sum_{i=1}^{M} p_i \log_2 p_i = \sum_{i=1}^{M} p_i \log_2 \frac{1}{p_i} [bit/symbol]$$

The first order Entropy of the source (*H*)

- The average amount of information carried by a source is the *entropy*

- The amount of information carried by a symbol is $\log_2 \frac{1}{p_i}$

- $\log_2 \frac{1}{p_i}$ is the uncertainty in symbol e_i (or the "surprise" when we see this symbol). Entropy – average "surprise".

Entropy of images

- Entropy in Matlab :
  - » h = entropy(uint8(im))

1st order Entropy = 7.4

1st order Entropy = 6.2

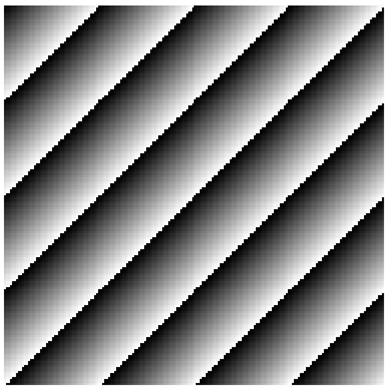
White is the most likely value in this picture

How to quantify compression efficiency ?

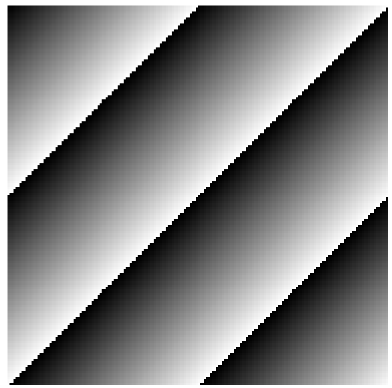
- Since there are different ways to represent the same information, then how to quantify which representation is better in term of compression efficiency ?

Compression ratio

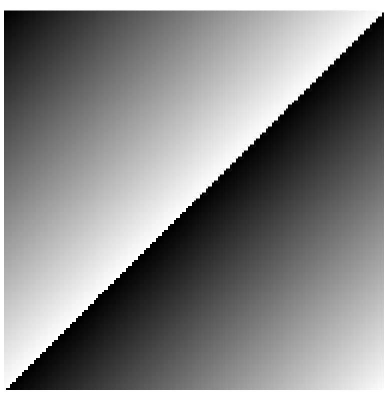
Size (bmp) = 17462



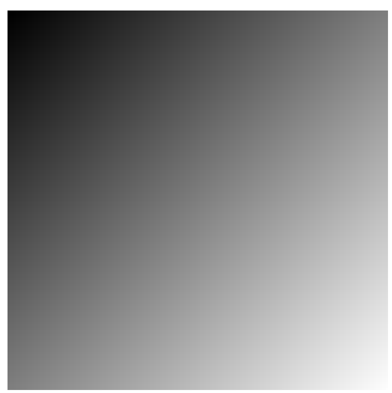
Size (bmp) = 17462



Size (bmp) = 17462



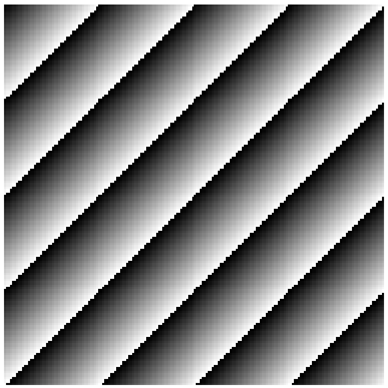
Size (bmp) = 17462



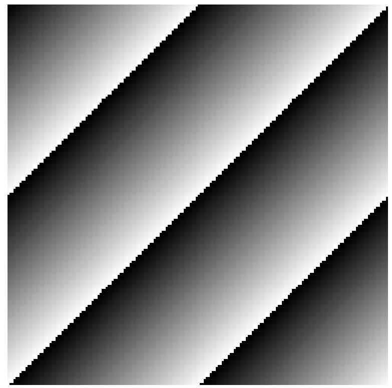
- The monochrome image is 128x128 pixels (**symbols of information**)

Compression ratio

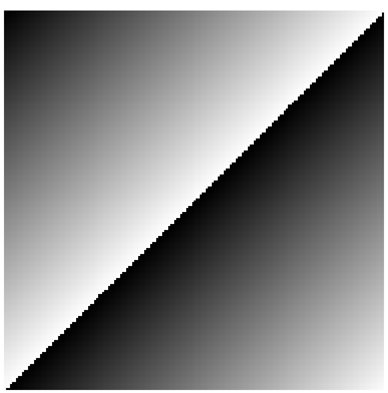
Size (jpg) = 10872,



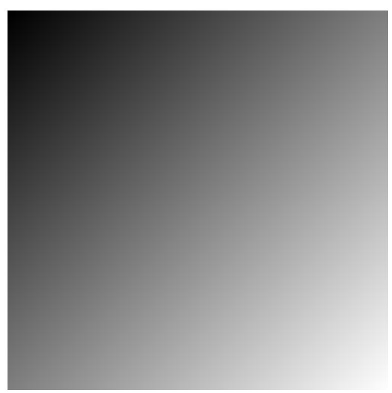
Size (jpg) = 6972,



Size (jpg) = 4508,



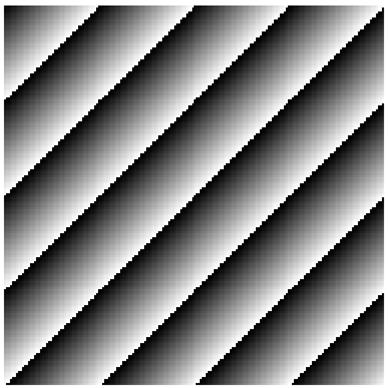
Size (jpg) = 2762,



- The monochrome image is 128x128 pixels (**symbols of information**)

Compression ratio

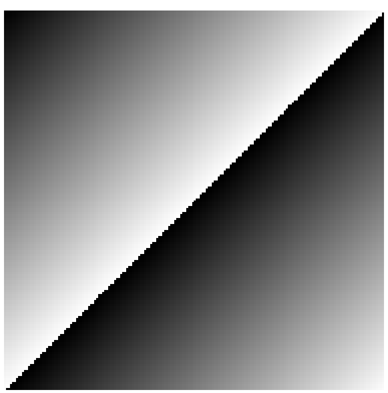
Size (jpg) = 10872, CR = 1.6



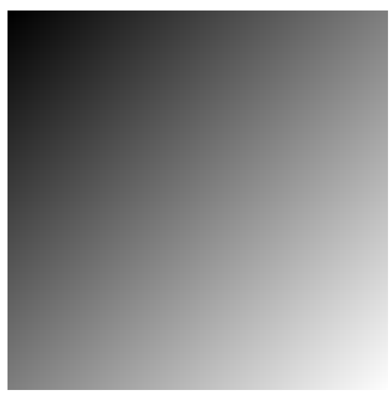
Size (jpg) = 6972, CR = 2.5

6.3 = 17462/2762

Size (jpg) = 4508, CR = 3.9



Size (jpg) = 2762, CR = 6.3



- The monochrome image is 128x128 pixels (**symbols of information**)
- Let n_1 and n_2 be the number of *data-unit* (coding words) in two data sets that represents the same information then

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

Compression ratio

- A compression algorithm searches a representation with:

$$CR > 1$$

Compression types

- There are two major categories of compression algorithms

1. Lossless compression algorithms:
    - The original data is recovered perfectly after decompression.
    - There is a theoretical limit on maximal compression. Depend purely on the image content.

    - Practical compression ratios CR<10 (still images).
    - Where it is used ? Medical, Hyperspectral, aerial, …

Compression types

- There are two major categories of compression algorithms

2. Lossy compression algorithms:
    - Decompression results in an approximation of the original image.
    - Compression rate is a function of reconstruction quality.

    - Practical compression ratios CR >10 (still images).
    - Where it is used ? Application where the end user is human who will view the media content.

Lossless - Introduction

- The goal of lossless compression is

to minimize the average length
of the compressed symbols

exploiting statistical properties of the data
  – *probability distribution*
  – *correlation (redundancy) of the data*

How to compress data?

- How to compress data?

- This could be achieved by finding a more compact signal representation by reducing the intrinsic redundancy (duplication) of the source signal

Types of redundancy

Types of redundancy

- Typical signals contain redundancy which could be exploited:
  - Coding (data representation) redundancy
  - Interpixel redundancy (Correlation between adjacent samples)
  - Psychovisual redundancy.

Types of redundancy

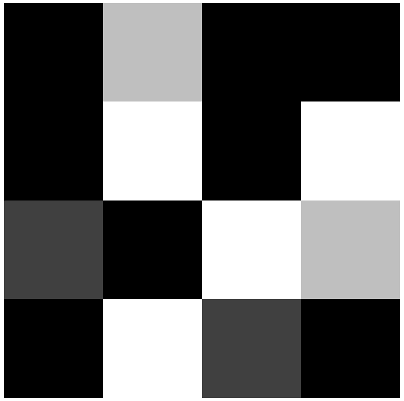
- In general, coding and interpixel redundancies can be exploited without losing any information → lossless compression.

- In general, lossy compression achieves much higher compression rates than lossless compression by exploiting the psychovisual redundancy

Coding redundancy

- This type of redundancy deals with the way we represent data.

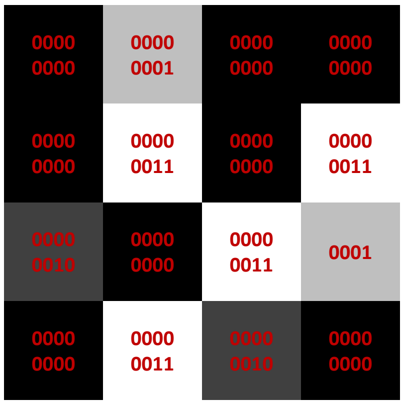
How to represent the information ?



8 bit/symbol (pixel)

- How to write this image on a file ?
- How are we going to represent each pixel ?

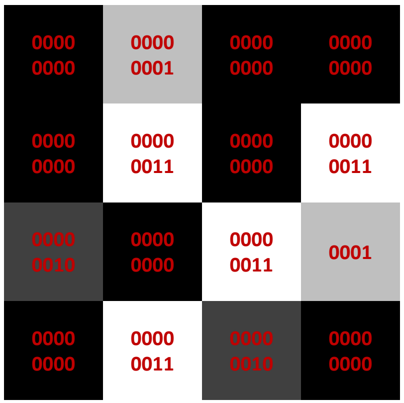
Fixed-Length Code



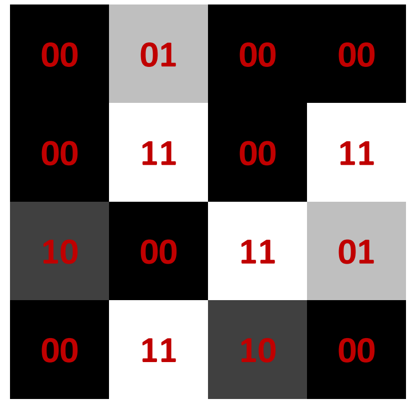
8 bit/symbol

- Is it efficient to use 8 bits per symbol for such image ?

Fixed-Length Code



8 bit/symbol



2 bit/symbol

- Which one is more efficient (compact) representation of the information ?

Fixed-Length Codes

- Properties
  - Use the same number of bits to represent all possible symbols (pixels) produced by the source
  - Simplify the decoding process
- Examples
  - American Standard Code for Information Interchange (ASCII) code
  - Bar codes
    - Universal Product Code (UPC) on products in stores
    - Credit card codes

Wikipedia

From Tran

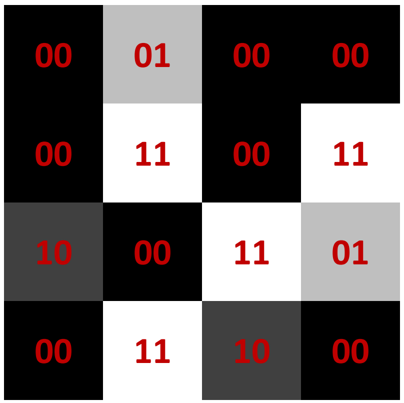
ASCII Code

- ASCII is used to encode and communicate alphanumeric characters for plain text; 7 bits per character

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

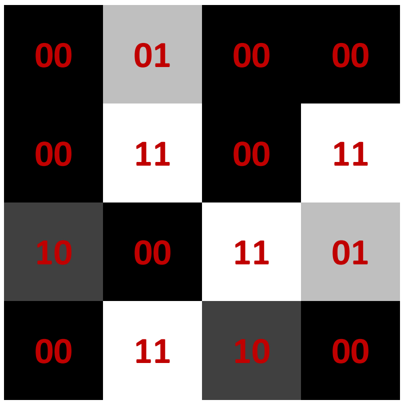
Source: www.asciitable.com

Fixed-Length Code

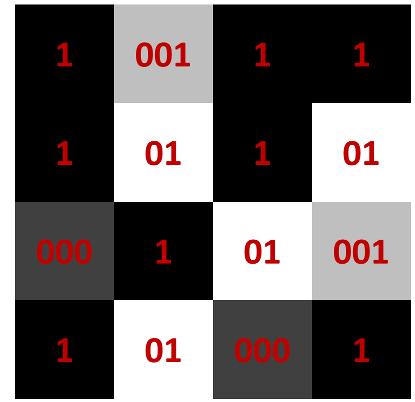


- 4 different symbols we use 2 bit/symbol (fixed-length code)
- Is it the most efficient representation of this information ?

……...-Length Codes



2 bit/symbol



(28/16) bit/symbol

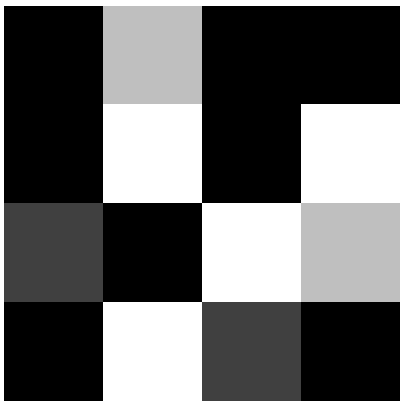
- Which one is the most efficient code ?

- Main problem with fixed-length codes: inefficiency

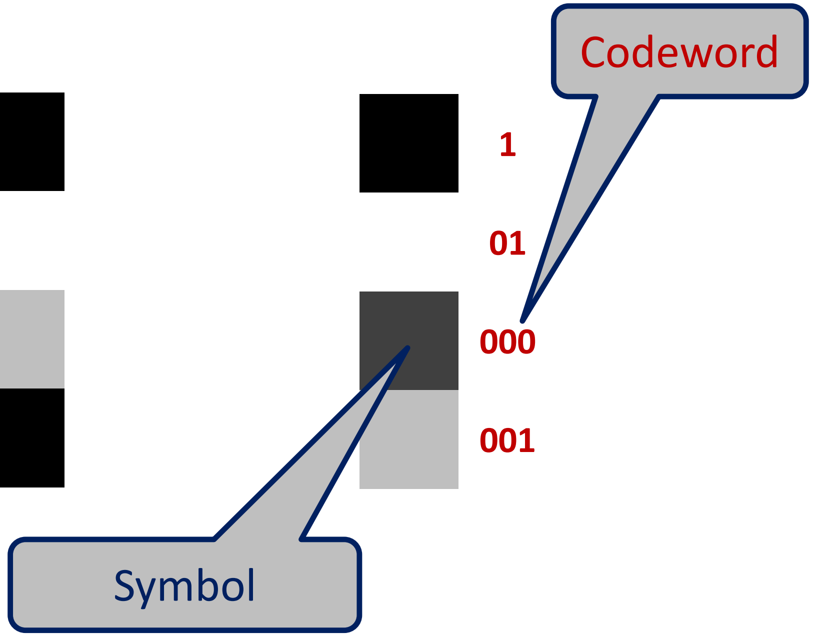
Variable-Length Codes

- Main properties of variable-length codes (VLC)
  - Use a different number of bits to represent each symbol
  - Allocate shorter-length codewords to symbols that occur more frequently
  - Allocate longer-length codewords to rarely-occurred symbols
  - More efficient representation; good for compression

Variable-Length Codes

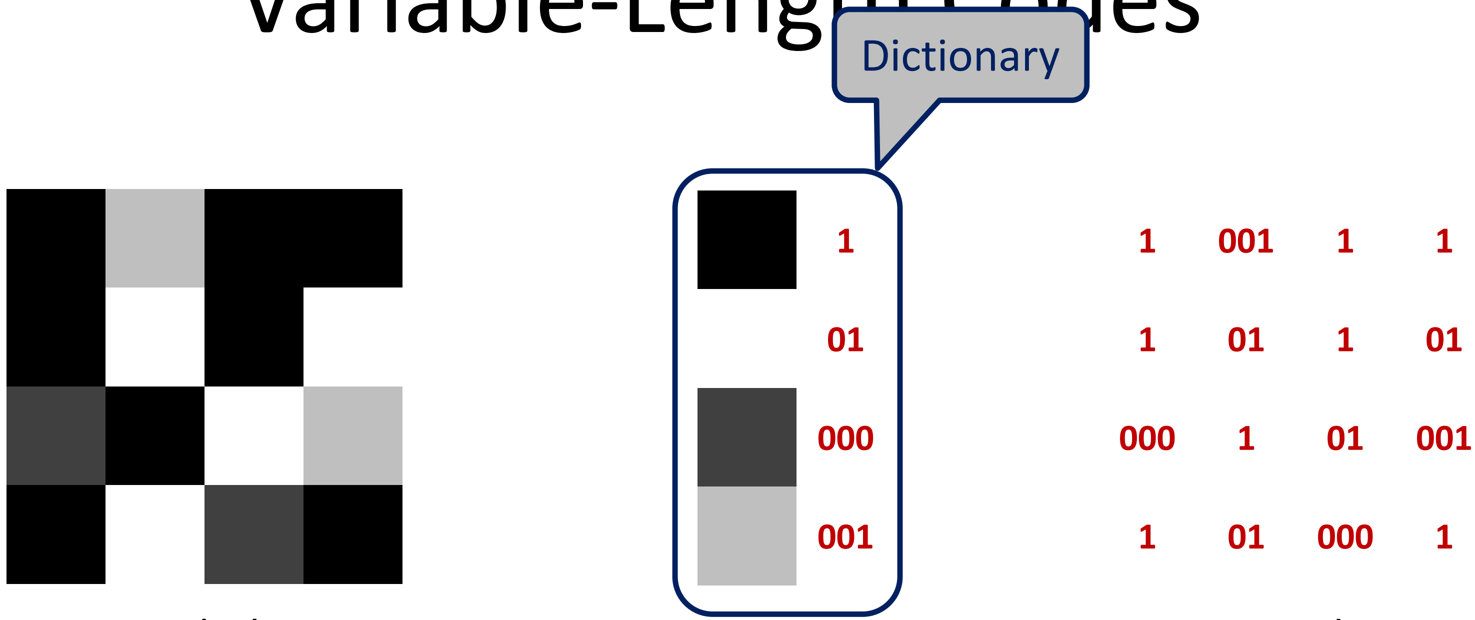


Symbols



- Encoder:
  - Associate the shortest binary *codeword* to the most “probable” symbols

Variable-Length Codes



Symbols

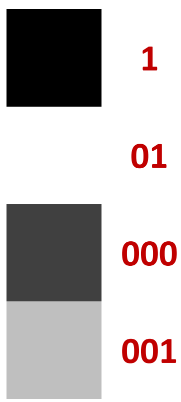
encode

- Encoder:
  - Associate the shortest binary ***codeword*** to the most “probable” symbols
  - Generate a ***dictionary***, encode and send all

Decoding VLC

- Suppose you received a file with these data (and with the dictionary):

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

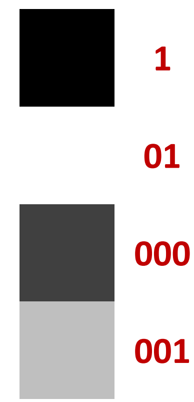


- To generate the information (decode)…

Decoding VLC

- Suppose you received a <u>file</u> with these data (and with the dictionary):

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



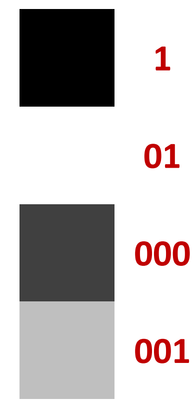
- To generate the information (decode)...

1 1 000 1 001 01 1 01 1 1 01 000 1 01 001 1

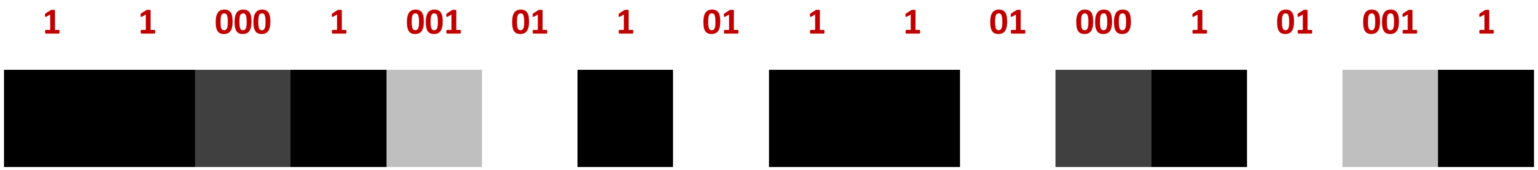
Decoding VLC

- Suppose you received a file with these data (and with the dictionary):

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



- To generate the information (decode)…



The 4x4 image write column-wise

Try decoding this string

1

<u>1</u>1

000

001

- try it ..

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

Variable length coding

- Associate the shortest binary codewords to the most "probable" symbols

1	8/16
01	4/16
000	2/16
001	2/16

- Prefix rule: no codeword should be a prefix of a longer codeword, e.g., 1 is not prefix of 01

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

1 1 000 1 001 01 1 01 1 1 01 000 1 01 001 1

Variable-Length Codes

- Examples of VLC
  - Shannon-Fano code
  - Huffman code

Shannon-Fano Code

- Algorithm
  - Line up symbols by decreasing probability of occurrence
  - Divide symbols into 2 groups so that both have similar combined probability
  - Assign **0** to 1st group and **1** to the 2nd
  - Repeat step 2
- Example

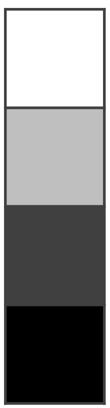
Symbols	Prob.	Code-word
A	0.35	0 0
B	0.17	0 1
C	0.17	1 0
D	0.16	1 1 0
E	0.15	1 1 1

Average code-word length =
$0.35 \times 2 + 0.17 \times 2 + 0.17 \times 2 + 0.16 \times 3 + 0.15 \times 3$
= 2.31 bits per symbol

Huffman Coding

Huffman Coding

- Allows us to build a quasi-optimal VLC



0.25

0.125

0.125

0.5

Huffman Coding

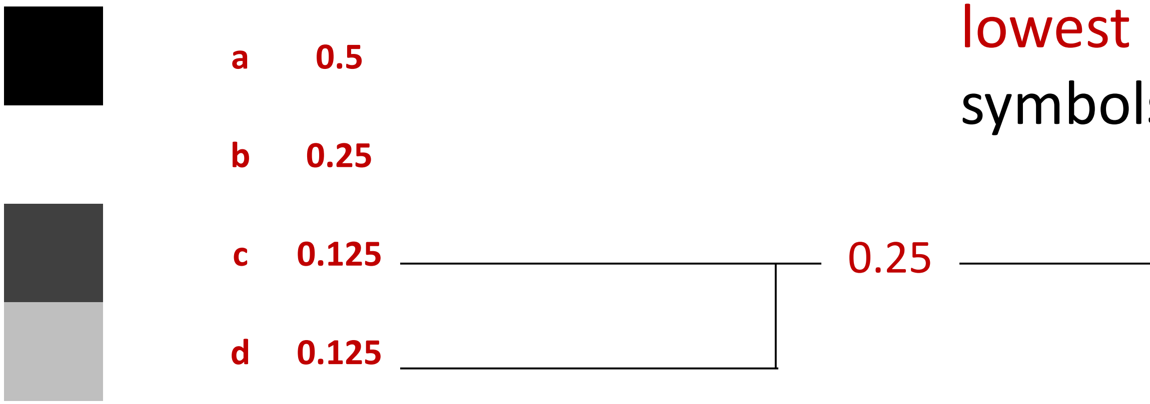
- Building a Huffman tree

a	0.5
b	0.25
c	0.125
d	0.125

- Order the symbols by the descending order of probabilities

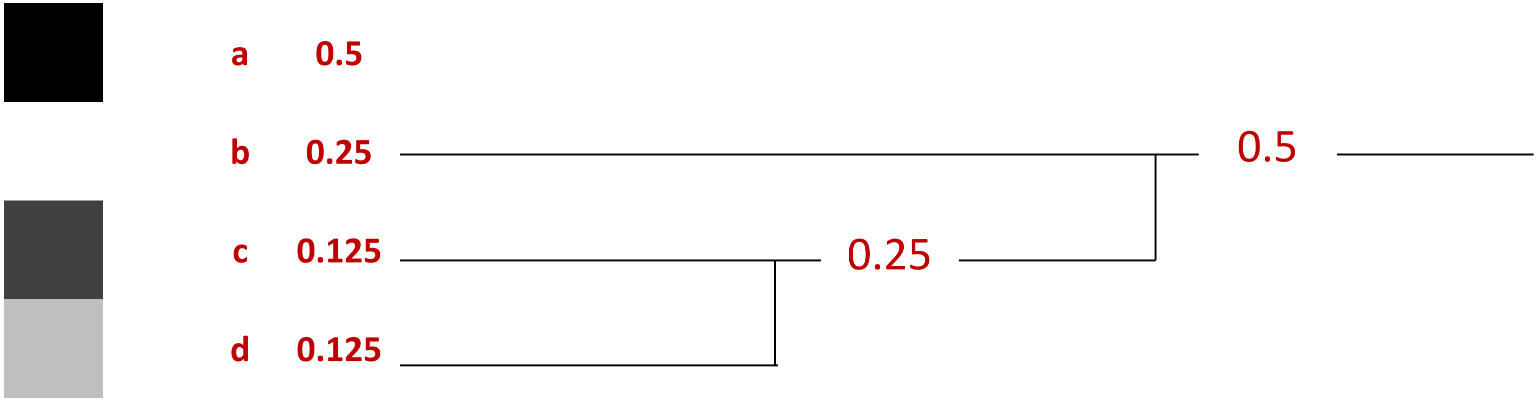
Huffman Coding

- Building a Huffman tree
  - Then combine the symbols of lowest probability to form new symbols recursively



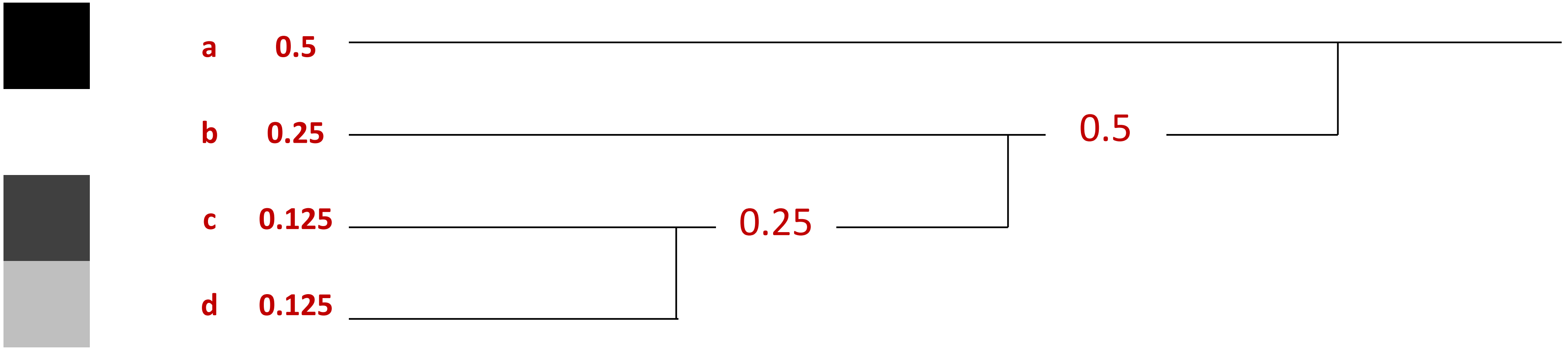
Huffman Coding

- Building a Huffman tree



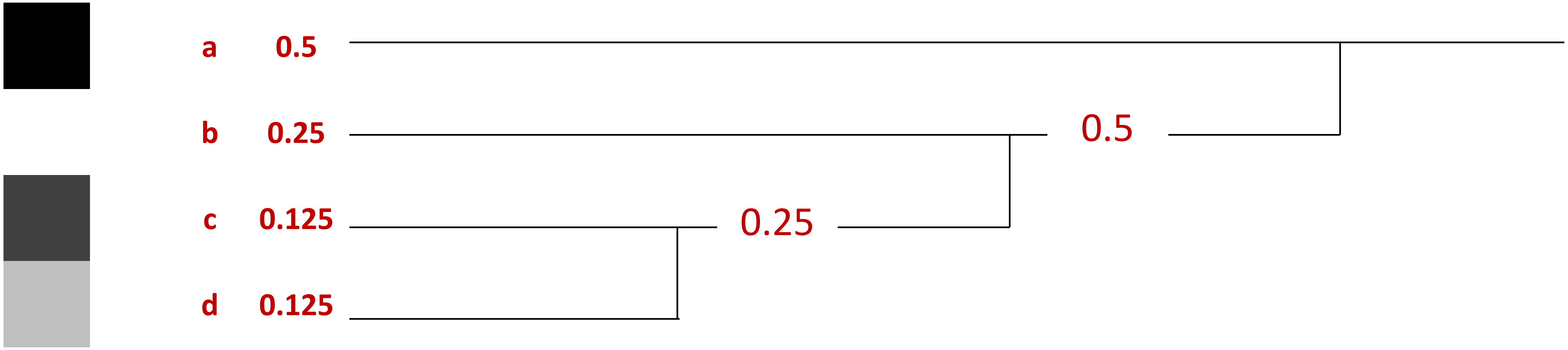
Huffman Coding

- Building a Huffman tree



Huffman Coding

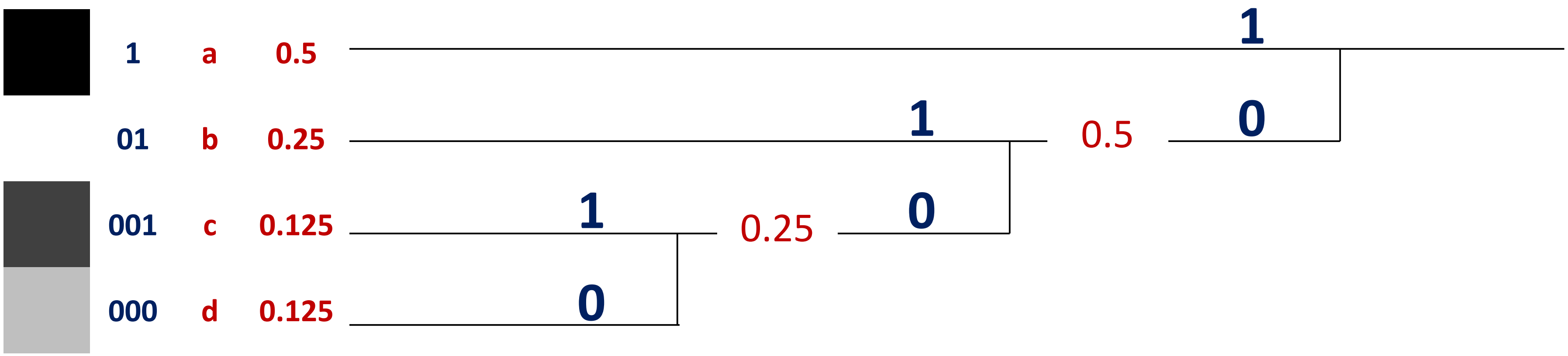
- Code assignment



- When Huffman tree is completely built, then assign a code-bit to the symbols in each branch of the tree

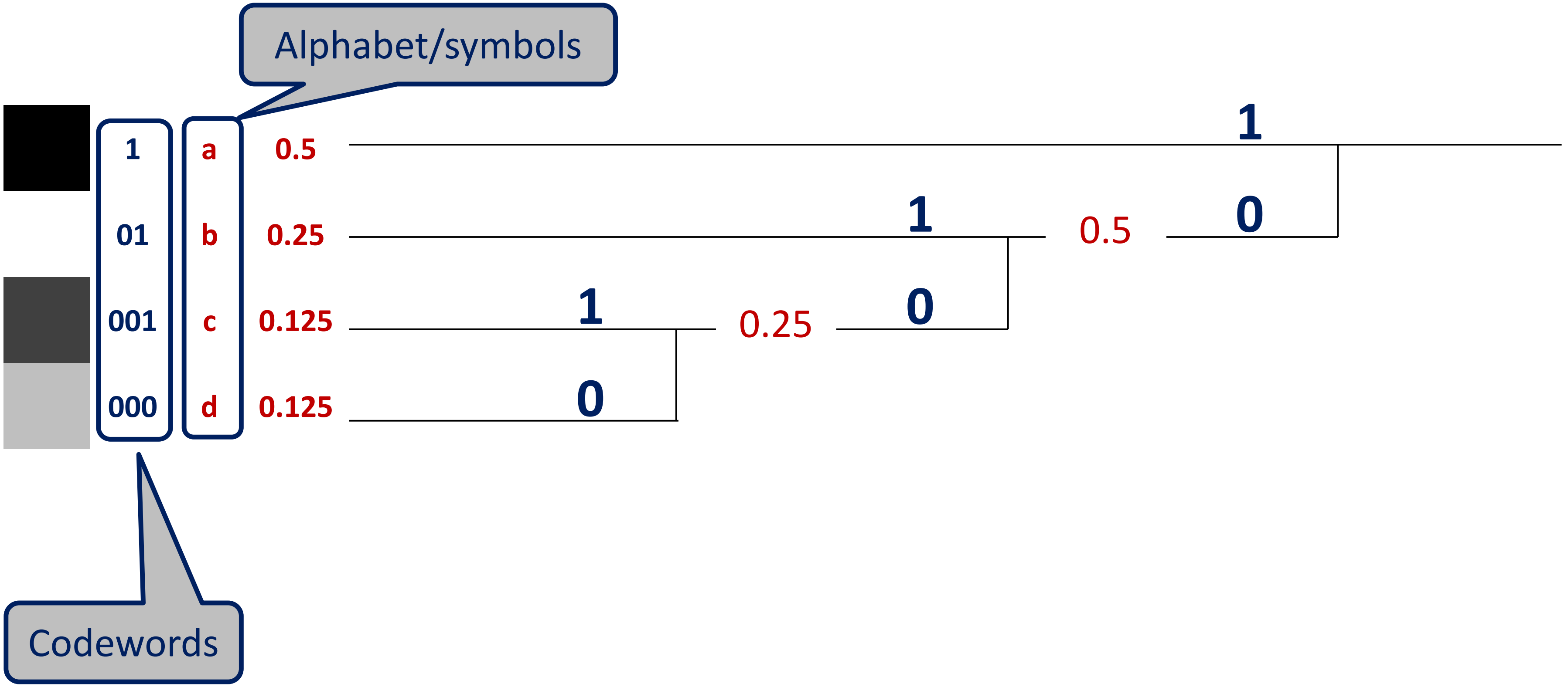
Huffman Coding

- Code assignment



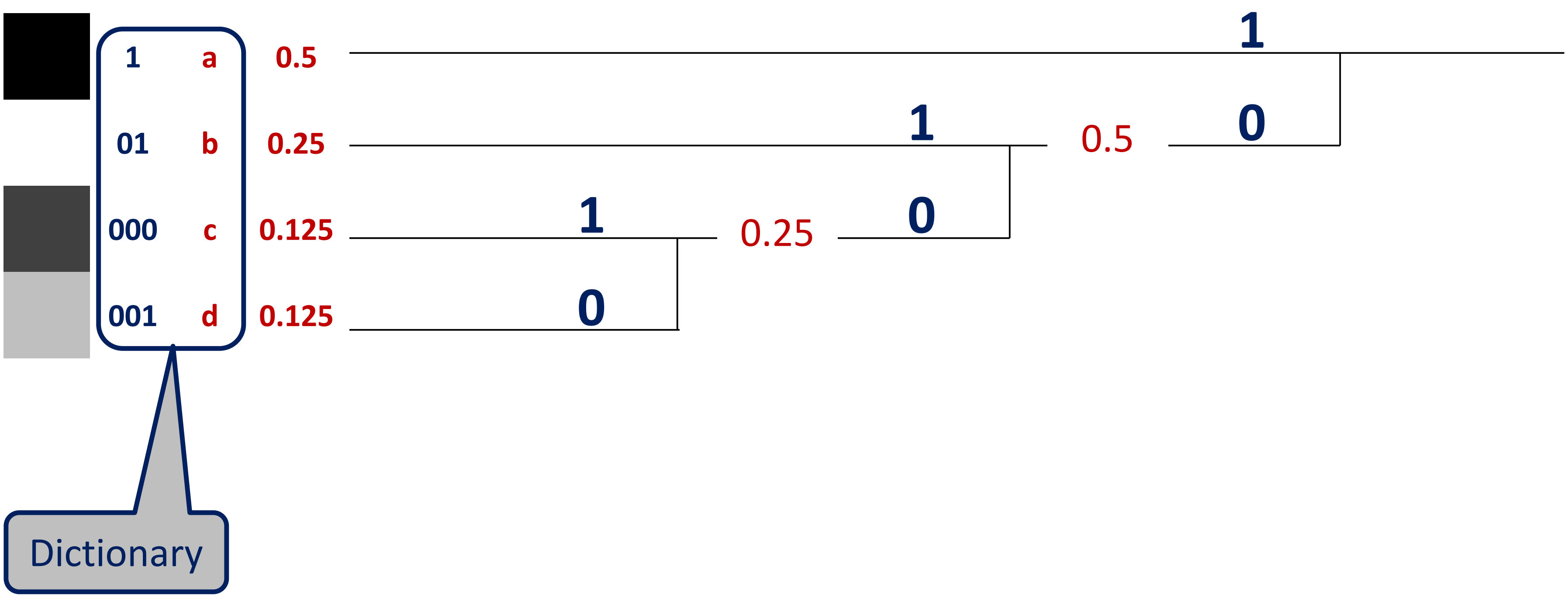
Huffman Coding

- Code assignment



Huffman Coding

- Allows us to construct a VLC



Huffman Coding

- In the previous example the average Length of the VLC code is:

$$0.5\times 1 \quad +0.25\times 2 \quad +0.125\times 3 \quad +0.125\times 3 \quad =\underline{1.75} \quad [bit/symbol]$$

- Compression Ratio: $CR = 2/1.75 \approx \underline{1.14}$

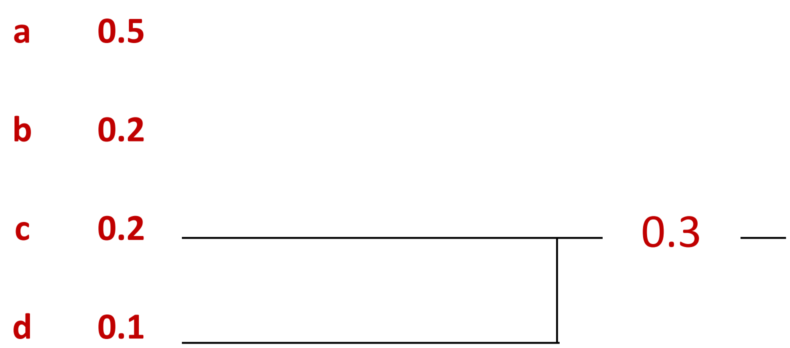
Example II: Huffman Coding

- Building a Huffman tree

a	**0.5**
b	**0.2**
c	**0.2**
d	**0.1**

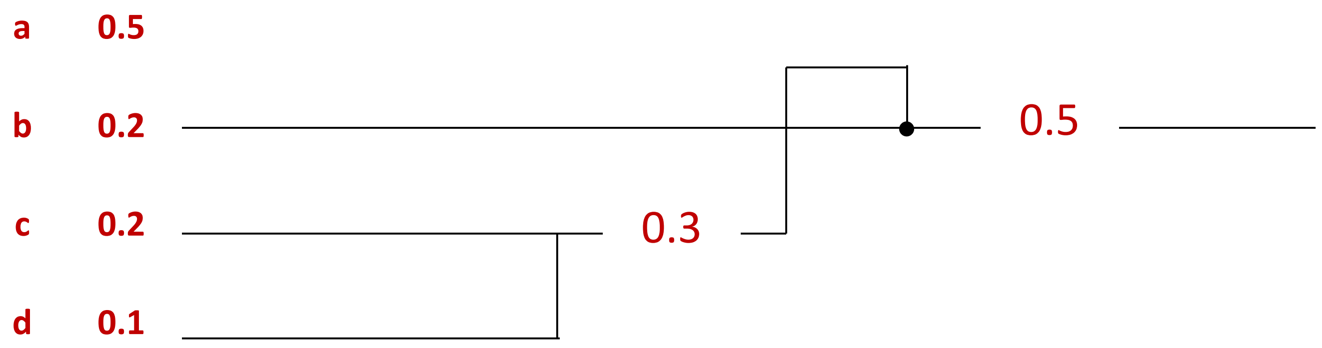
Example II: Huffman Coding

- Building a Huffman tree



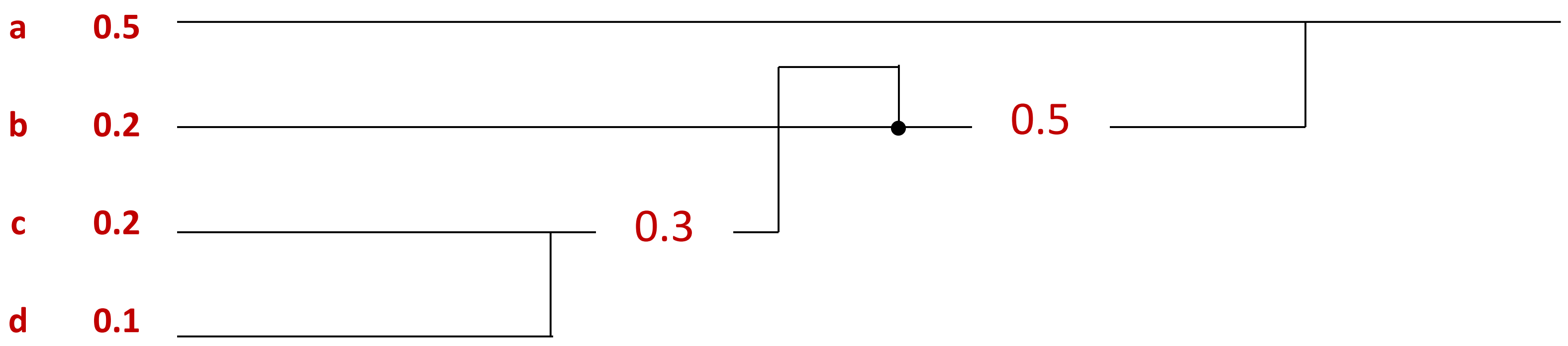
Example II: Huffman Coding

- Building a Huffman tree



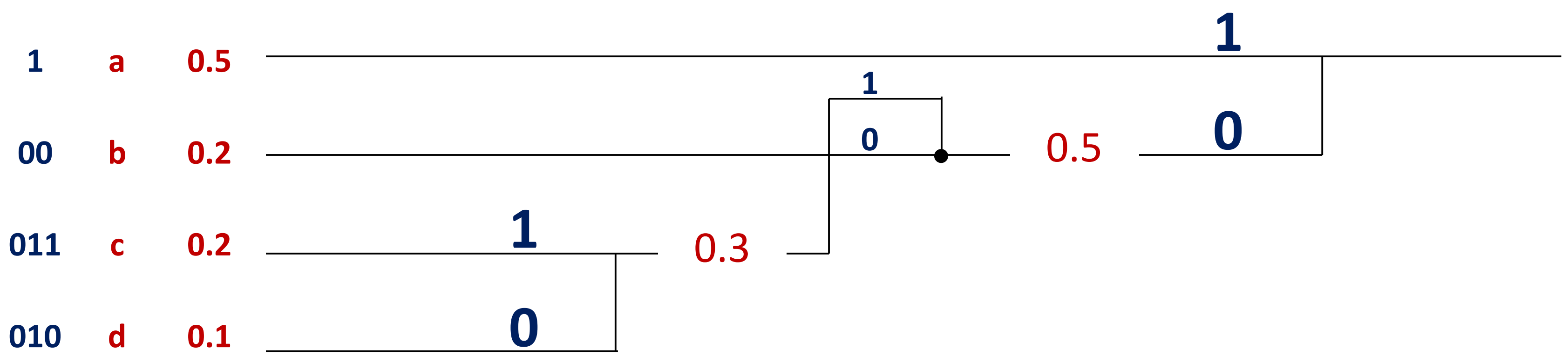
Example II: Huffman Coding

- Building a Huffman tree



Example II: Huffman Coding

- Code assignment

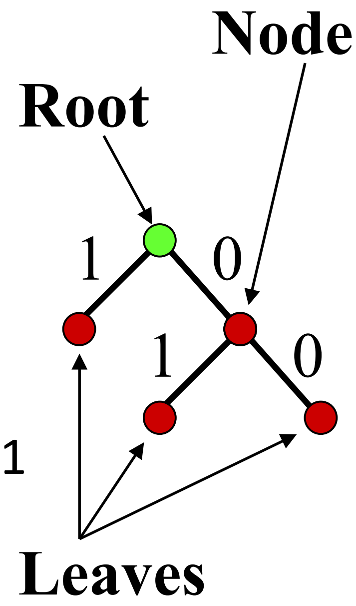


Huffman Code

- Shannon-Fano code [1949]
  - Top-down algorithm: assigning code from most frequent to least frequent
  - VLC, uniquely & instantaneously decodable (no code-word is a prefix of another)
  - Unfortunately not optimal in term of minimum redundancy
- Huffman code [1952]
  - Quite similar to Shannon-Fano in VLC concept
  - Bottom-up algorithm: assigning code from least frequent to most frequent
  - Minimum redundancy when probabilities of occurrence are powers-of-two
  - In JPEG images, DVD movies, MP3 music

Huffman Coding Algorithm

- Encoding algorithm
  - Order the symbols by decreasing probabilities
  - Starting from the bottom, assign **0** to the least probable symbol and **1** to the next least probable
  - Combine the two least probable symbols into one composite symbol
  - Reorder the list with the composite symbol
  - Repeat Step 2 until only two symbols remain in the list
- Huffman tree
  - Nodes: symbols or composite symbols
  - Branches: from each node, 0 defines one branch while 1 defines the other
- Decoding algorithm
  - Start at the root, follow the branches based on the bits received
  - When a leaf is reached, a symbol has just been decoded



Huffman Coding Example

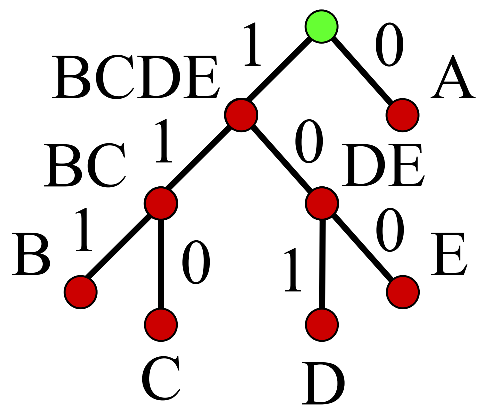
Symbols	Prob.	
A	0.35	
B	0.17	
C	0.17	
D	0.16	1
E	0.15	0

Symbols	Prob.	
A	0.35	
DE	0.31	
B	0.17	1
C	0.17	0

Symbols	Prob.	
A	0.35	
BC	0.34	1
DE	0.31	0

Symbols	Prob.	
BCDE	0.65	1
A	0.35	0

Huffman Tree



Huffman Codes

A	0
B	111
C	110
D	101
E	100

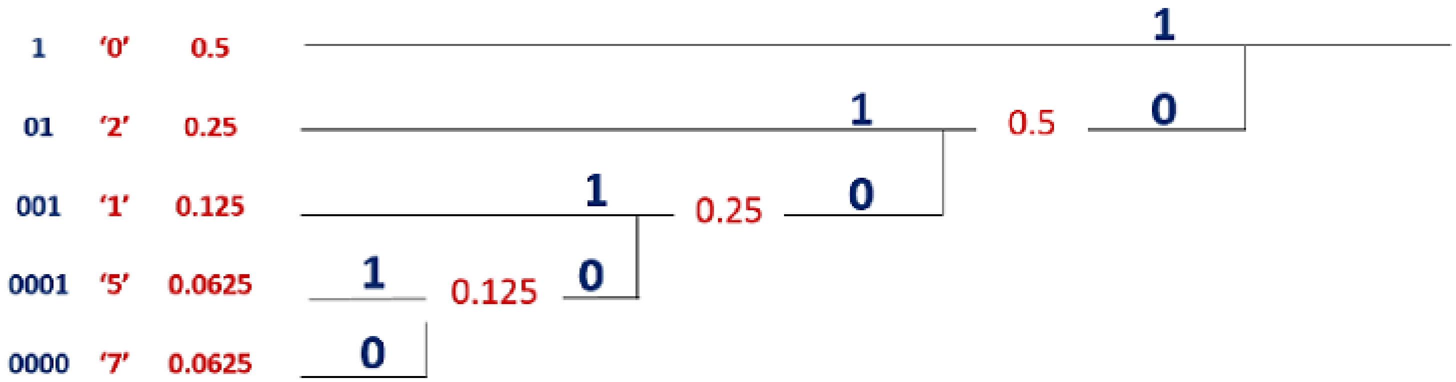
Average code-word length =
0.35 x 1 + 0.65 x 3 = 2.30 bits per symbol

Huffman coding example

- Image $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 7 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 7 & 5 & 5 & 5 & 5 & 2 & 2 \\ 0 & 7 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 7 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

- How to do Huffman coding?

Huffman coding example



Some observations about Huffman Coding

Entropy …

- In the first example of Huffman coding: does the equation of the average Length remind you of something you should know ?

$$
\begin{aligned}
&= 0.5 \times 1 \quad &&+ 0.25 \times 2 \quad &&+ 0.125 \times 3 \quad &&+ 0.125 \times 3 \\
&= 0.5 \times (-\log_2 0.5) \quad &&+ 0.25 \times (-\log_2 0.25) \quad &&+ 0.125 \times (-\log_2 0.125) \quad &&+ 0.125 \times (-\log_2 0.125) \\
&= -\sum_{i=1}^{M} p_i \log_2 p_i
\end{aligned}
$$

Entropy …

- Entropy in Matlab :
  - » h = entropy(uint8(im))

1st order Entropy = 7.4

1st order Entropy = 6.2

White is the most likely value in this picture

Efficiency of Huffman Coding

- Huffman is a code which is optimal when all symbols probabilities are integral power of ½
  - In example I:

$$\begin{aligned}
&\text{average length} && = 0.5\times 1 + 0.25\times 2 + 0.125\times 3 + 0.125\times 3 && = 1.75 \\
&\text{entropy} && = -(0.5\times \log_2 0.5 + 0.25\times \log_2 0.25 + 0.125\times \log_2 0.125 + 0.125\times \log_2 0.125) && = 1.75
\end{aligned}$$

  - In example II:

$$\begin{aligned}
&\text{average length} && = 0.5\times 1 + 0.20\times 2 + 0.200\times 3 + 0.100\times 3 && = 1.80 \\
&\text{entropy} && = -(0.5\times \log_2 0.5 + 0.2\times \log_2 0.2 + 0.2\times \log_2 0.2 + 0.1\times \log_2 0.1) && = 1.76
\end{aligned}$$

- What if the probabilities are not integral power of ½ ?

Bounds on lossless coding

- The average codeword length of a lossless coder cannot be less than entropy
- Entropy represents the target average number of bits/symbol of a lossless encoder
- *Coding Efficiency:*

$$\eta = H(X) / n$$

where n is the average codeword length

Context adaptive coding

- Example of context adaptive coding:
  - The **Run-Length Coding** reduces the length of a repeating character sequence

    a c d e e e e e e e e u h r r r r g e

    a c d #e8 u h #r4 g e

  - The **Dictionary Coding** compress data by searching for repeating sequence of characters

    \- He quietly quit the theatre -

Self-Reading

Arithmetic coding

- Frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total.

- Arithmetic coding encodes the entire message into a **single number.**

- Example:
  - ABBCAB
  - Equal probabilities
  - Fix-length： A=00, B=01 and C=10, 000101100001, ->12 bits
  - Arithmetic : ABBCAB=0.011201_3->0.0010110010_2 , -> 10 bits

Arithmetic coding

- Encode
  - set Low to 0
  - set High to 1
  - while there are input symbols do
  - take a symbol
  - CodeRange = High – Low
  - High = Low + CodeRange *HighRange(symbol)
  - Low = Low + CodeRange * LowRange(symbol)
  - end of while
  - output Low

- Example:
  - C A D A C D B
  - 0.5143876

A	B	C	D
0.1	0.4	0.2	0.3
[0, 0.1)	[0.1, 0.5)	[0.5, 0.7)	[0.7, 1]

1	C	[0.5, 0.7]
2	A	[0.5, 0.52]
3	D	[0.514, 0.52]
4	A	[0.514, 0.5146]
5	C	[0.5143, 0.51442]
6	D	[0.514384, 0.51442]
7	B	[0.5143876 0.514402]

Arithmetic coding

- Decode
  - get encoded number
  - do
  - find symbol whose range straddles the encoded number
  - output the symbol
  - range = symbol.HighValue- symbol.LowValue
  - substract symbol.LowValue from encoded number
  - divide encoded number by range
  - until no more symbols
- Example
  - Input: 0.5143876
  - Output: C A D A C D B

A	B	C	D
0.1	0.4	0.2	0.3
[0, 0.1)	[0.1, 0.5)	[0.5, 0.7)	[0.7, 1]

Index	encodedNumber	Range	Symbol	LowValue	HighValue
1	0.5143877	[0.5,0.7)	C	0.5	0.7
2	0.0719385	[0.0.1)	A	0	0.1
3	0.719385	[0.7,1]	D	0.7	1
4	0.064616667	[0.0.1)	A	0	0.1
5	0.646166667	[0.5,0.7)	C	0.5	0.7
6	0.730833333	[0.7,1]	D	0.7	1
7	0.102777778		B	0.1	0.5

End of Self-Reading

Types of redundancy

- Typical signals contain redundancy which could be exploited:
  - Coding (data representation) redundancy
  - Interpixel redundancy (Correlation between adjacent samples)
  - Psychovisual redundancy.

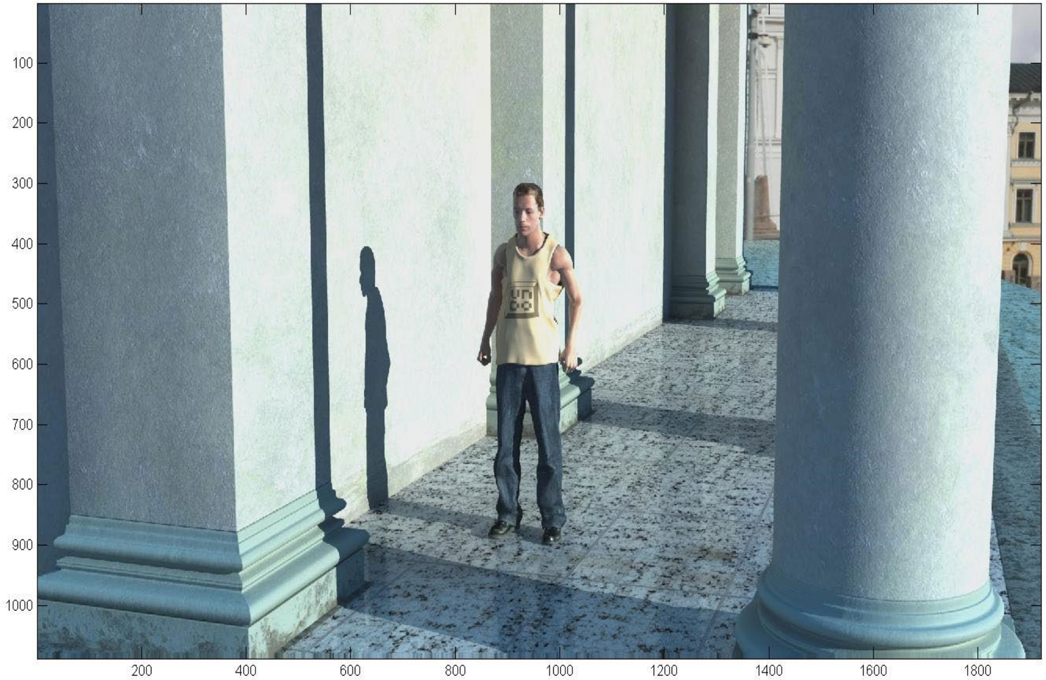
How could we exploit the context-data for image compression ?

How could we exploit the context-data for image compression ?

- In natural images do we have constant patterns with exactly the same intensities?

How could we exploit the context-data for image compression ?

- Which one is natural image and which one is synthetic (graphical image) ?



How could we recognize that an image is natural ?

- In natural images we do NOT have constant patterns with exactly the same intensities.

- Even with the smoothest surface and omnidirectional light the apparently-smooth areas will have some small intensities differences.
  - This is why run-length and dictionary coding do not perform well for natural image compression.

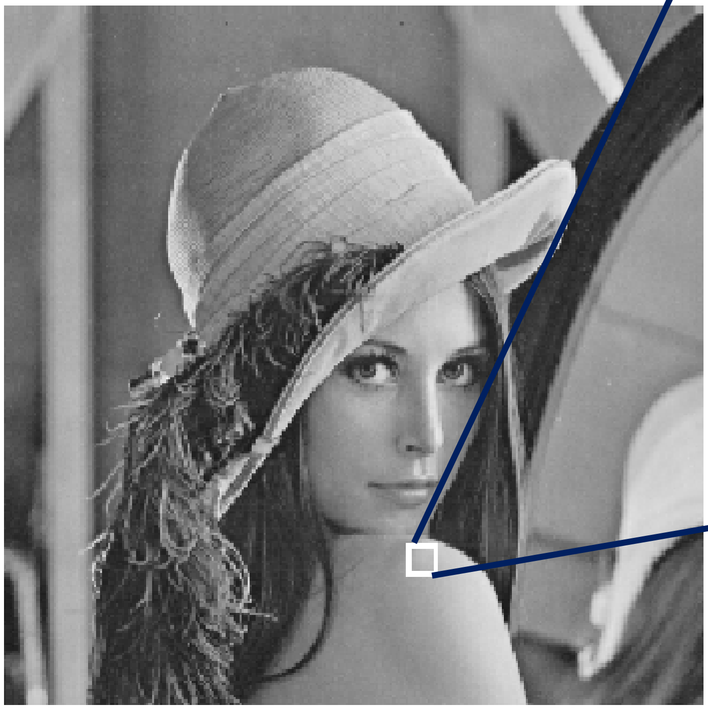
Smoothness of images and interpixel redundancy

Interpixel redundancy

- Example

Interpixel redundancy

- Example



Images are smooth

165	172	181	186	190	196	195	201
169	176	184	187	192	193	194	195
169	173	182	187	190	193	189	190
173	177	182	185	191	189	189	188
168	173	179	182	189	187	188	190
169	170	175	180	183	184	185	189
166	169	173	176	181	180	186	184
171	168	167	176	176	180	177	181

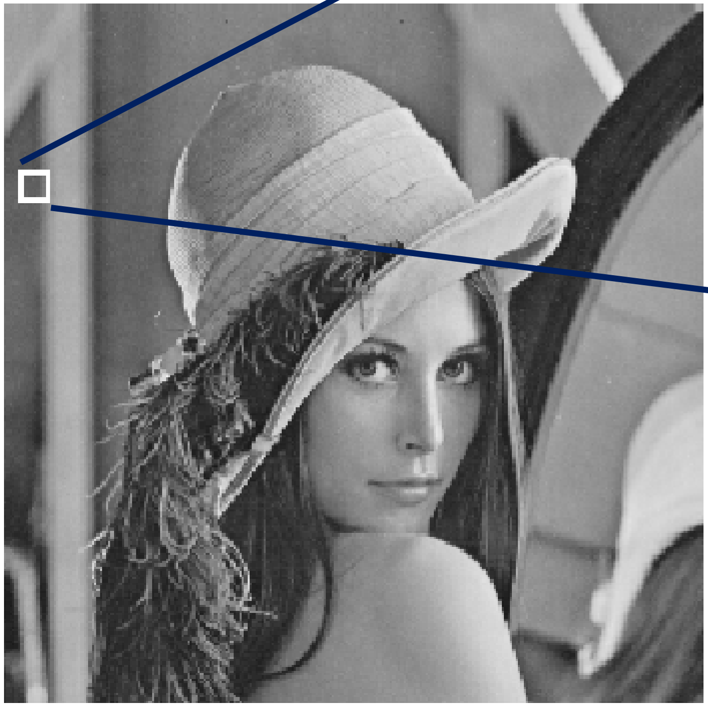
Interpixel redundancy

- Example

Interpixel redundancy

- Example

Images have gradual change of intensity



96	99	99	95	86	95	107	121
97	96	93	107	99	98	109	124
98	96	95	98	98	95	109	118
98	99	98	97	96	95	108	125
98	97	97	95	97	97	106	121
99	100	97	99	95	101	111	125
98	104	98	99	97	99	110	125
103	103	95	96	92	107	111	128

Interpixel redundancy

- If a pixel value can be reasonably predicted from its neighboring pixels the image is said to contain interpixel redundancy.
- Interpixel redundancy depends on the resolution of the image.
  - The higher the spatial resolution of an image, the more probable it is that two neighboring pixels will belong to the same object.

Interpixel redundancy

- Large areas of the image are uniform → **<u>images are smooth</u>** with **<u>gradual change of intensity</u>**
- This means adjacent pixels are almost the same → high correlation among pixels
- This is not exploited using variable length coding (which works on each single pixel)

Interpixel redundancy

- How to exploit the smoothness property of images for compression ?

Predictive coding

Predictive coding

- Given that neighboring pixels in natural images are correlated, so we could exploit some prediction approaches to reduce the dynamics of the data.

Predictive coding

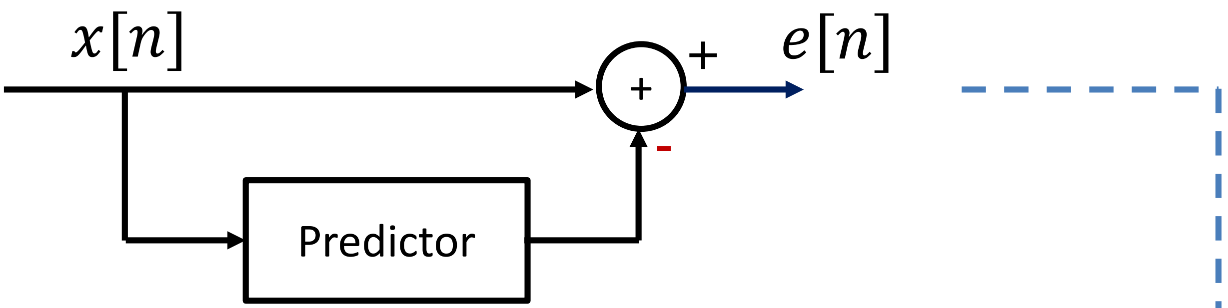
- Provides a data representation of the information where code words express the source symbol deviations from predicted values (usually values of neighboring pixels).
- Predictive coding efficiently reduces interpixel redundancies.
  - 1D & 2D – pixels are predicted from neighboring pixels.
  - Video – pixels are predicted between frames as well.

Predictive coding

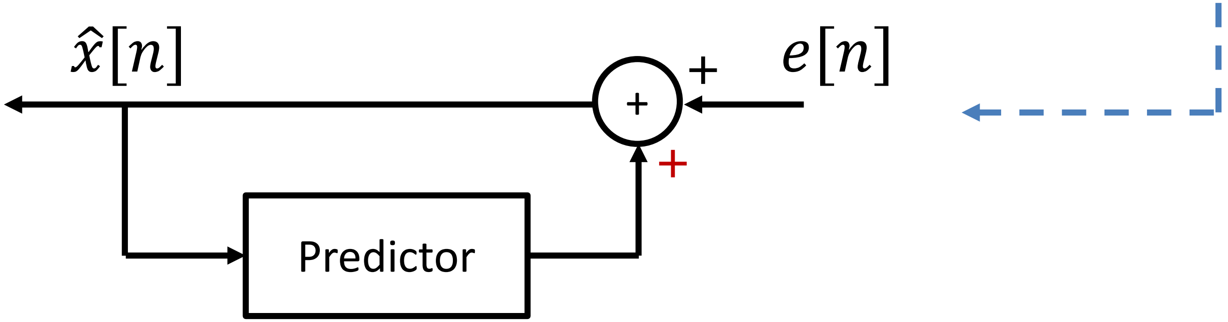
- Works well for all images with a high degree of interpixel redundancies.
- Works in the presence of noise (just not as efficiently).
- Predictive coding can be used in both lossless and lossy compression schemes.

Predictive coding

- Encoder (transmitter – Alice)

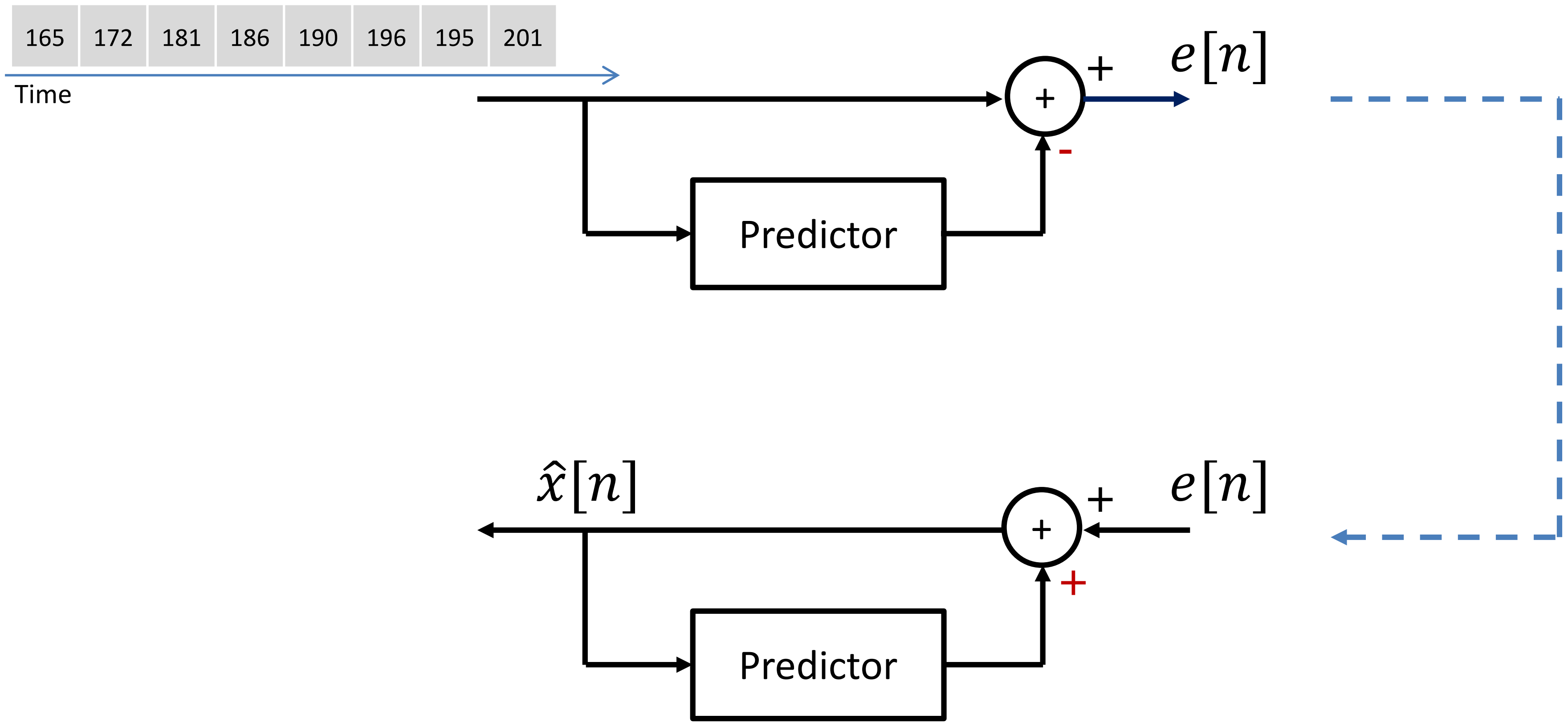


- Decoder (receiver – Bob)



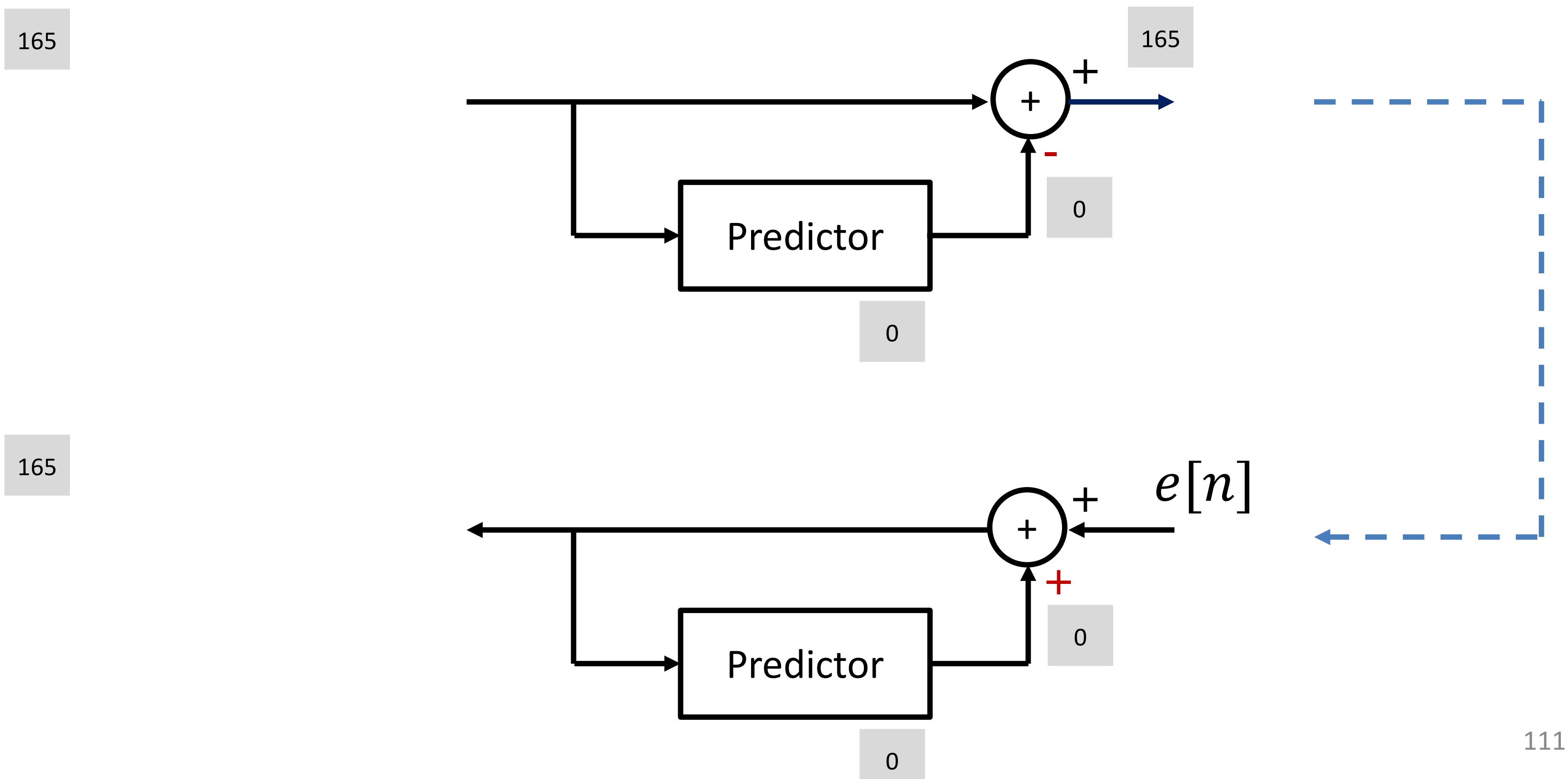
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



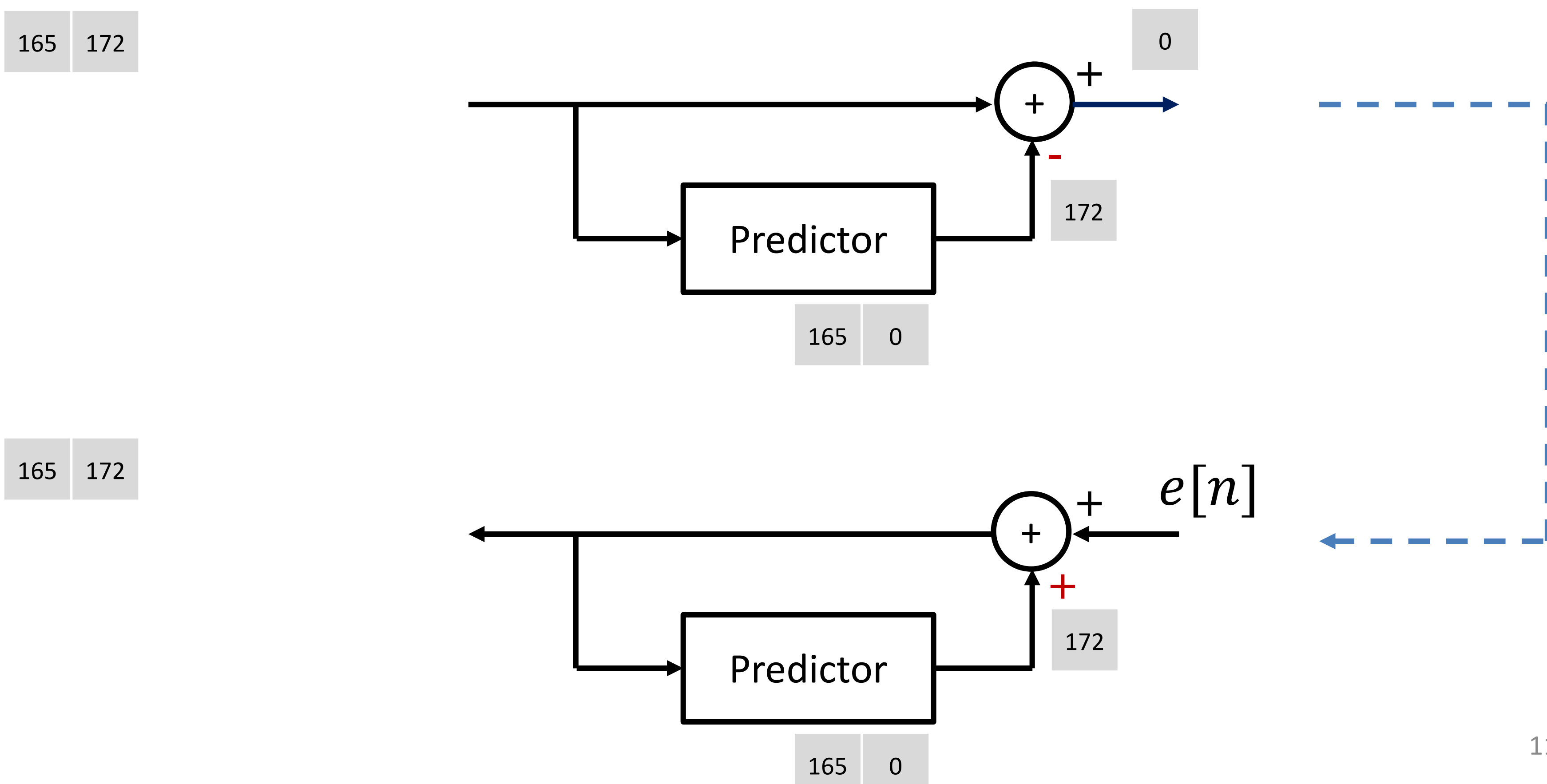
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



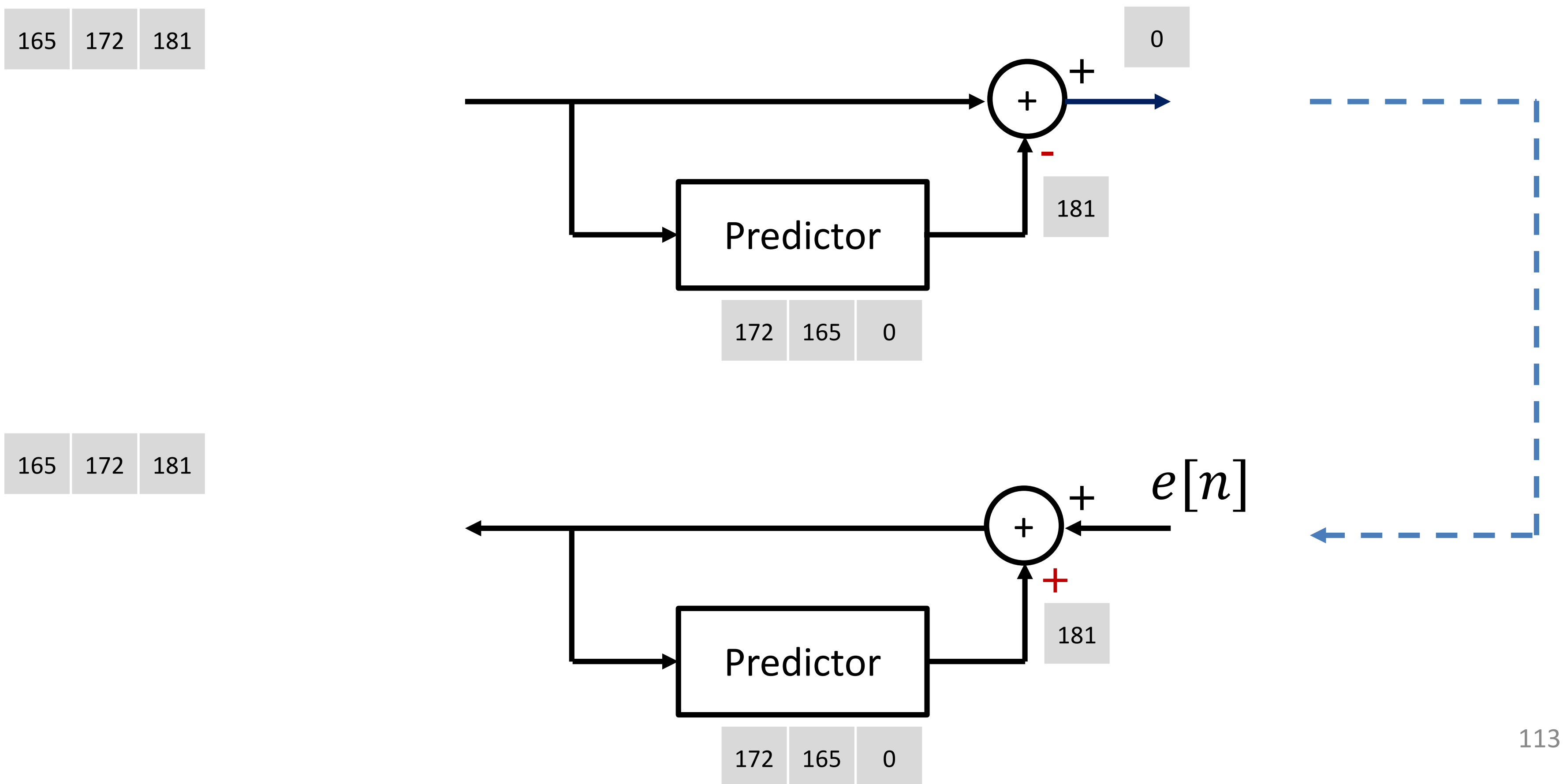
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



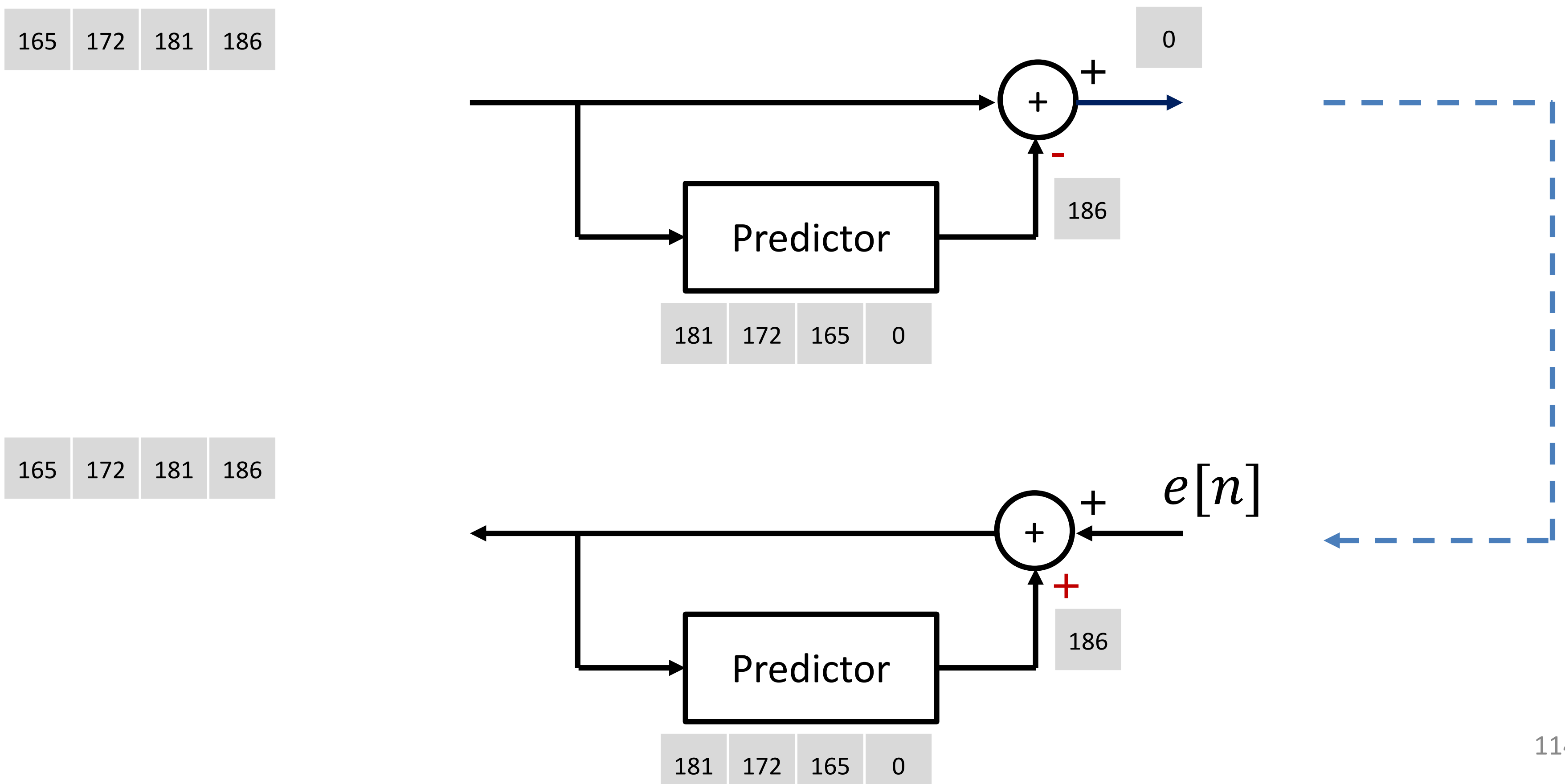
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



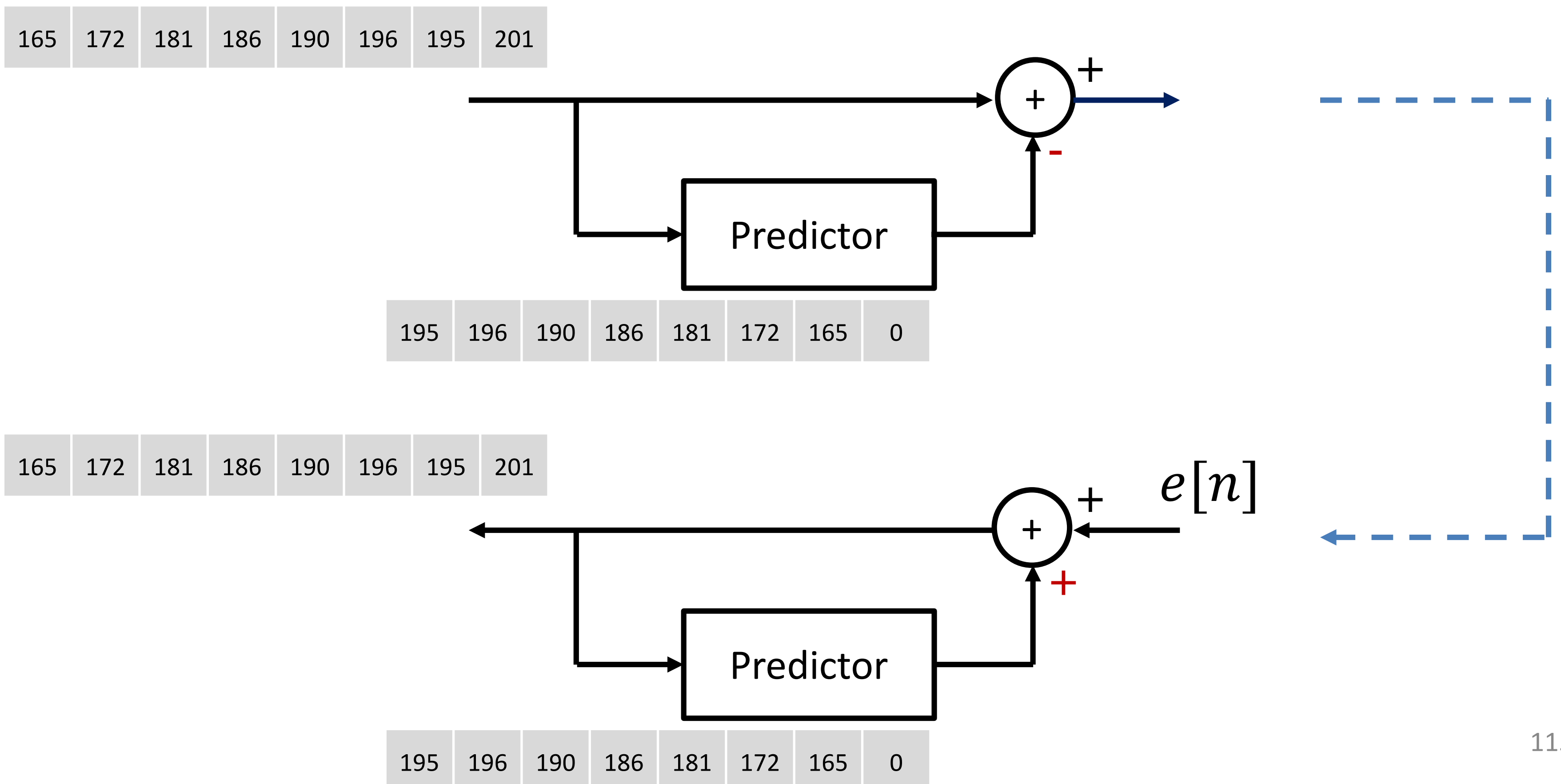
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



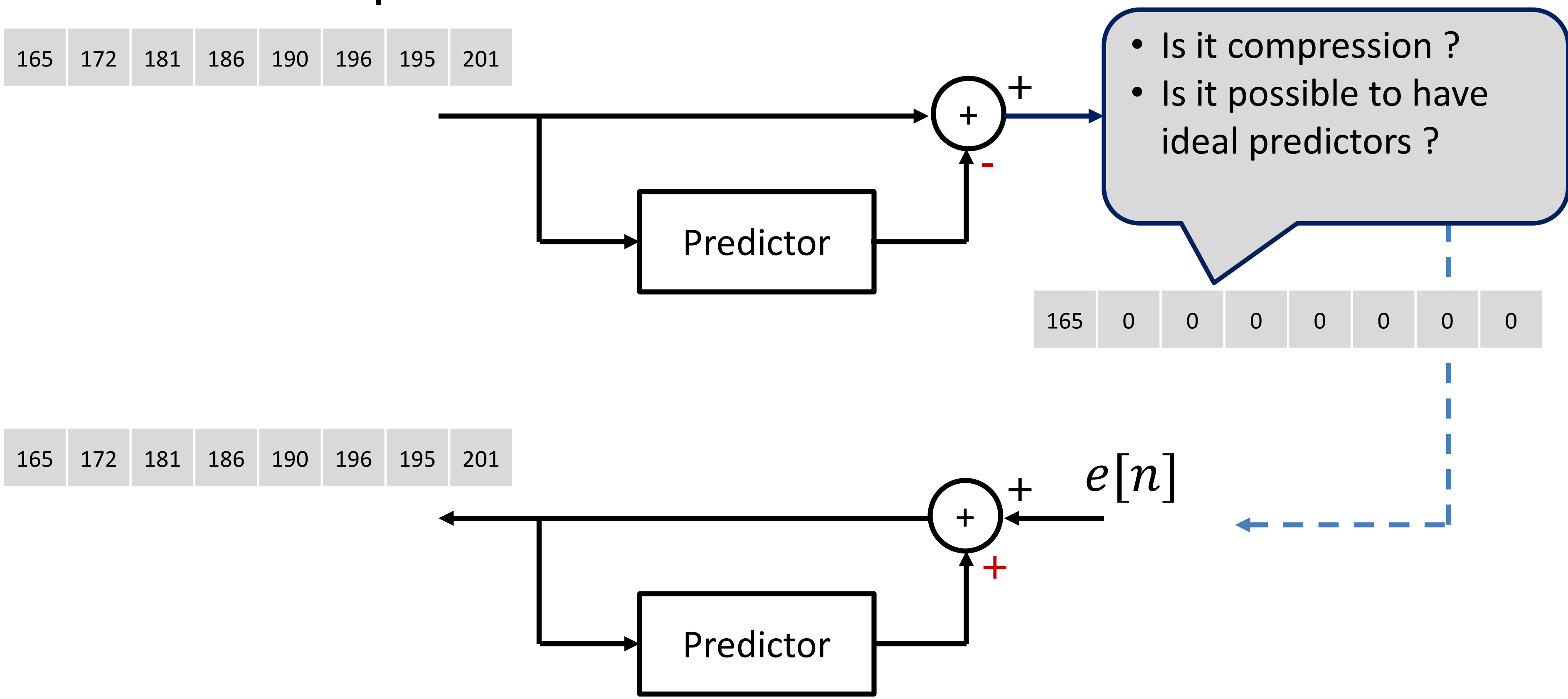
Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones



Predictive coding

- Ideal predictor, which exactly predict current value based on previous ones

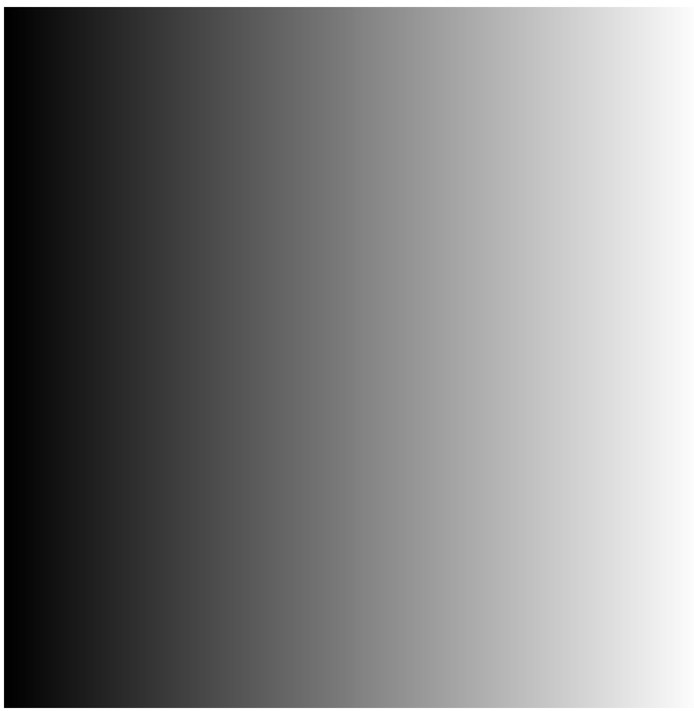


Predictive coding for images

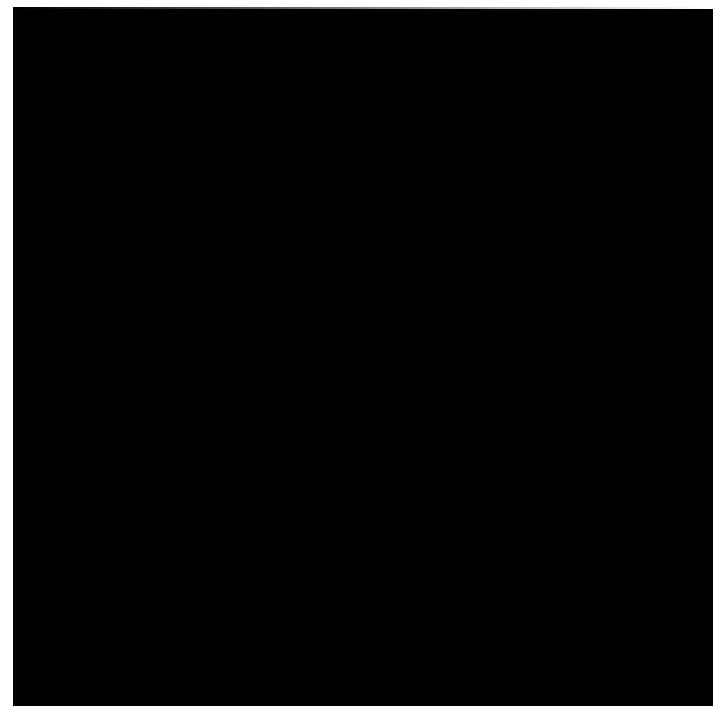
- linear prediction (running on rows)

$$e(r,c) \quad = x(r,c) - x(r,c-1)$$

Original image entropy = 8.00



DPCM image entropy = 0.10



code

Predictive coding for images

- Raster-scan DPCM coding (differential pulse code modulation)

165	172	181	186	190	196	195	201
169	176	184	187	192	193	194	195
169	173	182	187	190	193	189	190
173	177	182	185	191	189	189	188
168	173	179	182	189	187	188	190
169	170	175	180	183	184	185	189
166	169	173	176	181	180	186	184
171	168	167	176	176	180	177	181

$$\begin{aligned} p(r,c) &= f\big(x(r,c-1), x(r-1,c-1), x(r-1,c)\big) \\ e(r,c) &= x(r,c) - p(r,c) \end{aligned}$$

Predictive coding for images

- Raster-scan DPCM coding

165	172	181	186	190	196	195	201
169	176	184	187	192	193	194	195
169	173	182	187	190	193	189	190
173	177	182	185	191	189	189	188
168	173	179	182	189	187	188	190
169	170	175	180	183	184	185	189
166	169	173	176	181	180	186	184
171	168	167	176	176	180	177	181

$$
\begin{aligned}
p(r,c) &= f\big(x(r,c-1), x(r-1,c-1), x(r-1,c)\big) \\
e(r,c) &= x(r,c) - p(r,c)
\end{aligned}
$$

Predictive coding for images

- Raster-scan DPCM coding

$$e(r,c) \quad = x(r,c) - \frac{x(r,c-1) + x(r-1,c-1) + x(r-1,c)}{3}$$

Original

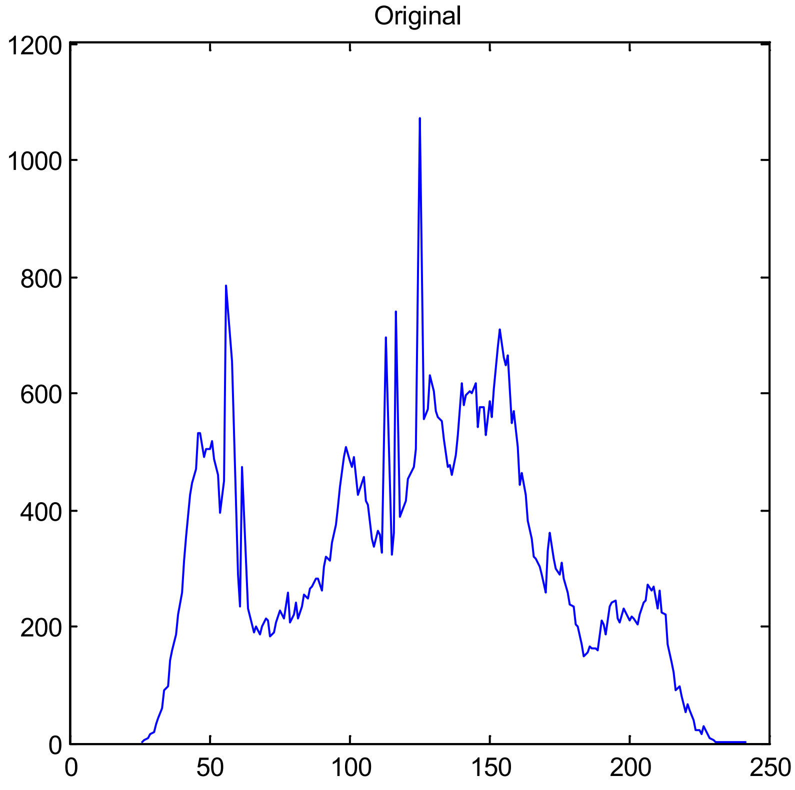
DPCM output

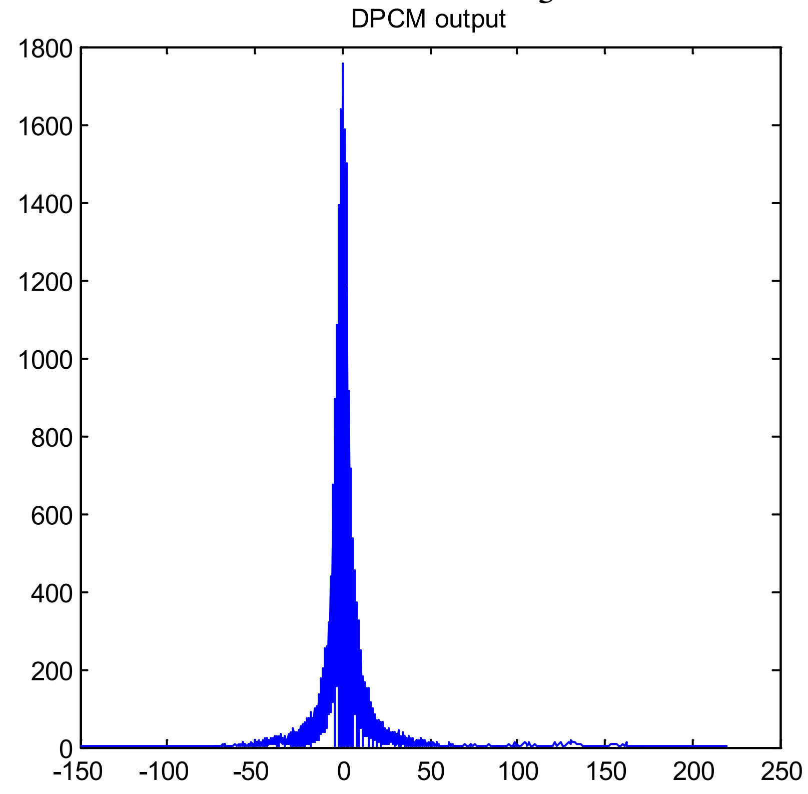
code

Predictive coding for images

- Raster-scan DPCM coding

$$e(r,c) \quad = x(r,c) - \frac{x(r,c-1) + x(r-1,c-1) + x(r-1,c)}{3}$$





code

Psychovisual redundancy

Psychovisual redundancy

- Human perception of the information of an image does not involve quantitative analysis of every pixel
- Pixel values can be modified up to a given extent without significant subjective degradation
- Proper alteration should involve psychovisually redundant information
  - The image is irreversibly altered

Psychovisual redundancy

- Psychovisual redundancy stem from that the human eye does not respond equally to all visual information.
- An observer searches for distinct features and mentally combines them into recognizable objects.
  - Use of prior knowledge for interpretation (face, wall, …)
- In this process certain information is relatively less important than other — this information is called psychovisually redudant.

- If the wall were slightly different this could not be perceived

LOSSY DATA COMPRESSION
(Non reversible algorithms)

Introduction

- Advantages of lossy compression:
  - Higher compression ratios

- Disadvantages:
  - The decoded signal is not exactly conform to the original

Examples of lossy compression

- Speech/Audio:
  - GSM/UMTS/WCDMA (multi-rate) speech compression
  - MP3 audio
- Image compression:
  - JPEG
  - JPEG2000
- Video compression
  - H.264/AVC , H.264/SVC
  - H.265 (HEVC)

Lossy compression

- Types of lossy compression
  - Predictive coding
  - Transform coding
  - Model-based coding
  - …

- Lossy quality measures:
  - PSNR
  - SNR
  - Subjective measures

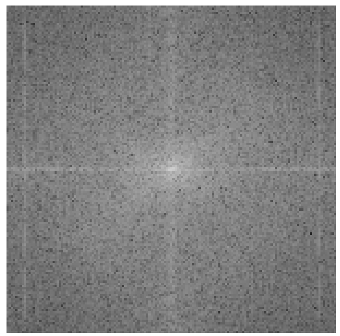
transform coding

General transform coding scheme

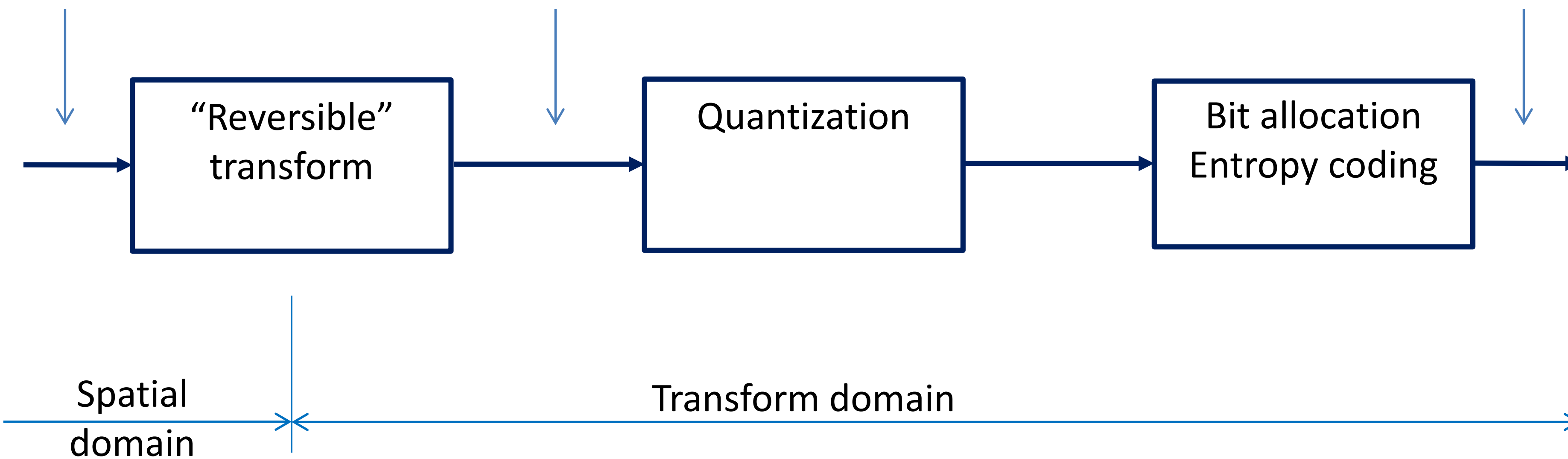
The transformation aims to reduce the inter-pixel correlation

- KLT (Karhunen-Loeve)
- DCT (Discrete cosine)
- DWT (Discrete wavelet)

Original



165	172	181	186	190	196	195	201



Example of 2-D DCT

Image

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	160	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	162	163	158	158	158

DCT

236	-1	-12	-5	2	-2	-3	1
-23	-17	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	1	1	0	0	0
-1	-1	1	2	0	-1	1	-1
2	0	2	0	-1	1	1	-1
-1	0	0	-1	0	2	1	-1
-3	2	-4	-2	2	1	-1	0

2-D DCT

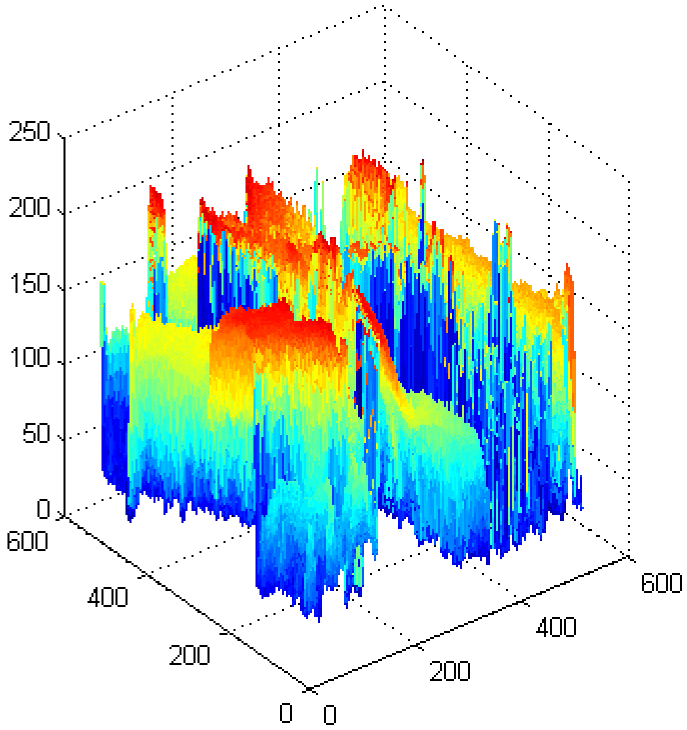
- The two-dimensional DCT (2D-DCT) could be written as :

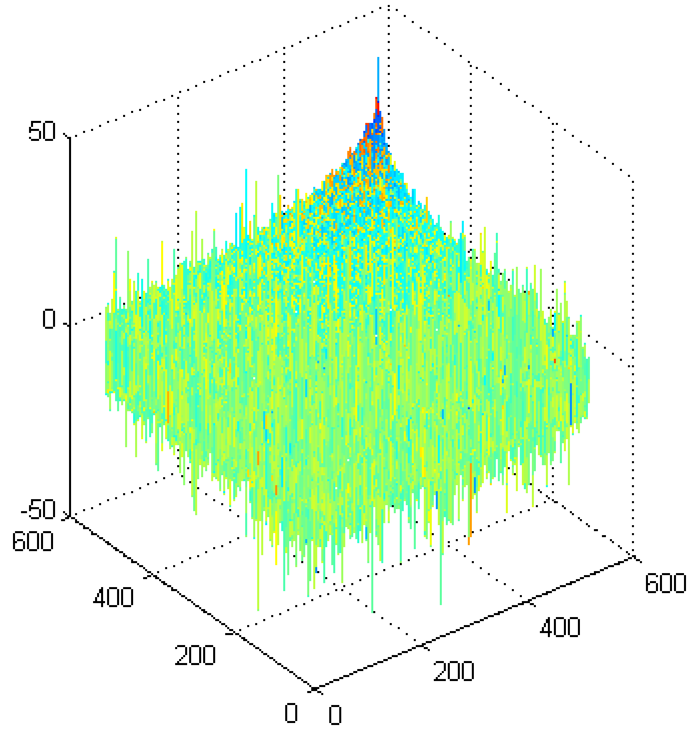
$$F(k,l) = \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2n+1)k\pi}{2N}\right]\cos\left[\frac{(2m+1)l\pi}{2N}\right]$$

$$k,l = 0,\ldots,N-1$$

$$\alpha(0) = \sqrt{\frac{1}{N}} \quad \alpha(k) = \sqrt{\frac{2}{N}}$$

Why the need for transform coding ?





code

Decorrelation property of the DCT

- It can be noticed that the coefficients in the transformed domain are much more unbalanced
- Or equivalently we could say in the transformed domain few coefficients concentrate most of the signal energy → therefore can be more efficiently compressed
- This property holds for most frequency transforms, if the image has low pass characteristics (smooth areas)

Joint Photographic Expert Group
-JPEG-

The JPEG standard

- JPEG stands for “Joint Photographic Expert Group".
- Voted as international standard in 1992
- This standardized image compression scheme is designed to work on full-color or gray-scale
- It specifies the decoder and the codestream syntax

The JPEG standard

- Most image standard specifies the decoder and the codestream syntax

165	172	181	186	190	196	195	201

Standard-compatible decoder

It is not very important how you compress (implement the standard), what matter is to generate a standard-compatible stream

JPEG coding steps

- Transformation of the image into a suitable color space
- Application of a 8x8 blocks DCT
- Quantization
- *zig-zag* reading
- Entropy (lossless) coding

JPEG: 8x8 DCT

- 8x8 DCT transform

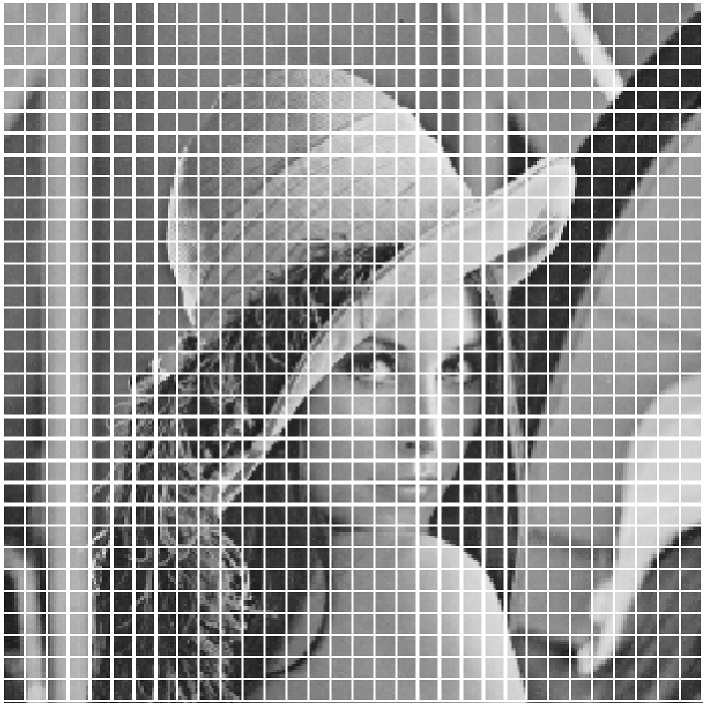
image size is 256 x 256

- The image is processed as 8x8 blocks

JPEG: 8x8 DCT

- 8x8 DCT transform

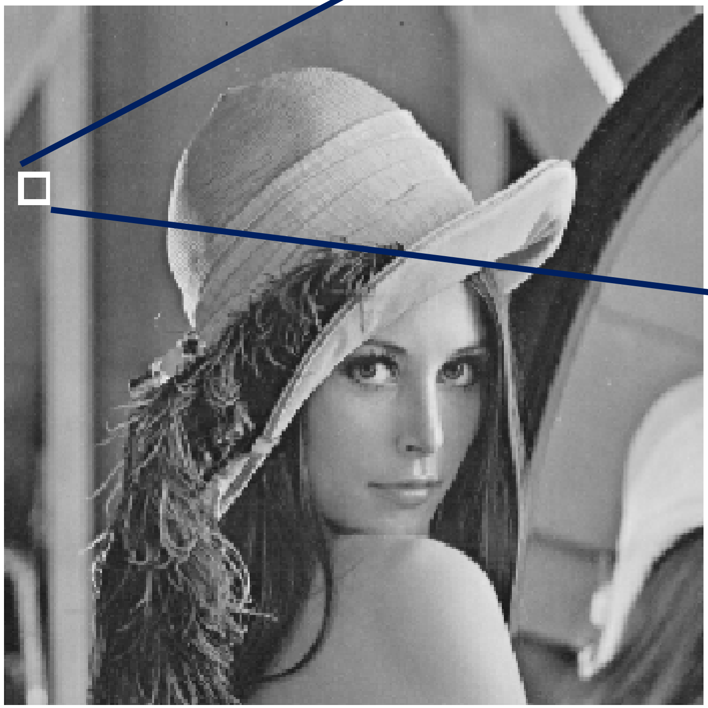
image size is 256 x 256



- Each 8x8 block of pixels is separately DCT transformed

JPEG: 8x8 DCT

- One block pixel

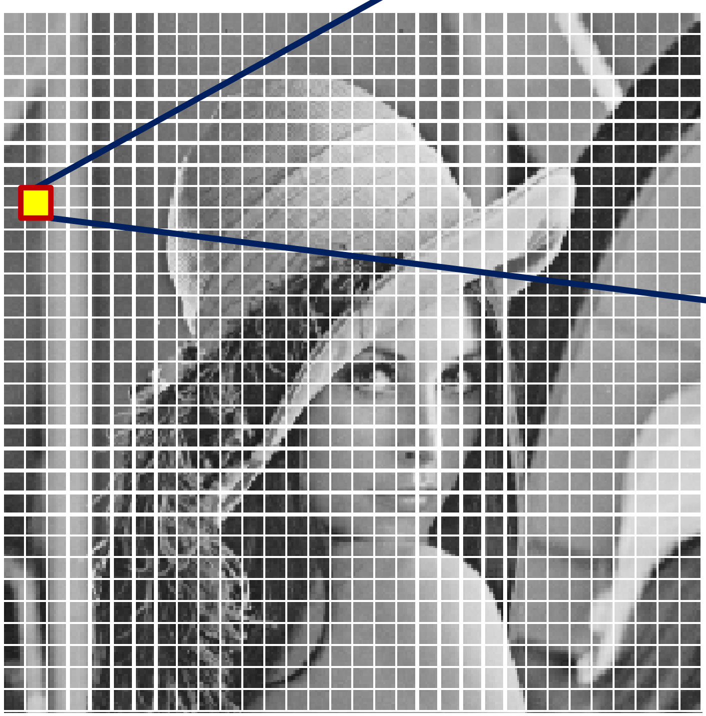


96	99	99	95	86	95	107	121
97	96	93	107	99	98	109	124
98	96	95	98	98	95	109	118
98	99	98	97	96	95	108	125
98	97	97	95	97	97	106	121
99	100	97	99	95	101	111	125
98	104	98	99	97	99	110	125
103	103	95	96	92	107	111	128

code

JPEG: 8x8 DCT

- 8x8 DCT transform



818.0	-46.6	44.4	-26.7	12.0	-3.0	-1.9	-6.0
-9.1	1.7	-5.1	-4.0	3.4	4.5	-2.0	2.2
2.8	-0.7	4.2	-0.3	-3.9	1.3	1.2	-6.7
-2.5	1.2	0.7	-3.7	-3.1	0.2	1.4	-0.2
-3.8	1.7	6.4	-1.0	-4.3	-1.7	4.9	0.2
-1.9	3.0	2.7	-4.5	-3.9	-1.4	-0.9	2.7
-2.5	0.7	3.4	2.3	-4.1	1.8	2.1	0.5
-4.9	3.0	-1.5	1.5	-1.6	-0.8	-0.2	3.4

- The DCT coefficients are real valued, so forward and inverse DCT are limited by computer precision
- No truly lossless compression is possible if the non-integer DCT is used

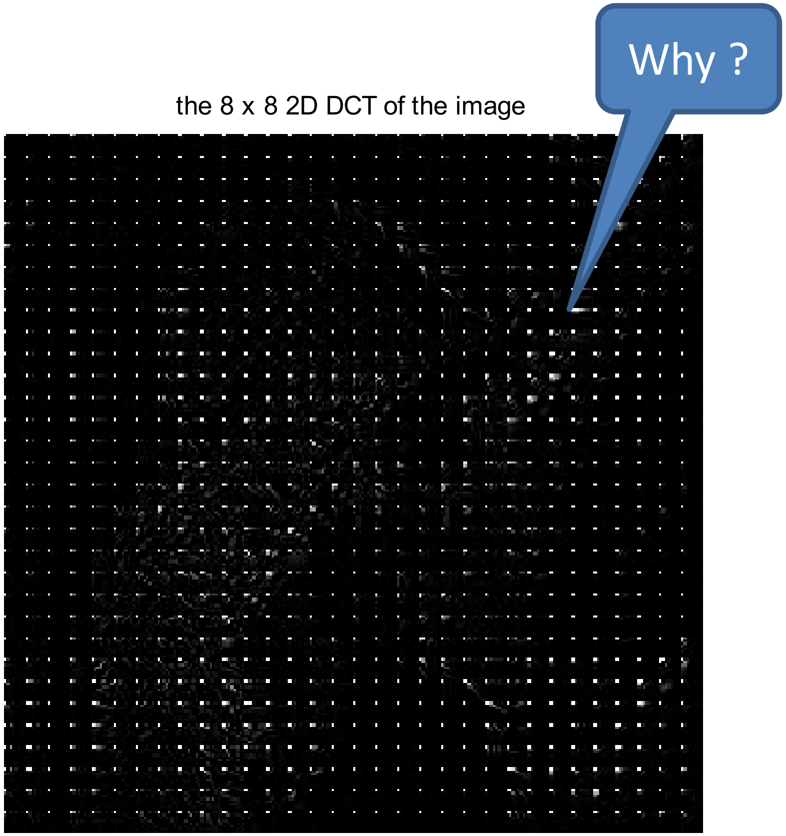
code

JPEG: 8x8 DCT

- Lenna in the 8x8 DCT domain

image size is 256 x 256

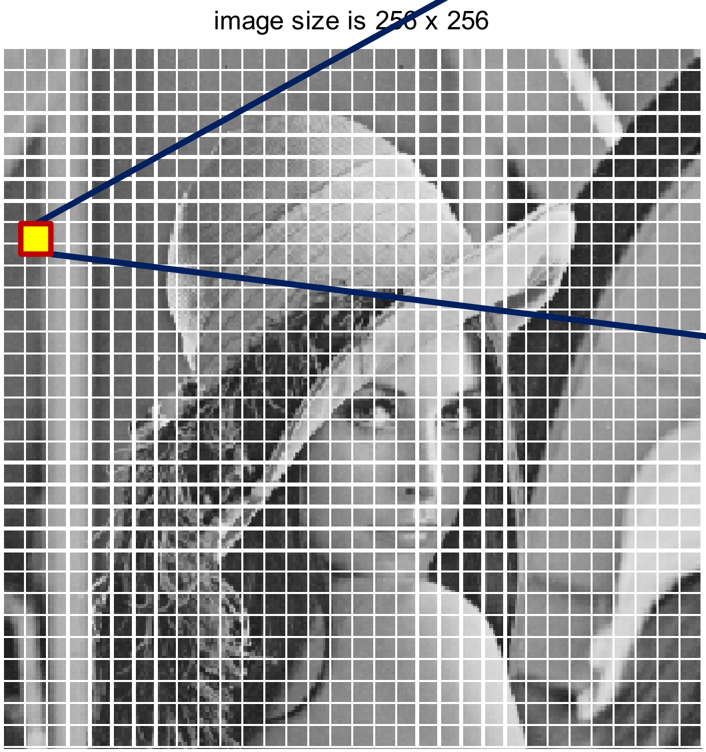
the 8 x 8 2D DCT of the image



code

JPEG: 8x8 DCT

- 8x8 DCT transform



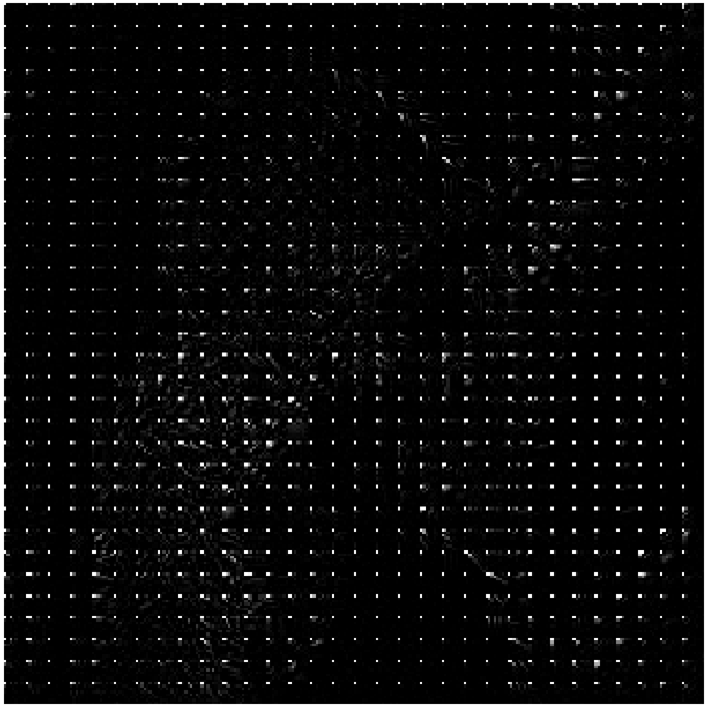
818.0	-46.6	44.4	-26.7	12.0	-3.0	-1.9	-6.0
-9.1	1.7	-5.1	-4.0	3.4	4.5	-2.0	2.2
2.8	-0.7	4.2	-0.3	-3.9	1.3	1.2	-6.7
-2.5	1.2	0.7	-3.7	-3.1	0.2	1.4	-0.2
-3.8	1.7	6.4	-1.0	-4.3	-1.7	4.9	0.2
-1.9	3.0	2.7	-4.5	-3.9	-1.4	-0.9	2.7
-2.5	0.7	3.4	2.3	-4.1	1.8	2.1	0.5
-4.9	3.0	-1.5	1.5	-1.6	-0.8	-0.2	3.4

JPEG: 8x8 DCT

- 8x8 DCT transform

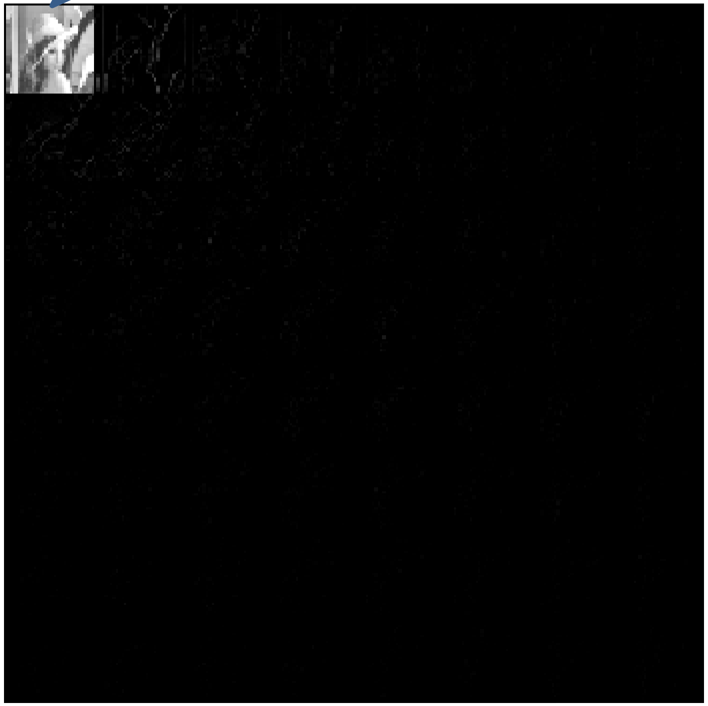
This format is not used in JPEG, it is just illustrative, however, it shows that the DC values are still correlated

the 8 x 8 2D DCT of the image (Natural order)



Natural order

the 8 x 8 2D DCT of the image (Reordered



Reordered (DC coeff. grouped)

code

JPEG: quantization

- This is the truly lossy stage of JPEG
- The *quantization matrix* is an 8 by 8 matrix of step sizes, with one element for each DCT coefficient
- Step sizes will be small for low frequencies, and large for high frequencies.
- The quantizer divides the DCT coefficient by its corresponding quantization step, then rounds to the nearest integer

Quantization example

$$\begin{bmatrix} 235.6 & -1.0 & /12.1 & -5.2 & 2.1 & -1.7 & -2.7 & 1.3 \\ -22.6 & -17.5 & -6.2 & -3.2 & -2.9 & -0.1 & 0.4 & -1.2 \\ -10.9 & -9.3 & -1.6 & 1.5 & 0.2 & -0.9 & -0.6 & -0.1 \\ -7.1 & -1.9 & 0.2 & 1.5 & 0.9 & -0.1 & 0.0 & 0.3 \\ -0.6 & -0.8 & 1.5 & 1.6 & -0.1 & -0.7 & 0.6 & 1.3 \\ 1.8 & -0.2 & 1.6 & -0.3 & -0.8 & 1.5 & 1.0 & -1.0 \\ -1.3 & -0.4 & -0.3 & -1.5 & -0.5 & 1.7 & 1.1 & -0.8 \\ -2.6 & 1.6 & -3.8 & -1.8 & 1.9 & 1.2 & -0.6 & -0.4 \end{bmatrix}$$

Original 8x8 block

Quantization table

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

Quantization example

$$\begin{bmatrix} 15 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

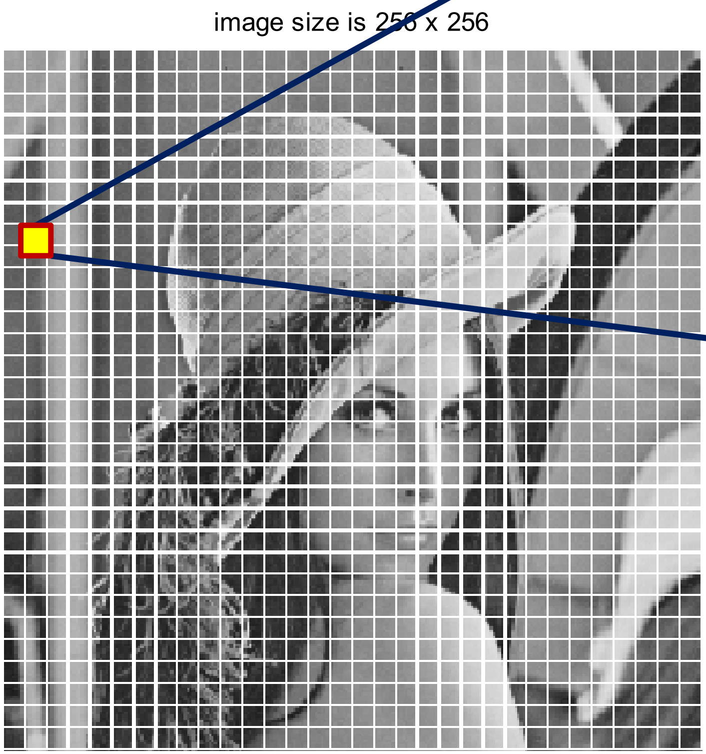
Quantized 8x8 block

Dequantized 8x8 block

$$\begin{bmatrix} 240 & 0 & -10 & 0 & 0 & 0 & 0 & 0 \\ -24 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

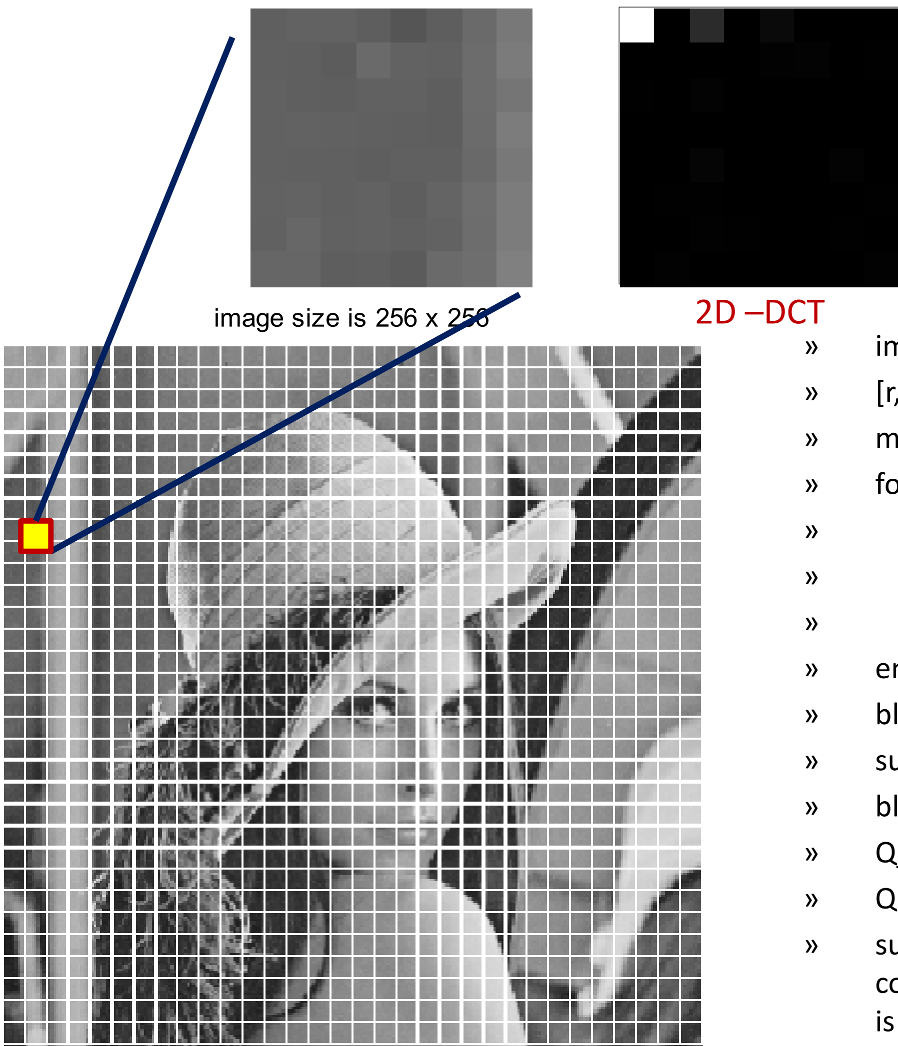
JPEG: quantization

- 8x8 DCT transform



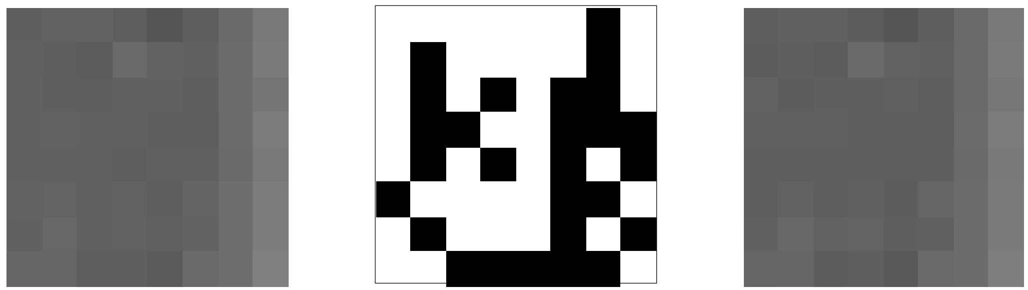
818.0	-46.6	44.4	-26.7	12.0	-3.0	-1.9	-6.0
-9.1	1.7	-5.1	-4.0	3.4	4.5	-2.0	2.2
2.8	-0.7	4.2	-0.3	-3.9	1.3	1.2	-6.7
-2.5	1.2	0.7	-3.7	-3.1	0.2	1.4	-0.2
-3.8	1.7	6.4	-1.0	-4.3	-1.7	4.9	0.2
-1.9	3.0	2.7	-4.5	-3.9	-1.4	-0.9	2.7
-2.5	0.7	3.4	2.3	-4.1	1.8	2.1	0.5
-4.9	3.0	-1.5	1.5	-1.6	-0.8	-0.2	3.4

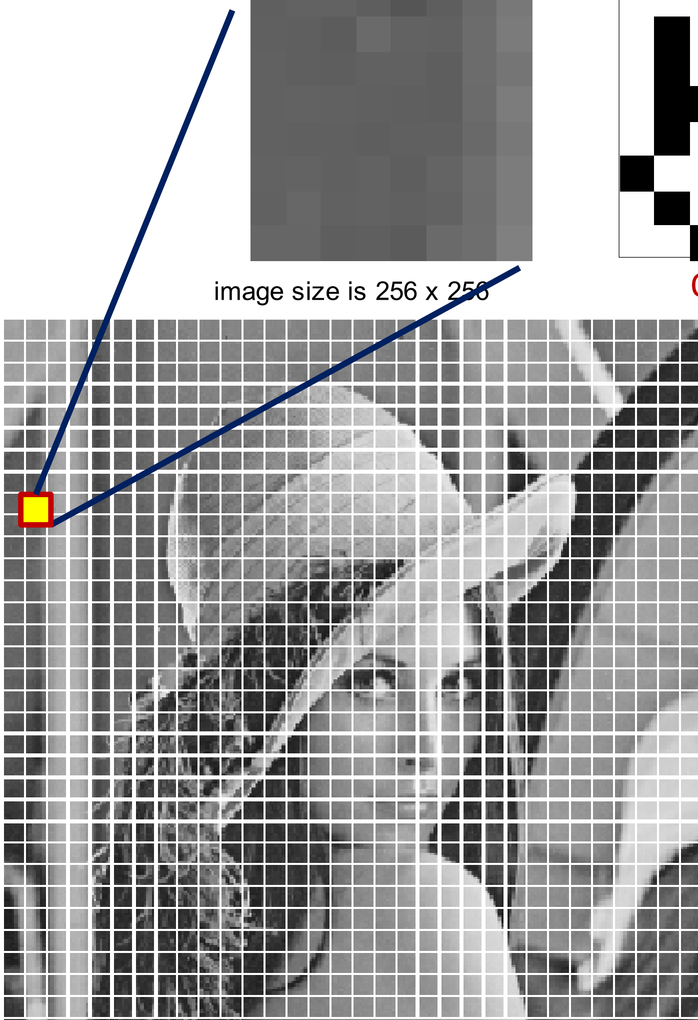
JPEG: quantization




```
» im = double(imread('lenna256.bmp'));
» [r,c] = size(im);
» mask = zeros(r,c);
» for ri = 64 + [1 : 8] -1
»    for ci = 8+[1:8]-1
»       mask(ri,ci) = 1;
»    end;
» end;
» blk = reshape(im(mask == 1), 8,8);
» subplot(1,3,1); image(blk); colormap(gray(255)); axis off;
» blkdct = dct2(blk);
» Q_step = 16;
» Qblkdct = round(blkdct/Q_step);
» subplot(1,3,2); image(255*(Qblkdct ~= 0)); colormap(gray(255)); axis off; title(sprintf('Quantization step is %d', Q_step));
» iblkdct = idct2(Q_step * Qblkdct);
» subplot(1,3,3); image(iblkdct); colormap(gray(255)); axis off;
» truesize([140 140])
```

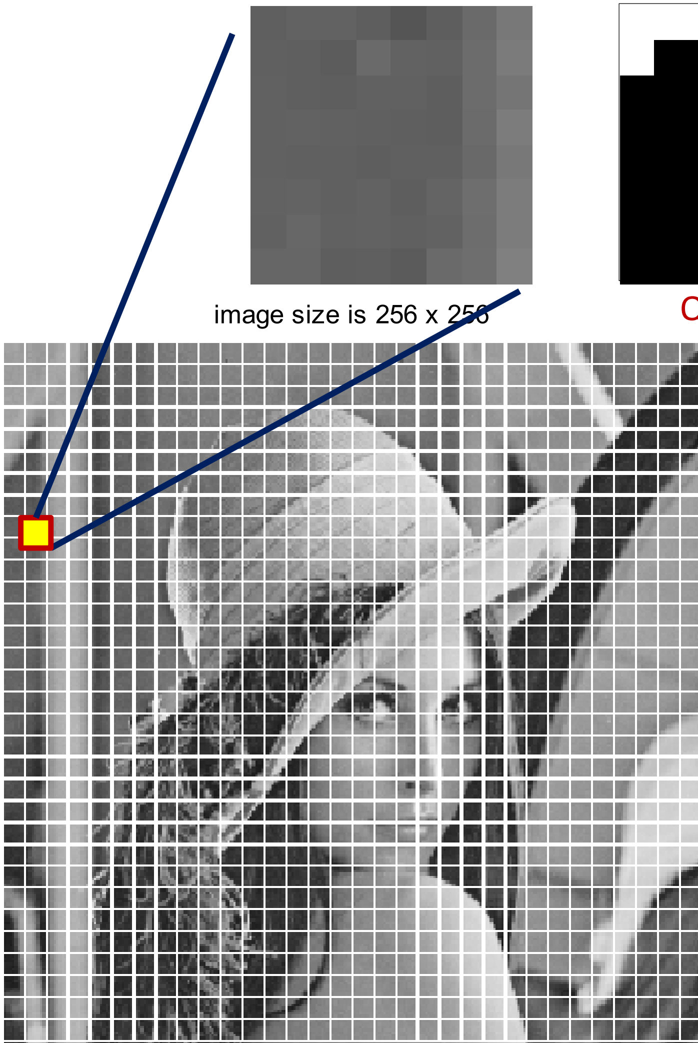
JPEG: quantization

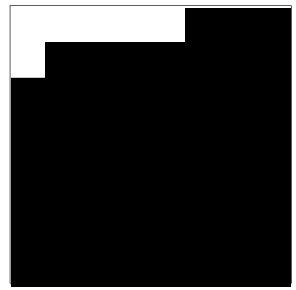





```
»   im = double(imread('lenna256.bmp'));
»   [r,c] = size(im);
»   mask = zeros(r,c);
»   for ri = 64 + [1 : 8] -1
»      for ci = 8+[1:8]-1
»         mask(ri,ci) = 1;
»      end;
»   end;
»   blk = reshape(im(mask == 1), 8,8);
»   subplot(1,3,1); image(blk); colormap(gray(255)); axis off;
»   blkdct = dct2(blk);
»   Q_step = 4;
»   Qblkdct = round(blkdct/Q_step);
»   subplot(1,3,2); image(255*(Qblkdct ~= 0)); colormap(gray(255)); axis off; title(sprintf('Quantization step is %d', Q_step));
»   iblkdct = idct2(Q_step * Qblkdct);
»   subplot(1,3,3); image(iblkdct); colormap(gray(255)); axis off;
»   truesize([140 140])
```

JPEG: quantization





Q-step=16

De-quntized

```
»   im = double(imread('lenna256.bmp'));
»   [r,c] = size(im);
»   mask = zeros(r,c);
»   for ri = 64 + [1 : 8] -1
»      for ci = 8+[1:8]-1
»         mask(ri,ci) = 1;
»      end;
»   end;
»   blk = reshape(im(mask == 1), 8,8);
»   subplot(1,3,1); image(blk); colormap(gray(255)); axis off;
»   blkdct = dct2(blk);
»   Q_step = 16;
»   Qblkdct = round(blkdct/Q_step);
»   subplot(1,3,2); image(255*(Qblkdct ~= 0));
    colormap(gray(255)); axis off; title(sprintf('Quantization step
    is %d', Q_step));
»   iblkdct = idct2(Q_step * Qblkdct);
»   subplot(1,3,3); image(iblkdct); colormap(gray(255)); axis off;
»   truesize([140 140])
```

JPEG: quantization

- JPEG quantization matrix (table)

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

JPEG: quantization effects

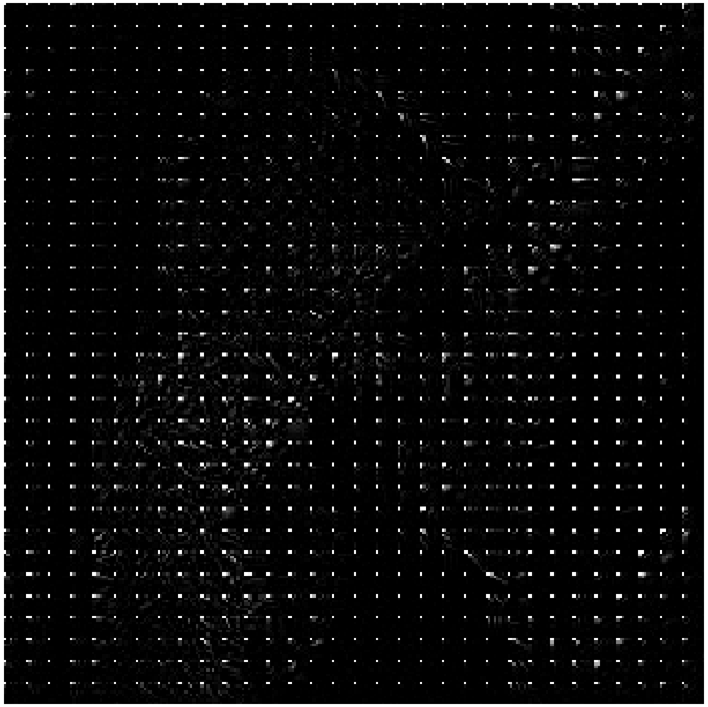
- Large quantization step drive small coefficients down to zero
- The result: many high frequency coefficients become zero, and therefore easier to code
- The low frequency coefficients undergo only minor adjustment
- The quantization matrix can be rescaled by multiplication by a quality factor(QP)

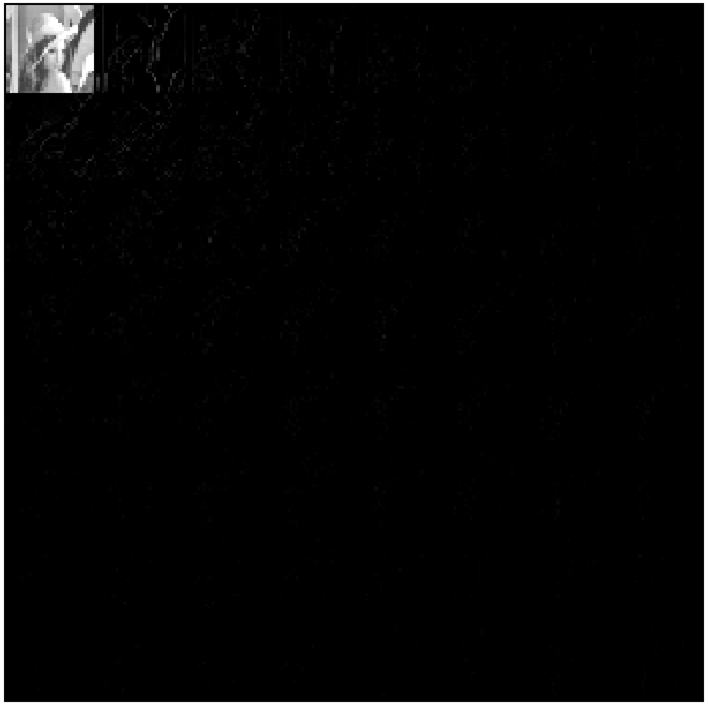
Quality factor

- When using JPEG images, one can set the quality of the image from very low quality to very high quality
- The file size varies inversely with the quality of the image.
- There are basically four "standard" levels of JPEG: Low, med, high, max

JPEG encoding

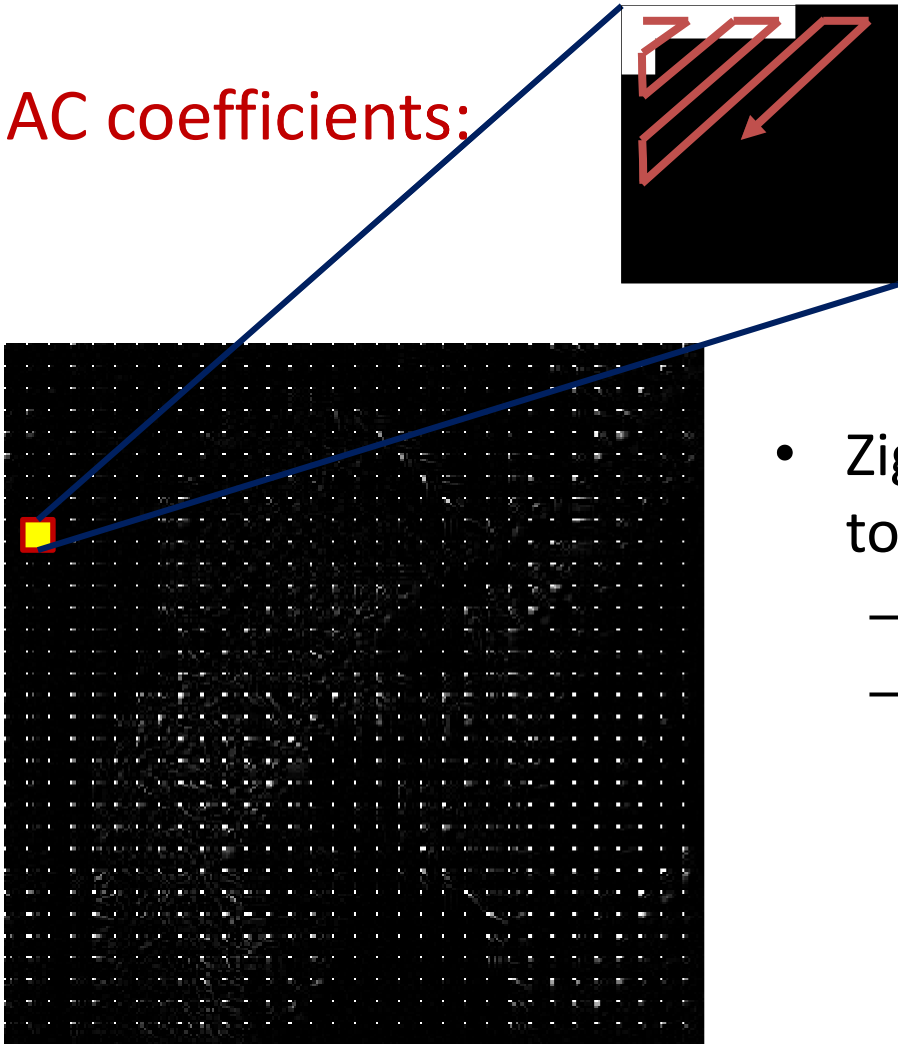
- Quantized DC values are coded by DPCM from block to block to remove the residual correlation.





JPEG encoding

- AC coefficients:



- Zig-zag reordering is performed to achieve large runs of zeros
  - Encoding of zero-runs
  - Entropy coding (Huffman)

JPEG encoding

- DC values:
  - Quantized DC values are coded by DPCM from block to block to remove the residual correlation.

- AC coefficients:
  - Zig-zag reordering is performed to achieve large runs of zeros
  - Encoding of zero-runs
  - Entropy coding (Huffman)

JPEG decoding

- The encoding steps are reversed
- Huffman decoding
- Run length decoding
- Coefficient de-quantization (each coefficient multiplied by the quantum)
- Inverse DCT

JPEG performance

A photo of a cat with the compression rate decreasing, and hence quality increasing, from left to right

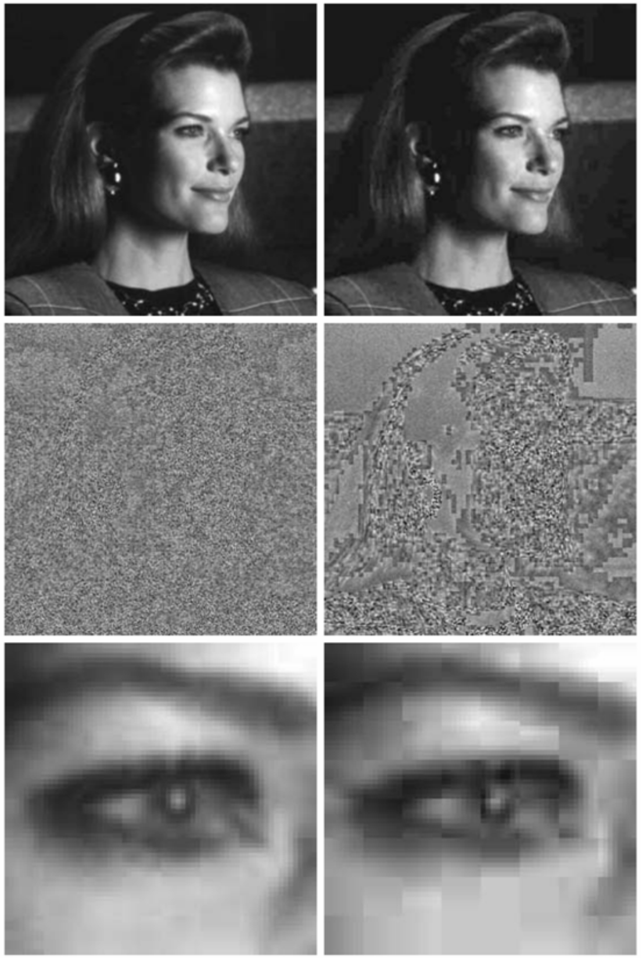
From Wikipedia.org

JPEG performance



Quality max - Size: 61k Quality med - Size: 14k Quality low - Size: 4k

JPEG Baseline Example



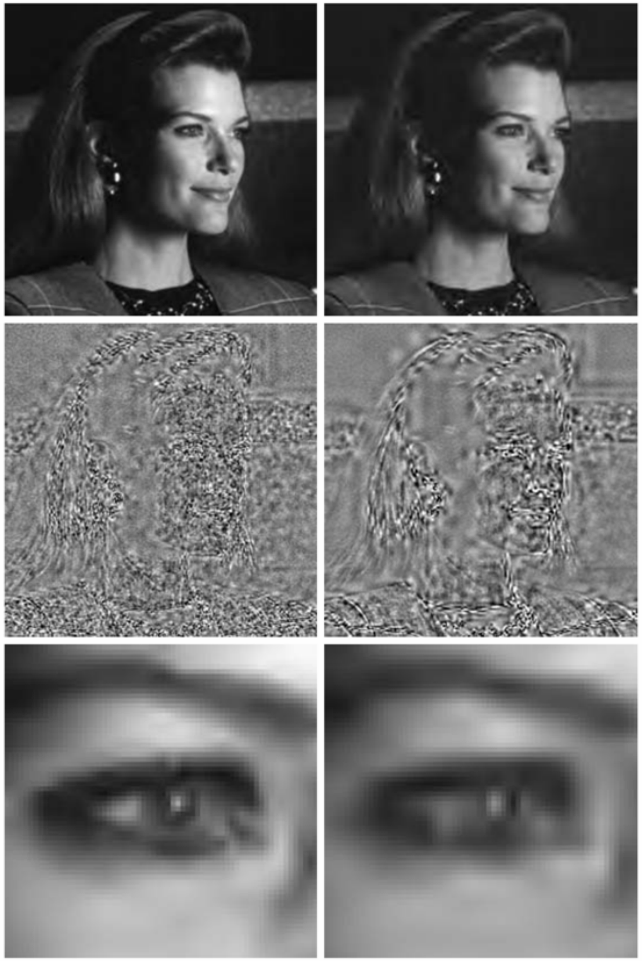
Reconstructed images

Differences between original images and reconstructed images (RMS errors 2.5 on left, 4.4 on right)

Details of reconstructed images

Left: JPEG Compression with, CR = 18:1
Right: JPEG Compression with, CR = 42:1

JPEG 2000 Example



Reconstructed images

Differences between original images and reconstructed images (RMS errors 3.7 on left, 5.9 on right)

Details of reconstructed images

Left: JPEG 2000 compression, CR = 42:1
Right: JPEG 2000 compression, CR = 88:1

JPEG performance

Original image

Encoded @ 24 bits per pixel

JPEG performance

Quality 95/100
3.926 bits per pixel (bpp)
CR = 24/3.926 = 6.1

JPEG performance

Quality 50/100

1.067 bits per pixel (bpp)

CR = 22.5

JPEG performance

Quality 25/100

0.705 bits per pixel (bpp)

CR = 34.0

JPEG performance

Quality 5/100 (min.useful)

0.291 bits per pixel (bpp)

CR = 82.5

Thanks