Lecture 7: Image coding and compression

Robin Strand

Centre for Image Analysis

Swedish University of Agricultural Sciences

Uppsala University

Today

- Information and Data
- Redundancy
- Image Quality
- Coding
- Compression
- File formats

Redundancy *information and data*

- *Data* is **not** the same thing as *information*.
- Data is the means with which information is expressed. The amount of data can be much larger than the amount of information.
- Redundant data doesn't provide additional information.
- Image coding or compression aims at reducing the amount of data while keeping the information by reducing the amount of redundancy.

Definitions

$n_1 =$ data

$n_2 =$ data – redundancy (i.e., data after compression)

Compression ratio $= C_R = \frac{n_1}{n_2}$

Relative redundancy $= R_D = 1 - \frac{1}{C_R}$

Different Types of Redundancy

- Coding Redundancy
  - Some gray levels are more common than other.
- Interpixel redundancy
  - The same gray level may cover a large area.
- Psycho-Visual Redundancy
  - The eye can only resolve about 32 gray levels locally.

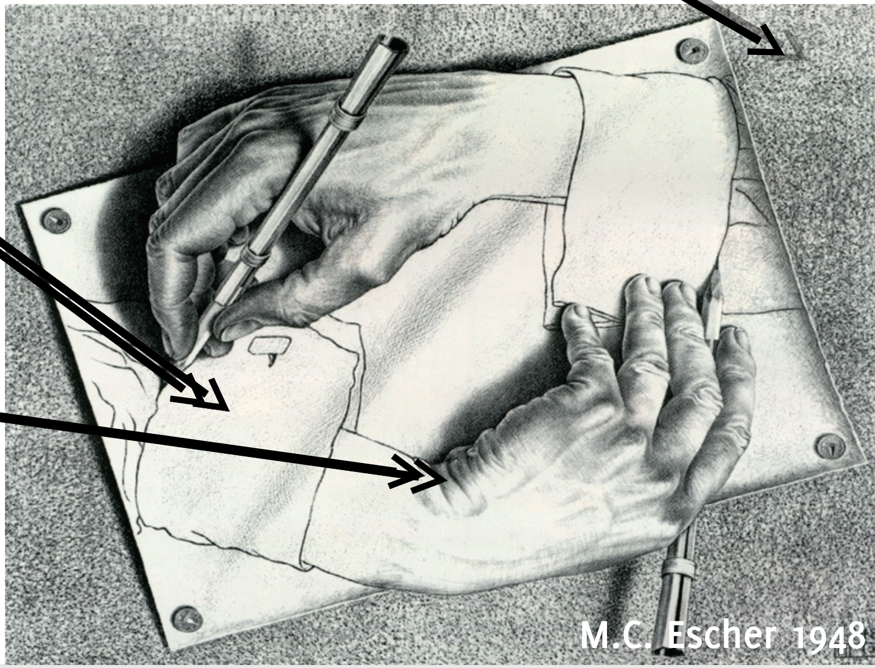


Image Coding and Compression

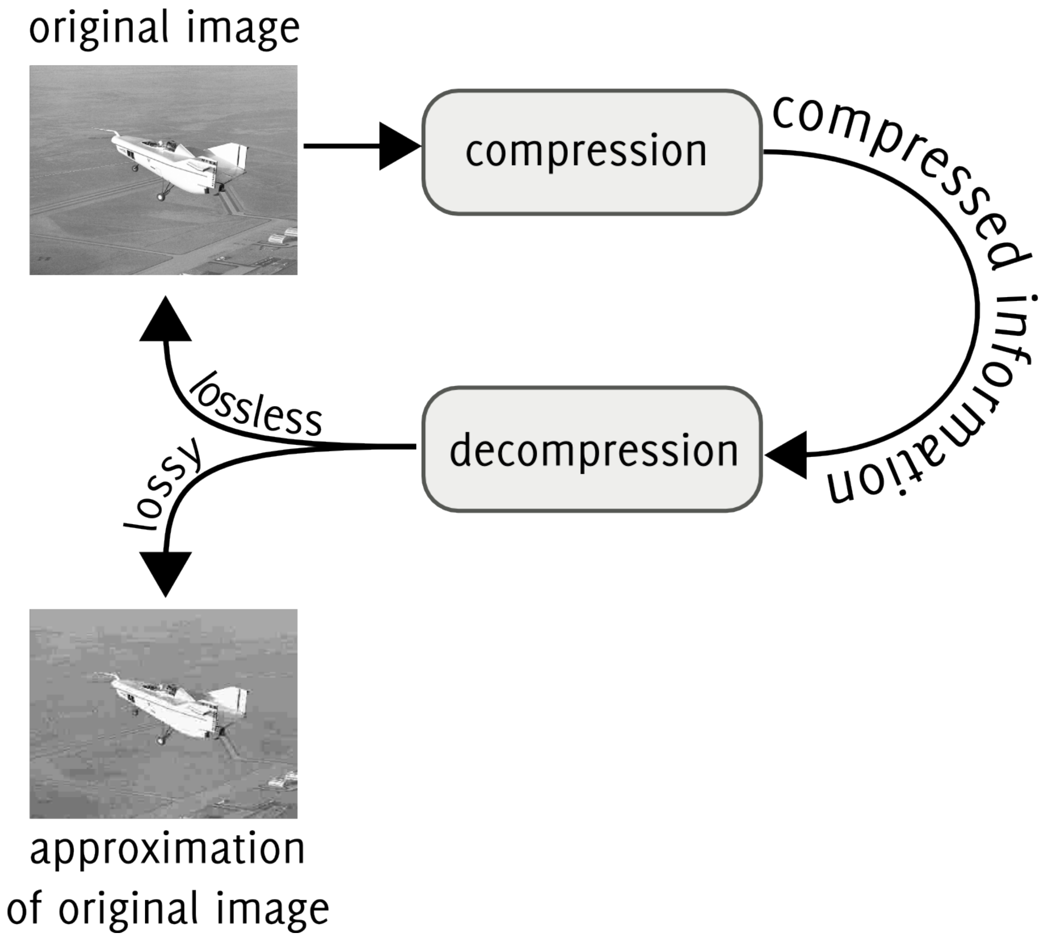


Image Compression

Image compression can be:

- **Reversible** (lossless), with no loss of information.
  - The image after compression and decompression is identical to the original image. Often necessary in image analysis applications.
  - The compression ratio is typically 2 to 10 times.
- **Non reversible** (lossy), with loss of some information.
  - Lossy compression is often used in image communication, compact cameras, video, www, etc.
  - The compression ratio is typically 10 to 30 times.

Image Coding and Compression

- Image coding
  - How the image data can be represented.
- Image compression
  - Reducing the amount of data required to represent an image.
  - Enabling efficient image storing and transmission.

Objective Measures of Image Quality

- Error

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

- Total Error

$$e_{tot} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(\hat{f}(x,y) - f(x,y) \right)$$

- Root-Mean-Square

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

Subjective Measures of Image Quality

- Problem
  - The objective image quality measures previously shown does not always fit with our perception of image quality.
- Solution
  - Let a number of test persons rate the image quality of the images on a scale. This will result in a subjective measure of image quality, or rather fidelity, but it will be based on how we perceive the quality of the images.

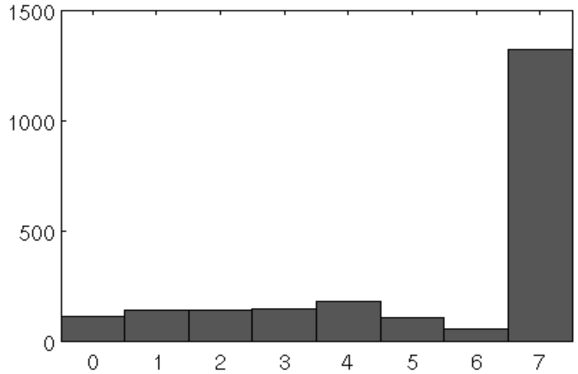
Measure the amount of data

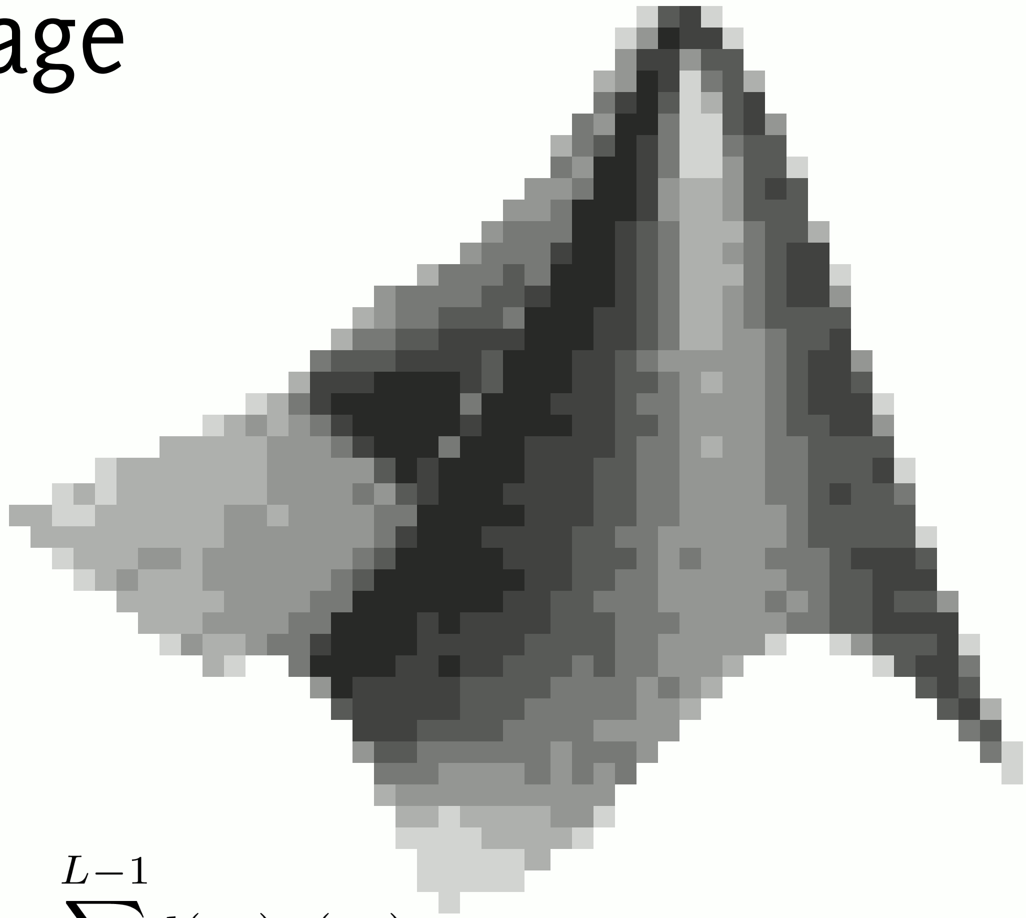
- The amount of data in an $M \times N$ image with L gray levels is equal to MNL_{avg}, *where*

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k),$$

$l(r_k)$ is the number of bits used to represent gray level r_k and $p(r_k)$ is the probability of gray level r_k in the image.

Example 3-bit image





r_k	count	$p(r_k)$	source
0	113	0.051	000
1	139	0.063	001
2	142	0.064	010
3	145	0.066	011
4	181	0.082	100
5	105	0.047	101
6	52	0.023	110
7	1323	0.601	111

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k)$$

$$L_{avg} = 3$$

Dealing with coding redundancy

- **Basic idea:** Different gray levels occur with different probability (non uniform histogram). Use shorter code words for the more common gray levels and longer code words for less common gray levels. This is called *Variable Code Length*.

Huffman Coding

- **First**
  1. Sort the gray levels by decreasing probability
  2. Sum the two smallest probabilities.
  3. Sort the new value into the list.
  4. Repeat 1 to 3 until only two probabilities remains.
- Second
  1. Give the code 0 to the highest probability, and the code 1 to the lowest probability in the summed pair.
  2. Go backwards through the tree one node and repeat from 1 until all gray levels have a unique code.

Example of Huffman coding

r_k	$p(r_k)$
7	0.601
4	0.082
3	0.066
2	0.064
1	0.063
0	0.051
5	0.047
6	0.023

Example of Huffman coding

r_k	$p(r_k)$	node 1
7	0.601	0.601
4	0.082	0.082
3	0.066	0.070
2	0.064	0.066
1	0.063	0.064
0	0.051	0.063
5	0.047	0.051
6	0.023	

Example of Huffman coding

r_k	$p(r_k)$	node 1	node 2
7	0.601	0.601	0.601
4	0.082	0.082	0.114
3	0.066	0.070	0.082
2	0.064	0.066	0.070
1	0.063	0.064	0.066
0	0.051	0.063	0.064
5	0.047	0.051	
6	0.023		

Example of Huffman coding

r_k	$p(r_k)$	node 1	node 2	node 3
7	0.601	0.601	0.601	0.601
4	0.082	0.082	0.114	0.130
3	0.066	0.070	0.082	0.114
2	0.064	0.066	0.070	0.082
1	0.063	0.064	0.066	0.070
0	0.051	0.063	0.064	
5	0.047	0.051		
6	0.023			

Example of Huffman coding

r_k	$p(r_k)$	node 1	node 2	node 3	node 4
7	0.601	0.601	0.601	0.601	0.601
4	0.082	0.082	0.114	0.130	0.152
3	0.066	0.070	0.082	0.114	0.130
2	0.064	0.066	0.070	0.082	0.114
1	0.063	0.064	0.066	0.070	
0	0.051	0.063	0.064		
5	0.047	0.051			
6	0.023				

Example of Huffman coding

r_k	$p(r_k)$	node 1	node 2	node 3	node 4	node 5
7	0.601	0.601	0.601	0.601	0.601	0.601
4	0.082	0.082	0.114	0.130	0.152	0.244
3	0.066	0.070	0.082	0.114	0.130	0.152
2	0.064	0.066	0.070	0.082	0.114	
1	0.063	0.064	0.066	0.070		
0	0.051	0.063	0.064			
5	0.047	0.051				
6	0.023					

Example of Huffman coding

r_k	$p(r_k)$	node 1	node 2	node 3	node 4	node 5	node 6
7	0.601	0.601	0.601	0.601	0.601	0.601	0.601
4	0.082	0.082	0.114	0.130	0.152	0.244	0.396
3	0.066	0.070	0.082	0.114	0.130	0.152	
2	0.064	0.066	0.070	0.082	0.114		
1	0.063	0.064	0.066	0.070			
0	0.051	0.063	0.064				
5	0.047	0.051					
6	0.023						

Huffman Coding

- First
  1. Sort the gray levels by decreasing probability
  2. Add the two smallest probabilities.
  3. Sort the new value into the list.
  4. Repeat 1 to 3 until only two probabilities remains.
- Second
  1. Give the code 0 to the highest probability, and the code 1 to the lowest probability in the summed pair.
  2. Go backwards through the tree one node and repeat from 1 until all gray levels have a unique code.

Example of Huffman coding

Assigning codes

r_k	$p(r_k)$	**node 1**	**node 2**	**node 3**	**node 4**	**node 5**	**node 6**	
7	0.601	0.601	0.601	0.601	0.601	0.601	0.601	0
4	0.082	0.082	0.114	0.130	0.152	0.244	0.396	1
3	0.066	0.070	0.082	0.114	0.130	0.152		
2	0.064	0.066	0.070	0.082	0.114			
1	0.063	0.064	0.066	0.070				
0	0.051	0.063	0.064					
5	0.047	0.051						
6	0.023							

Example of Huffman coding

Assigning codes

r_k	$p(r_k)$	node 1	node 2	node 3	node 4	node 5		node 6	
7	0.601	0.601	0.601	0.601	0.601	0.601	0	0.601	0
4	0.082	0.082	0.114	0.130	0.152	0.244	10	0.396	1
3	0.066	0.070	0.082	0.114	0.130	0.152	11		
2	0.064	0.066	0.070	0.082	0.114				
1	0.063	0.064	0.066	0.070					
0	0.051	0.063	0.064						
5	0.047	0.051							
6	0.023								

Example of Huffman coding
Assigning codes

r_k	$p(r_k)$	node 1	node 2	node 3	node 4		node 5		node 6	
7	0.601	0.601	0.601	0.601	0.601	0	0.601	0	0.601	0
4	0.082	0.082	0.114	0.130	0.152	11	0.244	10	0.396	1
3	0.066	0.070	0.082	0.114	0.130	100	0.152	11		
2	0.064	0.066	0.070	0.082	0.114	101				
1	0.063	0.064	0.066	0.070						
0	0.051	0.063	0.064							
5	0.047	0.051								
6	0.023									

Example of Huffman coding
Assigning codes

r_k	$p(r_k)$	node 1	node 2	node 3	node 4	node 5	node 6
7	0.601	0.601	0.601	0.601 0	0.601 0	0.601 0	0.601 0
4	0.082	0.082	0.114	0.130 100	0.152 11	0.244 10	0.396 1
3	0.066	0.070	0.082	0.114 101	0.130 100	0.152 11	
2	0.064	0.066	0.070	0.082 110	0.114 101		
1	0.063	0.064	0.066	0.070 111			
0	0.051	0.063	0.064				
5	0.047	0.051					
6	0.023						

Example of Huffman coding
Assigning codes

r_k	$p(r_k)$	node 1	node 2		node 3		node 4		node 5		node 6	
7	0.601	0.601	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0
4	0.082	0.082	0.114	101	0.130	100	0.152	11	0.244	10	0.396	1
3	0.066	0.070	0.082	110	0.114	101	0.130	100	0.152	11		
2	0.064	0.066	0.070	111	0.082	110	0.114	101				
1	0.063	0.064	0.066	1000	0.070	111						
0	0.051	0.063	0.064	1001								
5	0.047	0.051										
6	0.023											

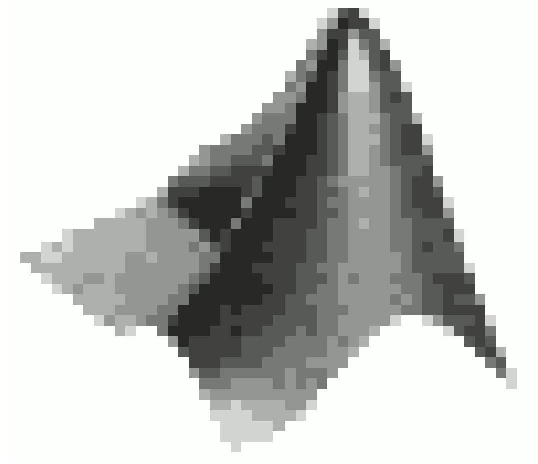
Example of Huffman coding
Assigning codes

r_k	$p(r_k)$	node 1		node 2		node 3		node 4		node 5		node 6	
7	0.601	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0
4	0.082	0.082	110	0.114	101	0.130	100	0.152	11	0.244	10	0.396	1
3	0.066	0.070	111	0.082	110	0.114	101	0.130	100	0.152	11		
2	0.064	0.066	1000	0.070	111	0.082	110	0.114	101				
1	0.063	0.064	1001	0.066	1000	0.070	111						
0	0.051	0.063	1010	0.064	1001								
5	0.047	0.051	1011										
6	0.023												

Example of Huffman coding
Assigning codes

r_k	$p(r_k)$	code	node 1		node 2		node 3		node 4		node 5		node 6	
7	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0	0.601	0
4	0.082	110	0.082	110	0.114	101	0.130	100	0.152	11	0.244	10	0.396	1
3	0.066	1000	0.070	111	0.082	110	0.114	101	0.130	100	0.152	11		
2	0.064	1001	0.066	1000	0.070	111	0.082	110	0.114	101				
1	0.063	1010	0.064	1001	0.066	1000	0.070	111						
0	0.051	1011	0.063	1010	0.064	1001								
5	0.047	1110	0.051	1011										
6	0.023	1111												

Example of Huffman coding



r_k	$p(r_k)$	code
7	0.601	0
4	0.082	110
3	0.066	1000
2	0.064	1001
1	0.063	1010
0	0.051	1011
5	0.047	1110
6	0.023	1111

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k) = 2.10$$

(Before Huffman coding $L_{avg} = 3$)

$$C_R = \frac{n_1}{n_2} = \frac{3}{2.10} = 1.43$$

$$R_D = 1 - \frac{1}{C_R} = \frac{n_1 - n_2}{n_1} = \frac{3 - 2.10}{3} = 0.3$$

Huffman Coding

- The Huffman code is completely reversible, i.e., lossless.
- The table for the translation has to be stored together with the coded image.
- The resulting code is unambiguous.
  - That is, for the previous example, the encoded string 011011101011 can only be parsed into the code words 0, 110, 1110, 1011 and decoded as 7, 4, 5, 0.
- The Huffman code does not take correlation between adjacent pixels into consideration.

Interpixel Redundancy

Also called spatial or geometric redundancy

- Adjacent pixels are often correlated, i.e., the value of neighboring pixels of an observed pixel can often be predicted from the value of the observed pixel.

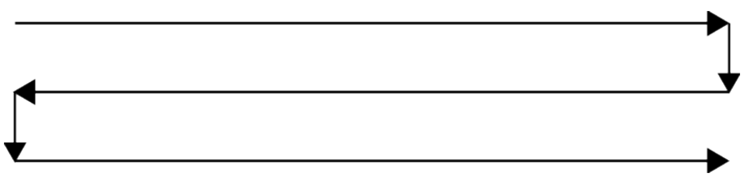
Coding methods:

- Run-length coding
- Difference coding

Run-length coding

- Every code word is made up of a pair (g,l) where g is the gray level, and l is the number of pixels with that gray level (length or "run").
- E.g., 56 56 56 82 82 82 83 80
  56 56 56 56 56 80 80 80

  creates the run-length code (56,3)(82,3)(83,1)(80,4)(56,5)
- The code is calculated row by row.



Difference Coding

- Keep the first pixel in a row. The rest of the pixels are stored as the difference to the previous pixel
- Example:

original	56	56	56	82	82	82	83	80	80	80	80
code $f(x_i)$	56	0	0	26	0	0	1	-3	0	0	0

- The code is calculated row by row.
- Both run-length and difference coding are reversible and can be combined with, e.g., Huffman coding.

Example of Combined Difference and Huffman Coding

Original image

9	8	7	7	7	5	5	5
7	7	7	7	4	4	5	5
6	6	6	9	9	9	6	6
6	6	7	7	7	9	9	9
3	7	7	8	8	8	3	3
3	3	3	3	3	3	3	3
10	10	11	7	7	7	6	6
4	4	5	5	5	2	2	6

Difference image

9	-1	-1	0	0	-2	0	0
0	0	0	3	0	-1	0	0
-1	0	0	3	0	0	-3	0
0	-1	0	0	-2	0	0	3
-3	4	0	1	0	0	-5	0
0	0	0	0	0	0	0	0
7	0	1	-4	0	0	-1	0
0	-1	0	0	3	0	-4	0

Number of bits used to represent the gray scale values: 4

L_{avg}=4

Huffman Coding

7	15	15	15	15	15	16	21	27	37
3	11	11	11	11	12	15	16	21	27
6	10	10	10	10	11	12	15	16	
5	8	8	8	8	10	11	12		
9	7	7	7	8	8	10			
4	4	4	5	7	8				
8	4	4	4	5					
2	2	3	4						
10	2	2							
11	1								

7	10	10	10	10	10	01	00	1	0
3	000	000	000	000	11	10	01	00	1
6	001	001	001	001	000	11	10	01	
5	010	010	010	010	001	000	11		
9	110	110	110	011	010	001			
4	0110	0110	111	110	011				
8	0111	0111	0110	111					
2	1111	1111	0111						
10	11100	1110							
11	11101								

L_{avg}=3.01

C_R=4/3.01=1.33

Difference Coding and Huffman Coding

0	42	42	42	42	42	42	42	42	42
-1	7	7	7	7	7	7	8	14	22
3	4	4	4	4	4	7	7	8	
-2	2	2	3	4	4	4	7		
1	2	2	2	3	4	4			
-3	2	2	2	2	3				
-4	2	2	2	2					
-5	1	2	2						
7	1	1							
9	1								

0	0	0	0	0	0	0	0	0	0
-1	100	100	100	100	100	100	11	10	1
3	110	110	110	110	110	101	100	11	
-2	10100	10100	1011	111	111	110	101		
1	10101	10101	10100	1011	1010	111			
-3	1110	1110	10101	10100	1011				
-4	1111	1111	1110	10101					
-5	10111	10110	1111						
7	101100	10111							
9	101101								

$L_{avg}=2$

$C_R=4/2=2$

LZW Coding

- **LZW**, **L**empel-**Z**iv-**W**elch
- In contrast to Huffman with variable code length LZW uses fixed lengths of code words which are assigned to variable length sequences of source symbols.
- The coding is done from left-to-right and row-by-row.
- Requires no a priori knowledge of the probability of occurrence of the symbols to be encoded.
- Removes some of the interpixel redundancy.
- During encoding a dictionary or “code-book” with symbol sequences is created which is recreated when decoding.

Psycho-Visual Redundancy

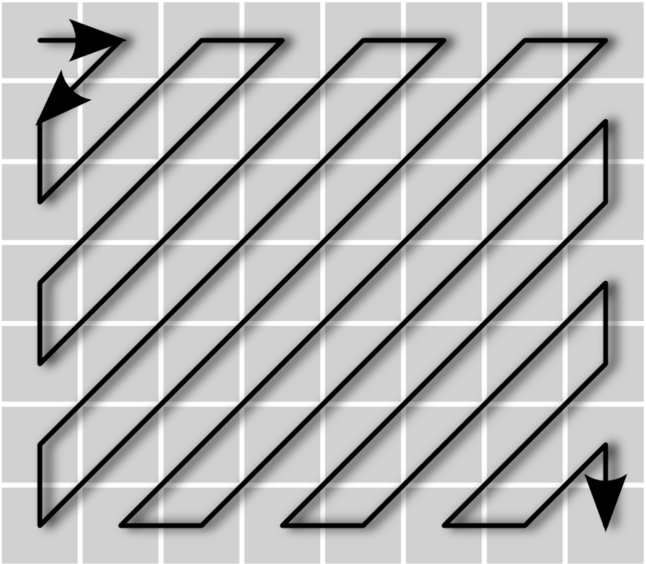
- If the image will only be used for visual observation much of the information is usually psycho-visual redundant. It can be removed without changing the visual quality of the image. This kind of compression is usually lossy.

50 kB (uncompressed TIFF)

5 kB (JPEG)

Transform Coding

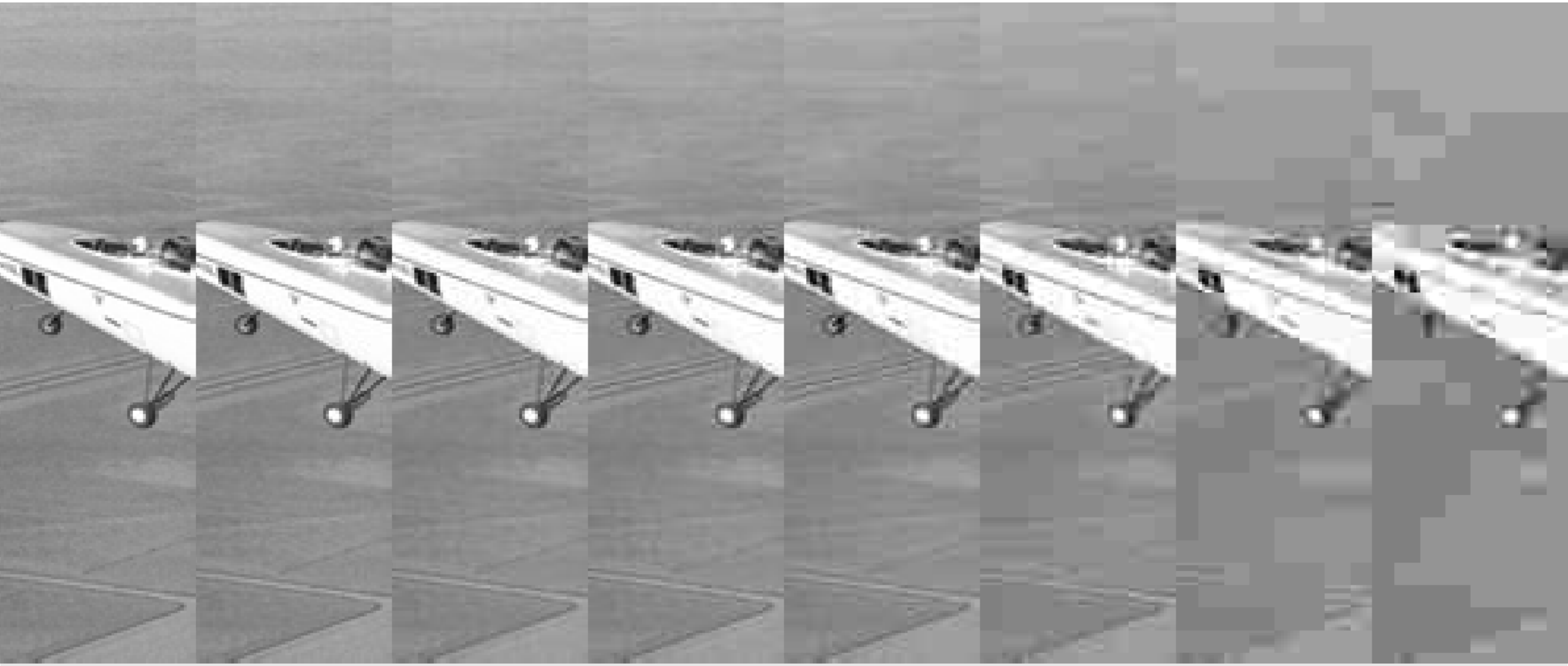
- Divide the image into blocks.
- Transform each block with discrete cosine transform or discrete Fourier transform.
- The transformed blocks are truncated to exclude the least important data.
- Code the resulting data using variable length coding, e.g., Huffman coding in a zigzag scan pattern.



JPEG: example of transform coding

File size in bytes
JPEG quality

9486	3839	2086	1711	1287	822	533	380
100 %	90 %	80 %	60 %	40 %	20 %	10 %	5 %



File Formats with **Lossy** Compression

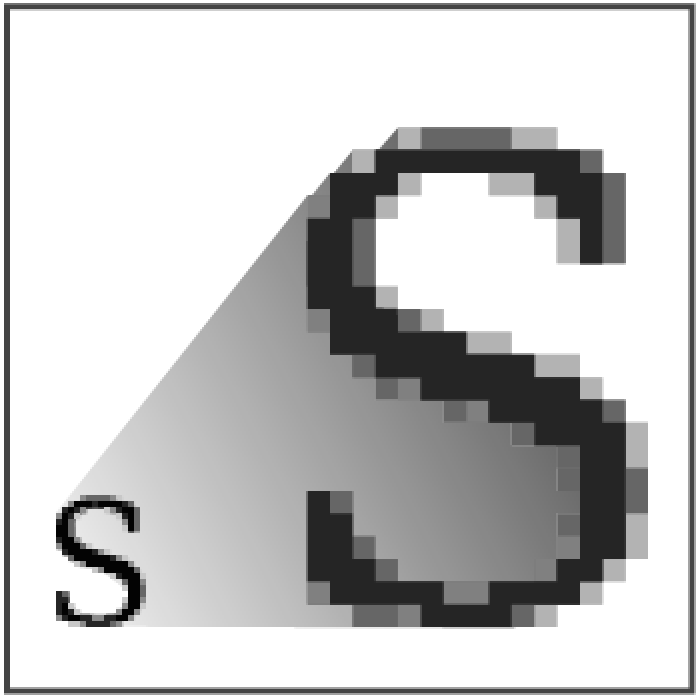
- **JPEG**, **J**oint **P**hotographic **E**xperts **G**roup, based on a cosine transform on 8x8 pixel blocks and Run-Length coding. Give arise to ringing and block artifacts. (`.jpg .jpe .jpeg`)
- **JPEG2000**, created by the **J**oint **P**hotographic **E**xperts **G**roup in **2000**. Based on *wavelet* transform and is superior to JPEG. Give arise only to ringing artifacts and allows flexible decompression (progressive transmission, region of interest, ...) and reading. (`.jp2 .jpx`)

File Formats with **Lossless** Compression

- **TIFF**, **T**agged **I**mage **F**ile **F**ormat, flexible format often supporting up to 16 bits/pixel in 4 channels. Can use several different compression methods, e.g., Huffman, LZW.
- **GIF, G**raphics **I**nterchange **F**ormat. Supports 8 bits/pixel in one channel, that is only 256 colors. Uses LZW compression. Supports animations.
- **PNG, P**ortable **N**etwork **G**raphics, supports up to 16 bits/pixel in 4 channels (RGB + transparency). Uses Deflate compression (~LZW and Huffman). Good when interpixel redundancy is present.

Vector based file formats

- Uses predefined shapes



Vector based file formats

- **PS**, **P**ost**S**cript, is a page description language developed in 1982 for sending text documents to printers.
- **EPS, E**ncapsulated **P**ost**S**cript, like PS but can embed raster images internally using the TIFF format.
- **PDF**, **P**ortable **D**ocument **F**ormat, widely used for documents and are supported by a wide range of platforms. Supports embedding of fonts and raster/bitmap images. Beware of the choice of coding. Both lossy and lossless compressions are supported.
- **SVG, S**calable **V**ector **G**raphics, based on XML supports both static and dynamic content. All major web browsers supports it (Internet Explorer from version 9).

Choosing image file format

- Image analysis
  - Lossless formats are vital. TIFF supports a wide range of different bit depths and lossless compression methods.
- Images for use on the web
  - JPEG for photos (JPEG2000), PNG for illustrations. GIF for small animations. Vector format: SVG, nowadays supported by web browsers.
- Line art, illustrations, logotypes, etc.
  - Lossless formats such as PNG etc. (or a vector format)

Lossy, lossless, and vector graphics