.

# Image compression

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

**Methods for Image Processing**

academic year 2018–2019

## Data and information

- The representation of images in a *raw* format requires a considerable amount of bits.
  - For a 3648×2736 RGB image (10 Megapixels), 1 byte per color channel:
    $3648 \times 2736 \times 3 \times 1 = 29\,942\,784$ bytes = 28.6 MiB.
- However, the *information* contained into the image can generally be described using a smaller quantity of bits.
  - The storage of the previous image (in JPG format) requires 3 618 322 bytes = 3.45 MiB.
- The first number (28.6 MiB) is referred to the case of considering the representation of the image as a collection of three bidimensional array of pixels, were each pixels is represented using a byte.
  - It is the format used to visualize or processing an image.
- The second number is related to a more complex representation of the same amount of pixels.
  - This format is more convenient to store the image.

.

## Data and information (2)

- Data is the physical implementation of the information.
  - The way information is represented and conveyed.
- For the same piece of information, infinite representations are possible.
- Data compression is the branch of the theory of information that studies the techniques for representing information and the computational resources they require.

## Measuring data compression

Given two representations of the same information, they can be compared in terms of their size, $b$ and $b'$ (measured, e.g., in bits), using the following measures:

- *Compression ratio*, $C$:

$$C = \frac{b}{b'}$$

- *Relative redundancy*, $R$:

$$R = 1 - \frac{1}{C}$$

- With the numbers of the previous example:
  - $C = 8.28$
  - $R = 0.879$
  - The 87.9% of the raw data are redundant with respect to the JPG representation, which means that one byte of JPG data contains the same information amount of 8.28 byte of raw data.

.

## Types of redundancies

- Coding redundancy
  - fewer bits can be used for representing the same amount of information (possible configurations)
- Spatial and temporal redundancy
  - spatial and temporal correlation of the pixels attributes
- Irrelevant information
  - information that is ignored by the human visual system or unuseful for the application

## Coding redundancy

- The intensity of the pixels can be seen as a discrete random variable, $r$:
  - $r \in [0, L-1]$
- Counting the number of occurrences of the $k$-th intensity level: $p(r = k) = p(r_k) = \frac{n_k}{MN}$
- If $l(r_k)$ is the number of bits used for coding the $k$-th intensity level, the average length for each symbol is: $L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k)p(r_k)$

.

## Coding redundancy: an example

| $r_k$ | $p_r(r_k)$ | Code 1 | $l_1(r_k)$ | Code 2 | $l_2(r_k)$ |
|---|---|---|---|---|---|
| $r_{87} = 87$ | 0.25 | 01010111 | 8 | 01 | 2 |
| $r_{128} = 128$ | 0.47 | 10000000 | 8 | 1 | 1 |
| $r_{186} = 186$ | 0.25 | 11000100 | 8 | 000 | 3 |
| $r_{255} = 255$ | 0.03 | 11111111 | 8 | 001 | 3 |
| $r_k$ for $k \neq 87, 128, 186, 255$ | 0 | — | 8 | — | 0 |

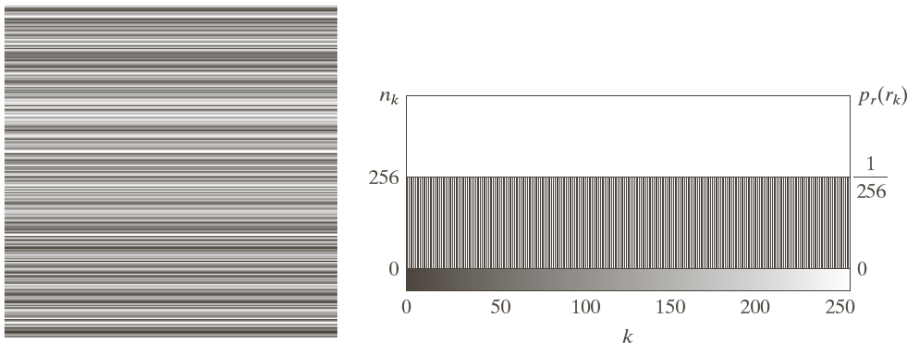
- Only four intensity levels, 256×256 pixels image.
- Code 1 uses 8 bits per pixels, code 2 uses a variable-length approach.
- For code 1, $L_{\text{avg}} = 8$ bits, while for code 2, $L_{\text{avg}} = 1.81$ bits.
- $C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} \approx 4.42$
- $R = 1 - \frac{1}{4.42} = 0.774$

## Spatial (or temporal) redundancy

- Often, the intensity (color) of a given pixel is correlated to that of its neighboring pixels.
  - The single pixel carry a small amount of information.
- There are several way for exploiting this fact for obtaining a more compact description of the image.
- For instance, if neighboring pixels are likely to have the same color (e.g., cartoons), instead of storing the color of each pixel, the intensity value can be followed by the number of repetitions.
  - *Run-length* coding.
- If the color of neighboring pixels is similar, their values can be predicted and only the differences can be stored.
  - Predictive coding.
- Techniques of this kind are based on transformations called mappings.
  - They can be reversible (*lossless*) or irreversible (*lossy*).

.

## Spatial redundancy: an example



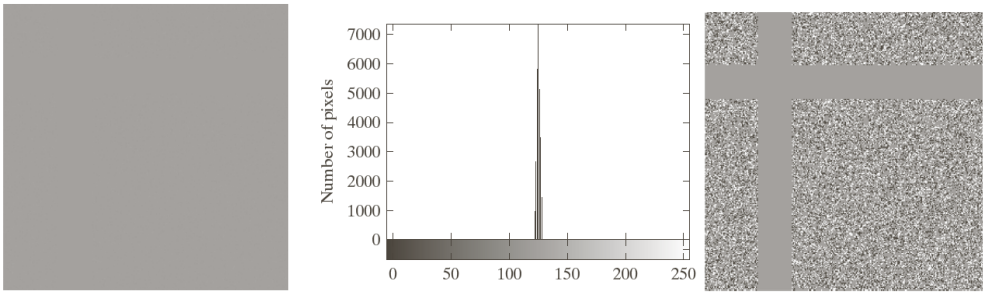

- Pixel of the same row share the same intensity.
- All the 256 intensity levels are used, and they have the same number of pixels (the intensities are equiprobable).
  - No advantage in using variable-length encoding.
- Only the intensity of the first pixel and the number of repetitions need to be stored.
  - *Run-length* encoding.

## Irrelevant information

- Omitting unuseful information is an effective way of reducing the amount of data to be stored.
- Among this kind of information, details that cannot be perceived (or whose absence is neutral for the context) is a notable example.
- However, when these details can carry useful information, they have to be preserved.
- This is the case of medical images, where a compromise between feasibility and information preservation have to be found.
  - Besides, compression process should not add artifacts that could lead to misinterpretation.
- This kind of redundancy is often handled with *quantization*.
  - Sets of intensities (or configurations of group of pixels) are mapped on a single intensity level (or on a single pattern).

.

## Irrelevant information: an example




a | b | c

(a) There is a structure that the human eye (brain) cannot detect.

(b) The histogram shows that although perceived as uniform, the midgray image is instead composed of pixel in a narrow range of intensities (in the range [125, 131]).

(c) When equalized, the structure in the image becomes apparent.

## Fidelity measurement

- How can be measured the fidelity of the representation of an approximate description $\hat{f}$ of an original image, $f$?
  - Objective fidelity criteria
    - Mathematical formalization of the concept of difference.
  - Subjective fidelity criteria
    - A group of people evaluate the quality of the representation.
- root-mean-square error, $e_{\text{rms}}$:

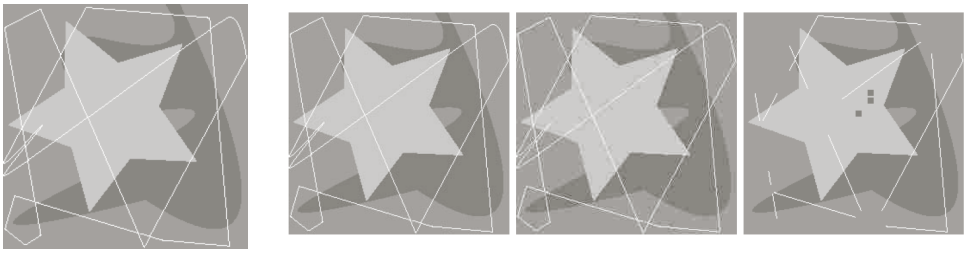
$$e_{\text{rms}} = \left( \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left( \hat{f}(x, y) - f(x, y) \right)^2 \right)^{\frac{1}{2}}$$

- Signal-to-noise ratio, SNR:

$$\text{SNR} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left( \hat{f}(x, y) - f(x, y) \right)^2}$$

.

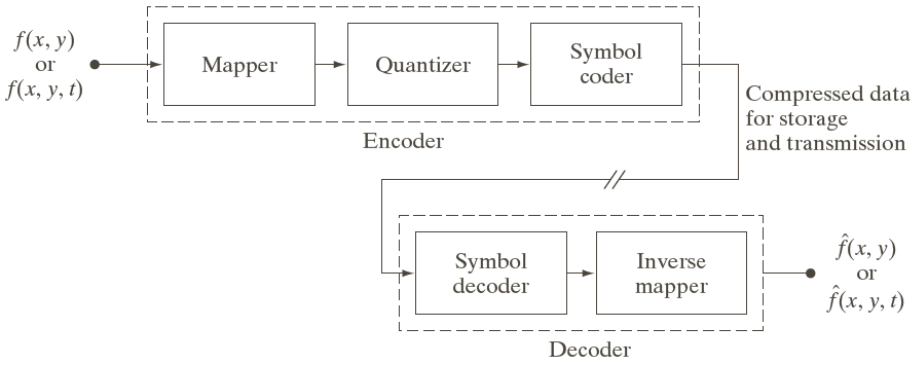
## Fidelity measurement (2)



$e_{\text{rms}} = 5.17$ *excellent* | $e_{\text{rms}} = 15.67$ *passable* | $e_{\text{rms}} = 14.17$ *unusable*

- Simple pixel-wise differences can hardly model human perception.

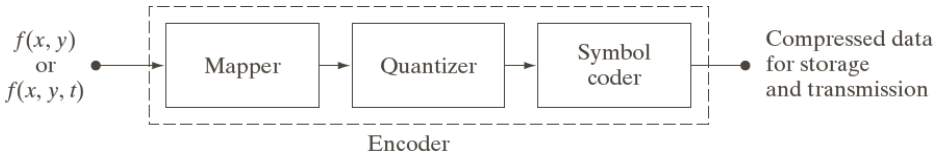
## Image compression system




- The compression scheme is generally composed of two elements:
  - *encoder*: performs compression;
  - *decoder*: recovers the original data.
- A *codec* is a device or program that can perform both the tasks.

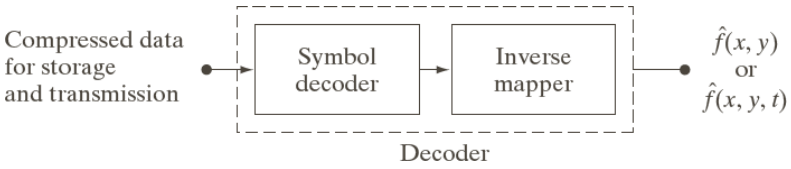
.

## Compression process




- The *mapper* reduces the spatial and temporal redundancy.
- The *quantizer* reduces the accuracy of the representation using an appropriate criterion (e.g., the final size or the bit-rate of the transmission).
- The *symbol coder* generates the actual encoding of the symbols output by the previous stage.

## Decompression process




- The *symbol decoder* recovers the original symbols.
- The *inverse mapper* operates the inverse mapping.
- Since the quantization is irreversible, no inverting stage can be realized for reverting the effects of the quantizer.

.

## Huffman coding

| Original source | | Source reduction | | | |
|---|---|---|---|---|---|
| Symbol | Probability | 1 | 2 | 3 | 4 |
| $a_2$ | 0.4 | 0.4 | 0.4 | 0.4 | 0.6 |
| $a_6$ | 0.3 | 0.3 | 0.3 | 0.3 | 0.4 |
| $a_1$ | 0.1 | 0.1 | 0.2 | 0.3 | |
| $a_4$ | 0.1 | 0.1 | 0.1 | | |
| $a_3$ | 0.06 | 0.1 | | | |
| $a_5$ | 0.04 | | | | |

| Original source | | | Source reduction | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Symbol | Probability | Code | 1 | | 2 | | 3 | | 4 | |
| $a_2$ | 0.4 | 1 | 0.4 | 1 | 0.4 | 1 | 0.4 | 1 | 0.6 | 0 |
| $a_6$ | 0.3 | 00 | 0.3 | 00 | 0.3 | 00 | 0.3 | 00 | 0.4 | 1 |
| $a_1$ | 0.1 | 011 | 0.1 | 011 | 0.2 | 010 | 0.3 | 01 | | |
| $a_4$ | 0.1 | 0100 | 0.1 | 0100 | 0.1 | 011 | | | | |
| $a_3$ | 0.06 | 01010 | 0.1 | 0101 | | | | | | |
| $a_5$ | 0.04 | 01011 | | | | | | | | |

## Huffman coding (2)

- It is optimal when coding is operated one symbol at a time.
  - Number of symbols and its distribution is fixed.
- The decoder is simply a look-up table.
- The construction of the decoding table requires computational resources linearly proportional to the number of symbols to be coded.
  - Predefined Huffmann codes are defined in several standards.

.

## Golomb coding

- The Golomb codes can encode non negative integers.
- The Golomb code $G_m(n)$ encodes the integer $n$ with respect to $m$ combining:
  - the code of $\lfloor n/m \rfloor$
  - the code of the remainder $n$ mod $m$
- The length of the coding string is proportional to coded value.
  - Smaller $n$, shorter $G_m(n)$.
- The Golomb codes are efficient for coding the differences.
- For $m$ equal to a power of two, the computation of the code is quite simple.
  - Golomb-Rice (or Rice) codes.

## Golomb coding (2)

| $n$ | $G_1(n)$ | $G_2(n)$ | $G_4(n)$ | $G^0_{\exp}(n)$ |
|---|---|---|---|---|
| 0 | 0 | 00 | 000 | 0 |
| 1 | 10 | 01 | 001 | 100 |
| 2 | 110 | 100 | 010 | 101 |
| 3 | 1110 | 101 | 011 | 11000 |
| 4 | 11110 | 1100 | 1000 | 11001 |
| 5 | 111110 | 1101 | 1001 | 11010 |
| 6 | 1111110 | 11100 | 1010 | 11011 |
| 7 | 11111110 | 11101 | 1011 | 1110000 |
| 8 | 111111110 | 111100 | 11000 | 1110001 |
| 9 | 1111111110 | 111101 | 11001 | 1110010 |

1. Form the unary code of $q = \lfloor n/m \rfloor$ ($q$ 1's ended with 0).
2. Let $k = \lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n$ mod $m$, and compute $r'$:

$$r' = \begin{cases} r \text{ truncated to } k-1 \text{ bits} & 0 \leq r < c \\ r+\text{c truncated to } k \text{ bits} & \text{otherwise} \end{cases}$$

3. Concatenate the results of steps 1 and 2.

## Golomb coding (3)

- The exponential-Golomb codes are efficient for encoding of run-lengths.
- An order-$k$ exponential-Golomb code $G^k_{\exp}(n)$ is computed as:
  1. Find an integer $i \geq 0$ such that

$$\sum_{j=0}^{i-1} 2^{j+k} \leq n \leq \sum_{j=0}^{i} 2^{j+k}$$

     and form the unary code of $i$.
  2. Truncate the binary representation of

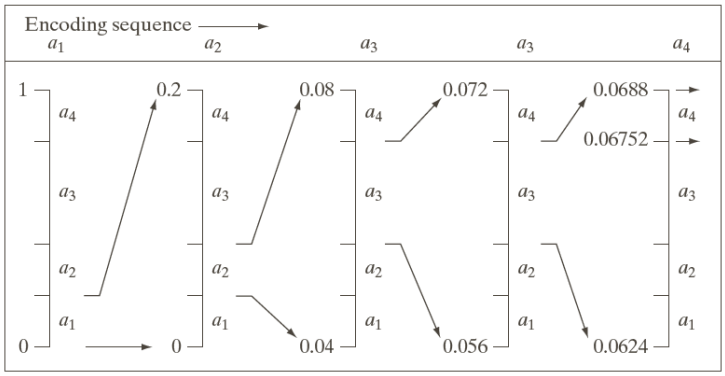
$$n - \sum_{j=0}^{i-1} 2^{j+k}$$

  3. Concatenate the results of steps 1 and 2.

## Arithmetic coding

- Encode a sequence of symbols as a number in [0, 1].

| Source Symbol | Probability | Initial Subinterval |
|---|---|---|
| $a_1$ | 0.2 | [0.0, 0.2) |
| $a_2$ | 0.2 | [0.2, 0.4) |
| $a_3$ | 0.4 | [0.4, 0.8) |
| $a_4$ | 0.2 | [0.8, 1.0) |




- Every number in [0.06752, 0.0688) encodes the sequence $a_1\ a_2\ a_3\ a_3\ a_4$.

.

## LZW coding

- The *Lempel-Ziv-Welch* (LZW) coding addresses spatial redundancy assigning fixed-length code words to variable-length sequence of symbols.
- Do not require a-priori knowledge of the source symbols distribution.
- At the beginning, the dictionary contains only the source symbols.
- When sequences are found for the first time, new entries are added to the dictionary.
- Dictionary is a key element for the representation.
  - The larger, the more effective is the compression.
  - The dictionary can be generated also at the decoder side.
    - No needs of transmitting it.
  - Variants of the basic algorithm allows to compose the dictionary such that it is representative of the actual distribution of the symbols.

## LZW coding (2)

4×4, 8 bit image:

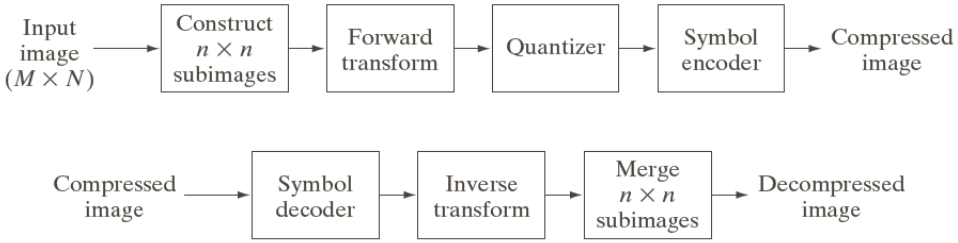
| | | | |
|---|---|---|---|
| 39 | 39 | 126 | 126 |
| 39 | 39 | 126 | 126 |
| 39 | 39 | 126 | 126 |
| 39 | 39 | 126 | 126 |

| Dictionary Location | Entry |
|---|---|
| 0 | 0 |
| 1 | 1 |
| ⋮ | ⋮ |
| 255 | 255 |
| 256 | — |
| ⋮ | ⋮ |
| 511 | — |

| Currently Recognized Sequence | Pixel Being Processed | Encoded Output | Dictionary Location (Code Word) | Dictionary Entry |
|---|---|---|---|---|
| | 39 | | | |
| 39 | 39 | 39 | 256 | 39-39 |
| 39 | 126 | 39 | 257 | 39-126 |
| 126 | 126 | 126 | 258 | 126-126 |
| 126 | 39 | 126 | 259 | 126-39 |
| 39 | 39 | | | |
| 39-39 | 126 | 256 | 260 | 39-39-126 |
| 126 | 126 | | | |
| 126-126 | 39 | 258 | 261 | 126-126-39 |
| 39 | 39 | | | |
| 39-39 | 126 | | | |
| 39-39-126 | 126 | 260 | 262 | 39-39-126-126 |
| 126 | 39 | | | |
| 126-39 | 39 | 259 | 263 | 126-39-39 |
| 39 | 126 | | | |
| 39-126 | 126 | 257 | 264 | 39-126-126 |
| 126 | | 126 | | |

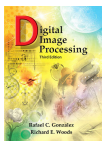
.

## Block transform coding




- The mapper partitions the image in blocks and applies a decorrelating transform.
  - The coefficients are less correlated than the pixels intensities.
  - Few coefficients are sufficient for approximating the block.

## Homeworks and suggested readings



DIP, Sections 8.1–10.2

- pp. 525–555

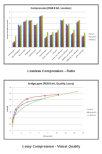


Image Compression Benchmark

- `http://www.imagecompression.info/`