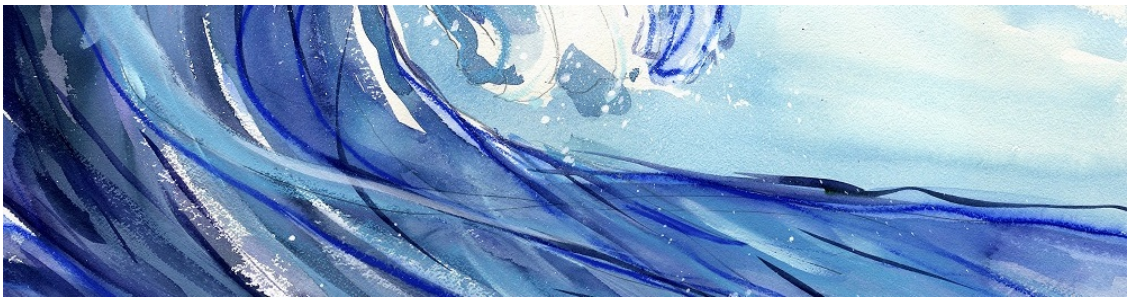
An Introduction to Imaging Sciences



Gabriel Peyré
CNRS & DMA
École Normale Supérieure
gabriel.peyre@ens.fr
www.gpeyre.com
www.numerical-tours.com

September 29, 2017

Presentation

The three chapters of this text are independent and present gentle introductions to a few important mathematical foundations of imaging sciences:

- Chapter 1 presents Shannon theory of compression, and insists in particular on the entropy bound for coding of information.
- Chapter 2 presents the basics of image processing, in particular some important processings (quantization, densoising, colors).
- Chapter 3 presents sampling theory, from Shannon classical sampling to compressed sensing. It also serves as a gentle introduction to the field of inverse problems regularization.

The exposition level for the two chapters is elementary. The last chapter presents more advanced mathematical concepts and results.

Contents

Chapter 1

Claude Shannon and Data Compression

The vast majority of data (text, sound, image, video, etc.) is stored and manipulated in digital form, that is, using integers which are converted into a succession of bits (0 and 1). Conversion from the continuous analog world to these discrete numerical representations is described by the theory developed by Claude Shannon (April 30, 1916–February 24, 2001), the founding father of the theory of information. The impact of this theory on our society is absolutely colossal. Yet his name is almost unknown to the general public. The centenary of the birth of Claude Shannon is therefore a good excuse to present the work of a major scientist.

1.1 Numeric Data and Coding

In the digital world that surrounds us, all data (images, films, sounds, texts, etc.) are coded in the form of a succession of 0 and 1. This encoding is not limited to storage on computers, it is also central for communications over the internet (email, "streaming" video, etc.) as well as for applications as diverse as music players, e-readers or mobile phones.

However, data (eg text, sounds, images, or videos) is initially represented as a succession of *symbols*, which are not necessarily 0 or of 1. For example, for the case of a text, the symbols are the letters of the alphabet. For the case of images, these are the values of the pixels. It is therefore necessary to be able to convert this sequence of symbols into a sequence of 0 and 1. It is also necessary to be able to do it in an economical way, that is to say using the shortest possible sequence. This is crucial in order to be able to store this data efficiently on a hard disk, or to transmit them quickly on the Internet. This problem of *compression* has become a major issue because the stored and transmitted data grow exponentially.

The theory developed by Claude Shannon describes the theoretical and algorithmic bases of this coding. He mathematically formalized the three key stages of conversion from the analog world to the digital world:

(i) *sampling*[1], which allows one to switch from continuous data to a succession of numbers;

(ii) *coding*[2] (also known as compression), which allows one to move to a more compact sequence of 0 and 1 (called binary code);

[1] https://en.wikipedia.org/wiki/Sampling_(signal_processing)
[2] https://en.wikipedia.org/wiki/Data_compression

(iii) *error-correcting code*[3], which makes code robust to errors and attacks.

For each of these steps, Claude Shannon has established performance "upper bounds" in [20, 21], under precise assumptions about the data and the transmission channel. These performance bounds set limits that can not be exceeded, regardless of the method used. For example, for the encoding phase (ii), this bound corresponds to the minimum theoretical size of the binary messages making it possible to code the desired information. In the second half of the 20th century, efficient computational methods and algorithms were developed that reach the limits of Shannon, leading to the 21st century on the explosion of the digital age. This article focuses on part (ii) and presents the basics of data compression as defined by Claude Shannon.

You can find at the end of this article a glossary summarizing the most important terms.

1.2 Encoding and Decoding

We will now describe and study the transformation (coding) from the sequence of $\{0, 1, 2, 3\}$ symbols to a binary code, that is, a sequence of 0 and 1.

1.2.1 Example of an Image

In the rest of this article, I will illustrate my remarks using grayscale images. Such an image is composed of pixels. To simplify, we will consider only pixels with 4 levels of gray:

- 0: black,
- 1: dark gray,
- 2: light gray,
- 3: white.

However, all that will be described hereafter can be generalized to an arbitrary number of gray levels (in general, the images that are found on the Internet have 256 levels) and even to color images (which can be decomposed in 3 monochrome images, the red, green and blue components).

Figure 1.1 shows an example of an image with 4 levels of gray, with a zoom on a subset of 5×5 pixels.

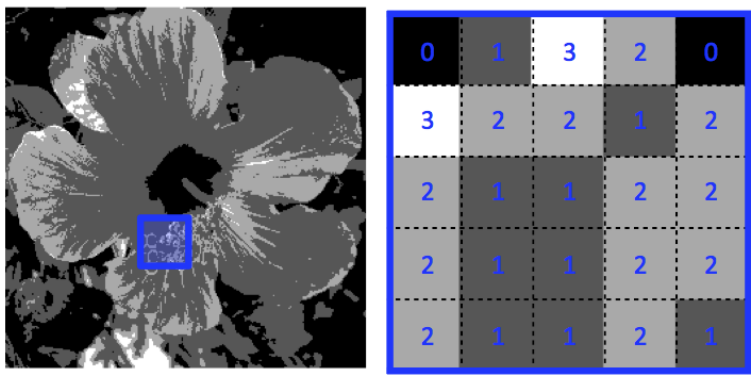


Figure 1.1: A greyscale image and a zoom on a square of 5×5 pixels

[3] https://en.wikipedia.org/wiki/Error_detection_and_correction

We will focus on this set of 25 pixels (the rest of the image is treated in the same way). If we put the corresponding values one after the other, we get the following sequence of symbols, which are numbers between 0 and 3

$$(0,1,3,2,0,3,2,2,1,2,2,1,1,2,2,2,1,1,2,2,2,1,1,2,1).$$

1.2.2 Uniform Coding

The coding step therefore proceeds by associating to each of the symbols $\{0,1,2,3\}$ a code word, which is a sequence of 0 and 1.

One possible strategy is to use coding

$$0 \mapsto 00, \quad 1 \mapsto 01, \quad 2 \mapsto 10, \quad 3 \mapsto 11.$$

This is a particular case of *uniform* coding, which associates with each symbol a code word of fixed length (here of constant length 2).

Thus the sequence of $(0,1,3)$ symbols is coded as

$$(0,1,3) \overset{\text{coding}}{\longmapsto} (00,01,11) \overset{\text{grouping}}{\longmapsto} 000111.$$

The complete sequence of symbols corresponding to the image of 5×5 pixels shown above will give the code

$$000111100011101001101001011010100101101010010110 01.$$

The length (ie the number of 0 and 1) in the sequence 0 and 1 used to encode a message is measured in number of *bits*. Using the previous uniform coding, which uses 2 bits per symbols, as one must code 25 symbols, a length

$$\bar{\mathcal{L}} = 25 \times 2 = 50 \text{ bits}$$

The *bit* ("binary digit") is the fundamental unit of information, and was introduced by John Tukey[4] who was a collaborator of Claude Shannon.

1.2.3 Logarithm and Uniform Coding

If the number N of possible symbols (in this case $N = 4$) is a power of 2, that is $N = 2^\ell$ (here $N = 4 = 2^2$ so that $\ell = 2$), one can always construct such a *uniform* code where one associates to each symbol its binary writing. We have given the example of the uniform coding of $N = 4$ symbols, and the case of $N = 8$ (thus $\ell = 3$) symbols corresponds to the coding

$$0 \mapsto 000, \quad 1 \mapsto 001, \quad 2 \mapsto 010, \quad 3 \mapsto 011,$$

$$4 \mapsto 100, \quad 5 \mapsto 101, \quad 6 \mapsto 110, \quad 7 \mapsto 111.$$

This binary code has a length ℓ, which is called the logarithm in base of 2[5] of N, which is noted

$$N = 2^\ell \quad \Longleftrightarrow \quad \log_2(N) \overset{\text{def.}}{=} \ell.$$

The definition of $\log_2(x)$ also extends to the case where x is not a power of 2, using the definition $\log_2(x) \overset{\text{def.}}{=} \ln(x)/\ln(2)$, where ln is the *natural logarithm*. In this case, $\log_2(x)$ is not an integer. For a strictly positive real number x, the logarithm satisfies $\log_2(1/x) = -\log_2(x)$, so for example, we have $\log_2(1/4) = -\log_2(4) = 2$.

[4] https://en.wikipedia.org/wiki/John_Tukey

[5] https://en.wikipedia.org/wiki/Binary_Logarithm

1.2.4 Variable-length Encoding

An important question is whether we can do better (that is, use fewer bits to code the same sequence of symbols). For example, the following coding may be used instead of a uniform code

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

With such coding, the $(0, 1, 3)$ symbol sequence is coded as

$$(0, 1, 3) \overset{\text{codage}}{\longmapsto} (001, 01, 000) \overset{\text{regroupement}}{\longmapsto} 00101000.$$

The complete sequence of symbols corresponding to the image of 5×5 pixels will give the code

$$0010100010010001101110101111010111110101101.$$

The length of the binary code obtained is therefore now

$$\bar{\mathcal{L}} = 42\text{bits}$$

This shows that it is therefore possible to do better than with a *uniform* coding using a *variable* coding, which associates a variable length code with each symbol.

It is also possible to define the average number of bits per symbol $\mathcal{L}$, which is computed, here for a sequence of 25 symbols, as

$$\mathcal{L} \overset{\text{def.}}{=} \frac{\bar{\mathcal{L}}}{25} = \frac{42}{25} = 1.68 \text{ bits.}$$

Compared to a uniform coding, it is seen that the average number of bits per symbol has changed from $\log_2(N) = 2$ bits to 1.68 bits.

1.2.5 Prefix Coding and Decoding

These codings, uniform or of variable length, would be of no interest if we did not ensure that the message obtained is *decodable*, ie we can find the sequence of symbols at the origin of a binary code. All encodings do not allow for this reverse path.

For uniform encodings, such as coding

$$0 \mapsto 00, \quad 1 \mapsto 01, \quad 2 \mapsto 10, \quad 3 \mapsto 11.$$

it is sufficient to separate the sequence of bits into packets of length $\log_2(N)$(here $N = 4$ and $\log_2(N) = 2$) and use the encoding table in the opposite direction. Thus, the 000111 binary code is decoded as

$$000111 \overset{\text{splitting}}{\longmapsto} (00, 01, 11) \overset{\text{decoding}}{\longmapsto} (0, 1, 3).$$

On the other hand, if we consider the coding

$$0 \mapsto 0, \quad 1 \mapsto 10, \quad 2 \mapsto 110, \quad 3 \mapsto 101,$$

then the bit sequence 1010 can be decoded in two ways:

$$1010 \overset{\text{splitting}}{\longmapsto} (10, 10) \overset{\text{decoding}}{\longmapsto} (1, 1),$$

or

$$1010 \overset{\text{splitting}}{\longmapsto} (101, 0) \overset{\text{decoding}}{\longmapsto} (3, 0).$$

This means that this sequence can be decoded either as the sequence $(1, 1)$, or as the $(3, 0)$ sequence. Note that the 10 encoding word used to encode 1 is the beginning of the 101 word used to encode 3.

To be able to do the decoding in an unambiguous way, it is enough that no word of the coding is the beginning of another word. When this condition is satisfied, we speak of *prefix*[6] and it is therefore possible to carry out the decoding step by step. It is easily verified that this is indeed the case for the non-uniform coding already considered previously

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

The progressive decoding of the symbol message of the pixels of the image is carried out as follows:

$$001010001001000110111010111010111010101101 \longrightarrow \text{decoding } 0$$
$$0\ 01000100100011011101011110101111010101101 \longrightarrow \text{decoding } 1$$
$$0\ 1\ 000100100011011101011110101111010101101 \longrightarrow \text{decoding } 3$$
$$0\ 1\ \ 3\ 100100011011101011110101111010101101 \longrightarrow \text{decoding } 2\ldots$$

1.2.6 Codes and Trees

As shown in Figure 1.2, in the top left, it is possible to place the set of binary codes of less than ℓ bits in a tree of depth $\ell + 1$. The 2^ℓ words of length exactly ℓ occupy the sheets, and the shorter words are the inner nodes.

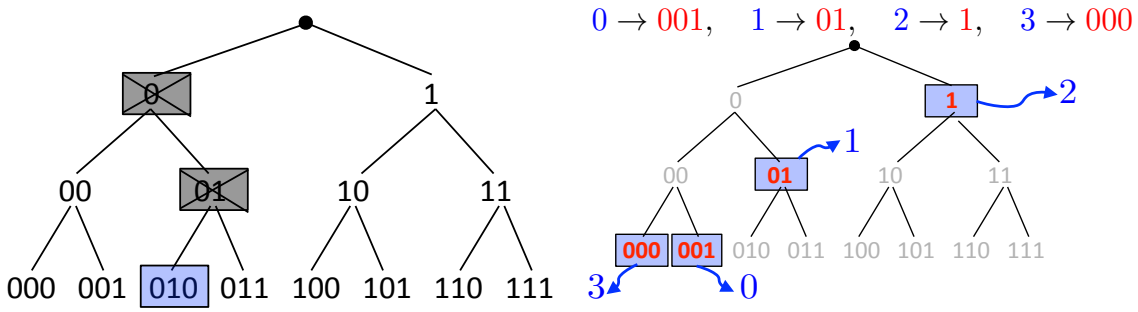


Figure 1.2: Left: complete tree of all codes of length 3; right: example of prefix code.

The prefix encodings are then represented as the leaves of the subtrees of this complete tree. Figure 1.2, top right, shows which subtree corresponds to the variable-length code

$$0 \mapsto 001, \quad 1 \mapsto 01, \quad 2 \mapsto 1, \quad 3 \mapsto 000.$$

Once a prefix encoding has been represented as a binary subtree, the decoding algorithm is particularly simple to implement. When decoding is begun, one moves to the root, and descends to each new bit read either to the left (for a 0) or to the right (for a 1). When one reaches a leaf of the subtree, one then sends the word of the code corresponding to this leaf, and one restarts to the root. The previous figure shows the decoding process.

[6] https://en.wikipedia.org/wiki/Prefix_Code

1.3 The Shannon Bound

After describing the coding techniques, we will now explain the Shannon theory, which analyzes the performance of these techniques (ie the number of bits needed for coding) by performing a random modeling of the message to be coded which is composed of a particular sequence of symbols).

1.3.1 Minimum Length Code and Random Modeling

The use of variable length prefix encoding shows that an average number of bits $\mathcal{L}$can be obtained than the number $\log_2(N)$ of bits obtained by a uniform code. The fundamental question, both on a theoretical and practical level, is whether we can find a prefix coding giving rise to a minimum number of bits per symbol.

This question is not correctly phrased, because its answer depends on the message to be coded, and this message is generally unknown a priori. A model is therefore needed to describe possible messages. The fundamental idea introduced by Claude Shannon is to use a probabilistic model: we do not know what messages we will have to code, but we assume that we know the probability of appearance of the symbols composing this message.

Shannon assumes that the symbols that make up the modeled message are drawn *independently*[7] according to a random variable V(the source of the message). This means that the symbols composing the modeled message are independent random variables with the same distribution as V.

1.3.2 Empirical Frequencies

In order to apply this probabilistic model to a given message, we will act as if we randomly draw each symbol one after the other according to probabilities identical to the frequencies observed (on average) in the case studied.

This means that we impose that the distribution of the V source to be equal to the empirical frequencies observed in the message. Empirical frequencies (p_0, p_1, p_2, p_3) are the frequency of appearance of the different symbols $(0, 1, 2, 3)$. For the set of the 25 pixels of the grayscale image

$$(0,1,3,2,0,3,2,2,1,2,2,1,1,2,2,2,1,1,2,2,2,1,1,2,1),$$

the frequency p_1 is equal to $9/25$ because the symbol 1 appears 9 times and it is desired to encode a sequence of 25 symbols. The list of empirical frequencies for this sequence of symbols is thus

$$p_0 = \tfrac{2}{25}, \quad p_1 = \tfrac{9}{25}, \quad p_2 = \tfrac{12}{25}, \quad p_3 = \tfrac{2}{25}.$$

The random modeling therefore imposes on the variable V to have for probability distribution (p_0, p_1, p_2, p_3), ie the probability that a symbol of the modeled message (assumed to be generated by the V source) is $\mathbb{P}(V = v) = p_v$.

This is an important example of modeling, which is of course not always relevant but allows a fine analysis of the problem. For example, in the case of an image, if a pixel is black, the next one is likely to be black, even if the overall black frequency is low. This defeats the independence hypothesis (the "Information Transformation" section details this example).

[7] https://en.wikipedia.org/wiki/Independencies_(probability)

1.3.3 Entropy

In order to answer the coding problem with a minimum average number of bits, Shannon introduced a fundamental mathematical object: *entropy*[8]. Entropy was invented by Ludwig Boltzmann[9] in the context of thermodynamics[10] and this concept was taken up by Claude Shannon to develop his theory of information. The entropy of the distribution of the source V is defined by the formula

$$\mathcal{H}_V \stackrel{\text{def.}}{=} -\sum_{v=0}^{N-1} p_v \times \log_2(p_v).$$

This formula means that we sum up for all possible v symbols the frequency of occurrence p_v of the symbol v multiplied by the logarithm $\log_2(p_v)$ of this frequency, then take the opposite (minus sign) of the number obtained.

As the logarithm is an increasing function, and as $\log_2(1) = 0$, we have $\log_2(p_v) \leqslant 0$ because p_v is always less than 1). The minus sign before the formula defining the entropy ensures that this quantity is always positive.

In our case, we have $N = 4$ values for the symbols, and we use the formula

$$\mathcal{H}_V \stackrel{\text{def.}}{=} -p_0 \times \log_2(p_0) - p_1 \times \log_2(p_1) - p_2 \times \log_2(p_2) - p_3 \times \log_2(p_3).$$

Note that if $p_v = 0$, then the convention $p_v \times \log_2(p_v) = 0 \times \log_2(0) = 0$. This convention means that null probabilities (ie, impossible events) are not taken into account in this formula. Moreover, it is consistent with the limit value of the function $x \mapsto x \ln(x)$ at $x = 0$.

The goal of entropy is to quantify the uncertainty on possible symbol sequences generated by the V source. We can show that the entropy verifies

$$0 \leqslant \mathcal{H}_V \leqslant \log_2(N).$$

The two extreme values thus correspond to respective minimum and maximum uncertainties.

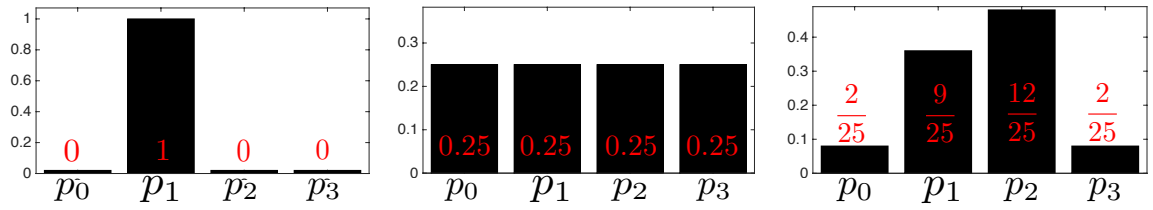


Figure 1.3: Three examples of probability distributions with corresponding entropies.

- **Minimal entropy** The entropy $\mathcal{H}_V = 0$ is minimal when the frequencies p_v are all null except one. The left-handed figure 1.3 shows the case where $p_1 = 1$ and all other probabilities are null.

[8] https://en.wikipedia.org/wiki/Entropy
[9] https://en.wikipedia.org/wiki/Ludwig_Boltzmann
[10] https://en.wikipedia.org/wiki/Entropie_(thermodynamics)

In this case,

$$\mathcal{H}_V = -0 \times \log_2(0) - 1 \times \log_2(1) - 0 \times \log_2(0) - 0 \times \log_2(0) = 0,$$

where we recall that $\log_2(1) = 0$ and that by convention we have $0 \times \log_2(0) = 0$. This corresponds to the modeling of a constant sequence of symbols, and the source will generate, for example, with probability 1 the following sequence of 25 symbols

$$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0).$$

- **Maximum entropy** On the other hand, $\mathcal{H}_V = \log_2(N)$ is maximal when all frequencies are equal, $p_v = 1/N$. In our case where $N = 4$, we have

$$\mathcal{H}_V = -\tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) - \tfrac{1}{4} \times \log_2(\tfrac{1}{4}) = \log_2(4) = 2,$$

where $\log_2(1/x) = -\log_2(x)$ is used and therefore in particular $\log_2(\frac{1}{4}) = -\log_2(4)$. The following figure 1.3, center, shows the histogram corresponding to this case.

This situation corresponds intuitively to the modeling of a sequence maximally *uncertain*. Here, for example, are two sequences of symbols generated by such a source V

$$(2,2,1,1,3,0,3,3,3,0,1,1,2,0,2,0,2,1,3,2,0,2,2,1,3),$$

$$(3,3,1,2,0,0,2,2,1,3,2,2,3,3,2,0,0,3,0,1,3,0,1,1,2).$$

- **Intermediate entropy** The intermediate situations between these two extremes correspond to intermediate entropies. For example, we can consider the distribution of the 25 pixels considered at the beginning of this article, which correspond to the message

$$(0,1,3,2,0,3,2,2,1,2,2,1,1,2,2,2,1,1,2,2,2,1,1,2,1).$$

For this distribution, we recall that we have the probabilities

$$p_0 = \tfrac{2}{25}, \quad p_1 = \tfrac{9}{25}, \quad p_2 = \tfrac{12}{25}, \quad p_3 = \tfrac{2}{25},$$

the figure 1.3, right, shows the histogram corresponding to these values.

The entropy then

$$\mathcal{H}_V = -\tfrac{2}{25} \times \log_2(\tfrac{2}{25}) - \tfrac{9}{25} \times \log_2(\tfrac{9}{25}) - \tfrac{12}{25} \times \log_2(\tfrac{12}{25}) - \tfrac{2}{25} \times \log_2(\tfrac{2}{25}) \approx 1.62,$$

which corresponds to a "intermediate" value of the entropy.

1.3.4 Average number of bits of a source

In the following, we denote c_v the code associated with a symbol v. The length (i.e. the number of bits) of each word c_v of code is denoted $L(c_v)$. For uniform coding, then the length is constant $L(c_v) = \log_2(N)$. On the other hand, if we take the example of variable coding

$$0 \mapsto c_0 \stackrel{\text{def.}}{=} 001, \quad 1 \mapsto c_1 \stackrel{\text{def.}}{=} 01, \quad 2 \mapsto c_2 \stackrel{\text{def.}}{=} 1, \quad 3 \mapsto c_3 \stackrel{\text{def.}}{=} 000,$$

then $L(c_0) = L(001) = 3$.

It can be seen that the average bit number $\mathcal{L}$ of the encoding of a message can be calculated using the empirical frequencies as follows:

$$\mathcal{L} = \sum_{v=0}^{N-1} p_v \times L(c_v).$$

This formula means that we sum up for all possible v symbols the frequency of occurrence p_v of the symbol multiplied by the length $L(c_v)$ of the code word c_v. For example, in our case, for $N = 4$, we have the formula

$$\mathcal{L} = p_0 \times L(c_0) + p_1 \times L(c_1) + p_2 \times L(c_2) + p_3 \times L(c_3).$$

As part of the random modeling using a V source, we will write $\mathcal{L}_v$ this average bit number, which is associated with the source V having the distribution $(p_v)_v$.

1.3.5 Shannon Bound for Coding

Claude Shannon showed in his article [20] that the entropy allows to bound the average number of bits $\mathcal{L}_v$ within the framework of this random model. He indeed showed that for any prefix encoding, one has

$$\mathcal{H}_V \leqslant \mathcal{L}_v.$$

This is a lower bound, and it says no prefix encoding can do better than this bound.

This result is fundamental because it describes an unbreakable limit, whatever the prefix encoding technique used. Its proof is too difficult to be exposed here, it uses the representation in the form of a tree detailed above in Section 1.2.6, one can look for example [14] for the details. This proof shows that it is necessary to spend on average at least $-\log_2(p_v)$ bits (which is, as we have already seen, always a positive number) to code a symbol v if one wants to have an effective coding. The most frequent symbols need fewer bits because p_v is smaller, so the optimal length $-\log_2(p_v)$ is also smaller. This is very natural, as can be seen in particular for the two extreme cases:

- **Minimal entropy** If $\mathcal{H}_V = 0$, then with probability 1, the sequence of symbols is composed of a single symbol. In this case, the use of a prefix encoding is very inefficient, since it must use at least one bit per symbol ie $\mathcal{L}_V \geqslant 1$, and thus such a coding is far from reaching the boundary of Shannon.

  The entropy being zero, the boundary says that one would wish to spend nothing for coding. This is logical, because there is no need to code such a sequence (since it is always the same).

- **Maximum entropy** If $\mathcal{H}_V = \log_2(N)$, then all symbols are equally probable, so we must use codewords of the same length for all symbols, which is obtained by a uniform code. As we saw above, such a code requires $\mathcal{L}_v = \log_2(N) = \mathcal{H}_v$ bits per symbol, and thus the lower bound of Shannon is tight in this case.

- **Intermediate entropy** In the case of the distribution of the 25 pixels considered at the beginning of this article, which correspond to the message

$$(0, 1, 3, 2, 0, 3, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1),$$

it is recalled that the entropy and the average number of bits, which have already been calculated, are respectively

$$\mathcal{H}_V \approx 1.62 \text{ bits} \quad \text{et} \quad \mathcal{L}_V = 1.68 \text{ bits}.$$

These values are in agreement with Shannon bound, and show that the prefix encoding used allows to be close enough to this bound.

We can ask whether this bound is precise, and whether it is possible to construct codes reaching the Shannon boundary in all cases (and not just the two extreme cases). Huffmann proposed in [13] a construction of an "optimal" encoding (ie having the average length $\mathcal{L}_V$ minimum for a given source V) using an elegant algorithm. The average length obtained by this coding satisfies

$$\mathcal{H}_V \leqslant \mathcal{L}_V \leqslant \mathcal{H}_V + 1.$$

The fact that this average length can be potentially as large as $\mathcal{H}_V + 1$(and therefore quite different from Shannon's lower bound $\mathcal{H}_v$) the length $L(c_{word})$ of a word c_v of the code is an integer, while the optimal length should be $-\log_2(p_v)$ which is not generally an integer. To overcome this problem, the symbols must be coded in groups, which can be done efficiently using Arithmetic Coding[11] [18], which reach the boundary of Shannon when we code an infinite sequence of symbols.

Shannon's theory thus makes it possible to bound the average coding length, which gives important information about the performance of a coding method for a given source. However, note that this does not give information on other potentially interesting statistical quantities, such as the maximum length or the median length.

1.3.6 Transformation of information

This entropy bound implicitely assumes that the symbols that make up the message to be coded are generated *independently* by the source V. This hypothesis allows a simple mathematical analysis of the problem, but it is generally false for complex data, as for example for the image shown in the following figure. Indeed, it is clear that the value of a pixel is not at all independent of those of its neighbors. For example, there are large homogeneous zones where the value of the pixels is quasi-constant.

In order to improve the coding performance, and to obtain effective image compression methods, it is crucial to retransform the sequence of symbols in order to reduce its entropy by exploiting the dependencies between the pixels. A simple transformation to do this involves replacing the p pixels $(v_i)_{i=1}^{P}$ with those of their differences $(d_i \overset{\text{def.}}{=} v_i - v_{i-1})_{i=1}^{P-1}$. Indeed, in a uniform zone, the successive differences will be zero because the pixels have the same value. Figure 1.4 shows how to perform such a calculation. It also shows that this transformation is bijective, that is to say that one can return to the original values $(v_i)_i$ by carrying out a gradual summation of the differences, that is to say calculating

$$v_i = v_0 + \sum_{j=1}^{i} d_j.$$

In order to make this inversion, it is of course necessary to retain the value v_0 of the first pixel. The bijectivity of the transformation

$$(v_0, \ldots, v_{P-1}) \longmapsto (v_0, d_1, \ldots, d_{P-1})$$

[11] https://en.wikipedia.org/wiki/ArithmeticCoding

is crucial for decoding and displaying the decoded image.

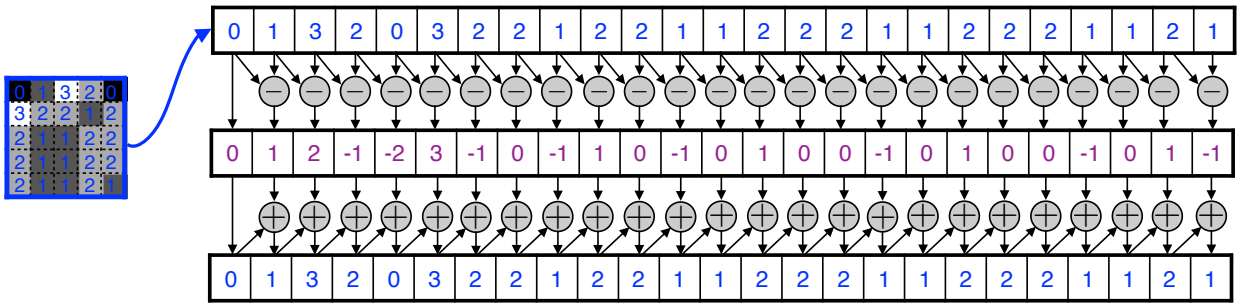
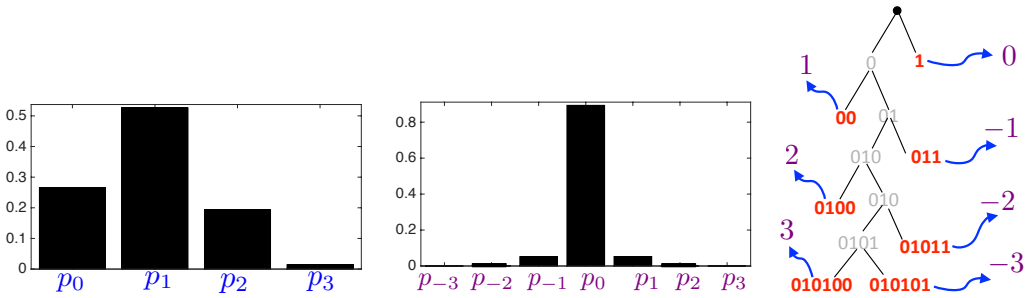


Figure 1.4: Difference representation

As the pixels can take values $\{0, 1, 2, 3\}$, the differences can take the values $\{-3, \ldots, 3\}$. In particular, they may be negative (which does not pose any particular problem for defining a coding). The following figure compares the histograms of pixels and differences. We notice that the histogram of the differences is much more "peaked" in the neighborhood of 0, which is logical, because many differences (corresponding to the homogeneous zones) are null or small. The entropy $\mathcal{H}_D$ of the histogram of the differences (which can be modeled with a source D) is therefore much lower than the entropy $\mathcal{H}v$ of the pixels.

The figure 1.5 shows a comparison of the histograms of pixel values and differences, calculated over the entire image (and not only on the initial 25 pixel subset). It also shows the tree of an optimal prefix encoding (computed by the Huffman [13] algorithm) associated with the histogram of the differences.



$\mathcal{H}_v \approx 1.54, \mathcal{L} \approx 1.67$ $\mathcal{H}_D \approx 0.61, \mathcal{L} \approx 1.16$ Coding Tree

Figure 1.5: Comparison of histograms of pixel values and differences, and a code tree for these differences.

This tree corresponds to the coding

$$-3 \mapsto 010101, -2 \mapsto 01011, -1 \mapsto 011, 0 \mapsto 1, 1 \mapsto 00, 2 \mapsto 0100, 3 \mapsto 010100.$$

This encoding has an average length $\mathcal{L} \approx 1.16$ bits. This average number matches well to the entropy bound and is significantly smaller than the mean length associated with the pixel histogram (1.67 bits), which is itself smaller than the average length associated with a ($\log_2(4) = 2$ bits). If encode

the entire image of 256×256 in gray level, we get the following gains, where 1kb=8×1024 bits is a *kilo byte*.

Uniform encoding		Variable encoding		Coding of differences
16.3 kb	$\longrightarrow$	13.7 kb	$\longrightarrow$	9.5 kb

The most efficient methods of image compression use more complex transformations, and exploit in a finer way the local regularity of the images. This is the case of the JPEG-2000[12] compression method, which is considered to be the most efficient at the moment, using the wavelets[13], see the book [14] for more details. There are many other cases where non-independence of symbols can be used to improve coding performance. An important example is the sequence of letters that compose a text.

1.4 Conclusion

The mathematical theory initiated by Claude Shannon defines a framework necessary for the development of effective techniques for the acquisition, processing, storage and transmission of data in digital form. These techniques that revolutionized communications and computing during the second half of the 20th century, and enabled the growth of the Internet at the beginning of the 21st century. Without the revolutionary contributions of Shannon, you could not go on vacation with your entire library in your electronic reader, and all episodes of *Game of Thrones* on your tablet!

For more details on the theory of information, one can have look at [7], for its use in signal and image processing, one can look at [14]. The computer codes used to reproduce the figures in this article are available online at[14], and other codes are available on the site `www.numerical-tours.com` [17].

Glossary

– **Pixel**: location on the square grid of an image, sometimes used to refer to the associated value.
– **Symbol**: element v of a finite set, for example $\{0, \ldots, N-1\}$.
– **Code**: 0 and 1 sequence used to encode a v symbol.
– **Coding**: set of correspondences between v symbols and associated codes, for example $2 \mapsto 10$. Also refers to the action of replacing a sequence of symbols with a set of bits.
– **Empirical distribution**: frequency p_v of appearance of symbols v in the sequence of symbols to be coded.
– **Histogram**: graphical representation of the empirical distribution, which can also by extension designate this distribution.
– **Source**: random variable V modeling the symbols, with the distribution $\mathbb{P}(V = v) = p_v$.
– **Entropy**: $\mathcal{H}_v$ is a positive number associated with the source V and depends on its probability distribution $(p_v)_v$.
– **Number of mean bits of a sequence:** $\mathcal{L}$ is associated with the encoding of a sequence of symbols.

[12] `https://fr.wikipedia.org/wiki/JPEG_2000`
[13] `https://en.wikipedia.org/wiki/Ondelette`
[14] `https://github.com/gpeyre/2016-shannon-theory`

– Number of source mean bits: $\mathcal{L}_v$ is associated with the encoding of symbols generated by V.

Acknowledgments

I thank Marie-Noëlle Peyré, Gwenn Guichaoua, François Béguin, Gérard Grancher, Aurélien Djament and François Sauvageot for their careful proofreading of a French version of this text.

The image of the flower is due to Maitine Bergounioux. The image of Shannon used for the logo of the article is due to the telehistoriska user of the flickr site (under license CC-BY-NC-2.0).

Chapter 2

Image Processing

Digital cameras take precise photographs of the world around us. The user wants to be able to store his photos on his hard drive with minimum memory requirement. He also wishes to be able to reprocess them in order to improve their quality. This article presents the mathematical and computer tools used to perform these different tasks.

2.1 The pixels of an image

A digital image in gray levels is an array of values. Each box of this table, which stores a value, is called a pixel. By noting n the number of rows and p the number of columns in the image, we manipulate an array of $n \times p$ pixels. Figure 2.1, left, shows a visualization of a square table with $n = p = 240$, which represents $240 \times 240 = 57600$ pixels. The digital cameras can record much larger images, with several millions of pixels.

The values of the pixels are stored in a computer or a digital camera in the form of relative integers between entre 0 et $255 = 2^8 - 1$, making 256 possible values for each pixel. The value 0 is black, and the value 255 is white. The intermediate values correspond to gray levels ranging from black to white. Figure 2.1 shows a subset of 6×6 pixels taken from the previous image. You can see both the values that make up the table and the gray levels that allow you to display the image on the screen.

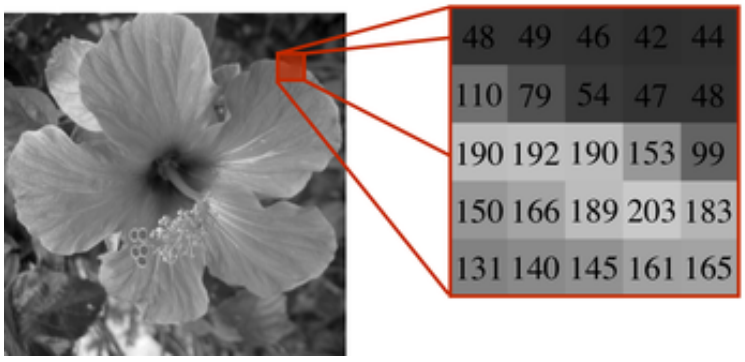


Figure 2.1: *Sub-image of size* 5×5

2.2 Image Storage

2.2.1 Binary Codes

Storing large images on the hard drive of a computer takes a significant amount of places. Integer numbers are stored in binary, in the form of a succession of 0 and 1. Each 0 and each 1 corresponds to an elementary unit of information, called bit. To obtain the binary expression of a pixel having the value 179, it is necessary to decompose this value as a sum of powers of two. We thus obtain

$$179 = 2^7 + 2^5 + 2^4 + 2 + 1,$$

where care has been taken to order the powers of two in decaying order. In order to make the binary more explicit, we add "$1\times$" before each power that appears in the expression, and "$0\times$" before the powers that do not appear

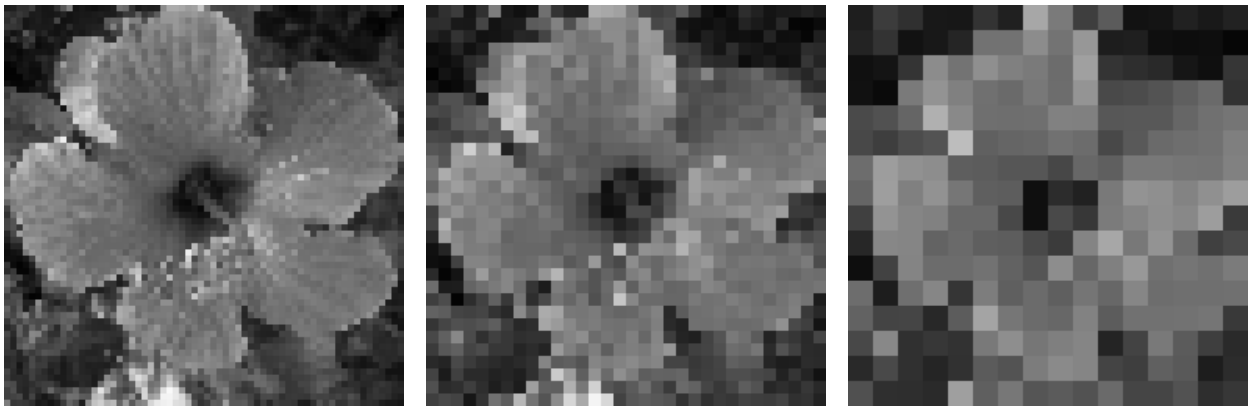
$$179 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0.$$

Using such a decomposition, the value of each pixel, which is a number between 0 and 255, requires $\log_2(256) = 8$bits. The binary writing of the value 179 of the pixel is thus $(1, 0, 1, 1, 0, 0, 1, 1)$. Any value between 0 and 255 can be written in this way, which requires the use of 8 bits. Indeed, there are 256 possible values, and $256 = 2^8$. To store the complete image, it is therefore necessary to use $n \times p \times 8$bits. For the image shown in the previous figures, it is thus necessary to use

$$256 \times 256 \times 8 = 524288\text{bits}.$$

Equivalently, this image requires 57.6kb (kilobytes), since a kilobyte is equal to 8 bits.

2.2.2 Sub-sampling an Image



One row / column of 4 | One row / column out of 8 | One row / column of 16

Figure 2.2: *Subsampling of an image*

In order to reduce the required storage space of an image, the number of pixels can be reduced. The easiest way to do this is to delete rows and columns in the original image. Figure 2.2, at the top left, shows what is obtained if one row is kept out of 4 and one column out of 4. We thus have divided by $4 \times 4 = 16$ the number of pixels of the image, and thus also divided by 16 the number of bits required to store the image on a hard disc. In fig. 2.2, one can see the results obtained by removing more and more rows and columns. Of course, the quality of the image degrades quickly.

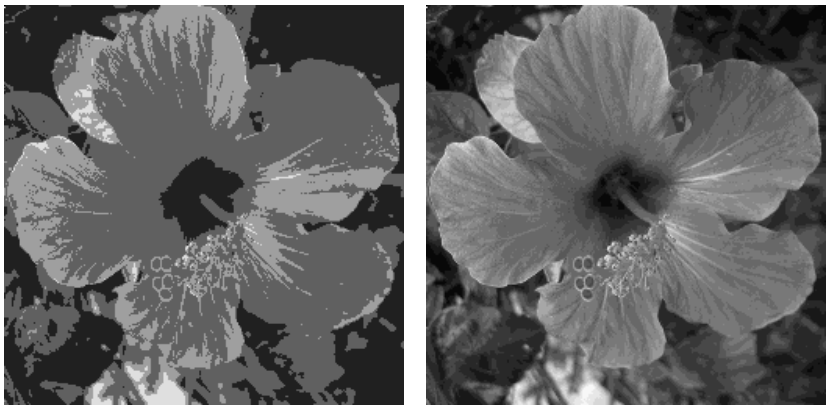
2.2.3 Quantizing an image

Another way to reduce the memory space required for storage is to use fewer integers for each value. For example, we can use only integers between 0 and 3, which will give an image with only 4 levels of gray. One can convert the original image to an image with 4 levels of values by performing the replacements:

– the values in $0, 1, \ldots, 63$ are replaced by the value 0 (black),
– the values in $64, 1, \ldots, 127$ are replaced by the value 1 (light gray), The values in $128.1, \ldots, 191$ are replaced by the value 2 (dark gray),
– the values in $192, \ldots, 255$ are replaced by the value 3 (white).

Such an operation is called quantization. Figure 2.3, in the center, shows the resulting image with 4 grayscale levels.

We have already seen that we can represent any value between 0 and 255 using 8 bits using binary coding. In the same way, one can check that any value between 0 and 3 can be represented using 2 bits. This thus results in a reduction of a factor 8/2=4 of the memory footprint necessary for storing the image on a hard disk. Figure 2.3 shows the results obtained using less and less gray levels.



4 niveaux de gris 16 niveaux de gris

Figure 2.3: *Quantizing an image*

As with the reduction of the number of pixels, the reduction of the number gray levels greatly affects the quality of the image. In order to minimize the size of an image without changing its quality, more complex methods of image compression. The most effective method is JPEG-2000. It uses the theory of wavelets.

2.3 Noise Removal

2.3.1 Local Averaging

Images are sometimes of poor quality. A typical example of a defect is the noise which appears when a picture is under-exposed, that is if there is not enough light. This noise corresponds to small random fluctuations of the gray levels. Figure 2.5, on the left, shows such a noisy image.

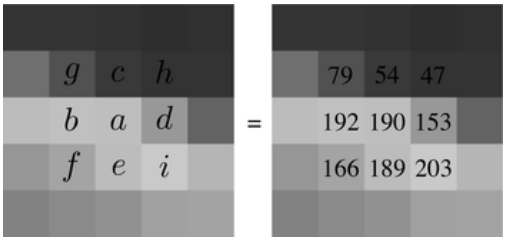


Figure 2.4: *Pixels neighborhood.*

In order to remove the noise in the images, it is necessary to modify the pixel values. The simplest operation is to replace the value a of each pixel by the average of a and the values b, c, d, e, f, g, h, i of the 8 pixels that surround a. Figure 2.4 shows an example of a neighborhood of 9 pixels. A modified image is thus obtained by replacing a by

$$\frac{a+b+c+d+e+f+g+h+i}{9},$$

since the average of 9 values is averaged. In our example, this average is

$$\frac{190+192+79+54+47+153+203+189+166}{9} \approx 141.$$

By performing this operation for each pixel, a large part of the noise is removed, because this noise is made up of random fluctuations, which are decreased by averaging. Figure 2.5, top left, shows the effect of such a calculation. All the noise was not removed by this operation. In order to remove more of noise, one can average more values around each pixel. Figure 2.5 shows the result obtained by increasing the average of values.

Mean on 9 pixels | Mean on 25 pixels | Mean on 49 pixels

Figure 2.5: *Mean with increasing width*

Pixel averaging is very effective in removing noise in images, unfortunately it also destroys much of the the information of the image. It can indeed be seen that the images obtained by averaging are blurry. This is particularly visible near contours, which are not sharp.

2.3.2 Local Median

To reduce this blur, the mean can be replaced by the median. In the example of the neighborhood of 9 pixels used in the previous section, the 9 sorted values are:

$$47, 54, 79, 153, 166, 189, 190, 192, 203.$$

The median of these nine values is 166. In order to remove more noise, it is enough to calculate the median over a larger number of neighboring pixels, as shown in Figure 2.6. One can observe that this method is more efficient than the mean calculation because the resulting images are less blurry. However, just as with the calculation of averages, if we take neighborhoods that are too large, we lose also information of the image, especially the edges of the objects are degraded.

Median on 9 pixels | Median on 25 pixels | Median on 49 pixels

Figure 2.6: *Median filtering with increasing windowing size.*

2.4 Detecting Edges of Objects

In order to locate objects in the images, it is necessary to detect their edges. These edges correspond to areas of the image where pixel values change rapidly. It is the case for example when passing from the boat (which is dark, ie. with small values) to the sea (which is clear, therefore with large values).

In order to know if a pixel with a value a is along an edge of an object, the values b, c, d, e of its four neighbors are taken into account, which have a common side with it (Figure 2.7). This allows the edges of objects to be detected as accurately as possible.

A value ℓ is calculated according to the formula

$$\ell = \sqrt{(b-d)^2 + (c-e)^2}.$$

In our example, we thus obtain

$$\ell = \sqrt{(192-153)^2 + (189-54)^2} = \sqrt{19746} \approx 141.$$

It may be noted that if $\ell = 0$, then $b = c$ and $d = e$. On the contrary, if ℓ is large, this means that the neighboring pixels have very high values different, the pixel considered is therefore probably on the edge of an object.

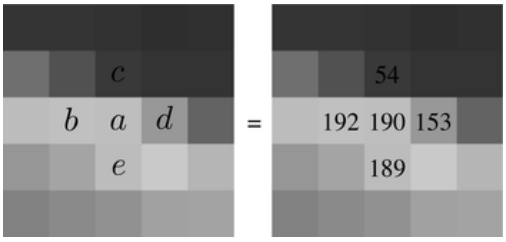


Figure 2.7: *Example of a neighborhood of 5 pixels.*

Figure 2.8 shows an image whose pixel value is $\min(\ell, 255)$. It is necessary to take the minium with 255, because the value of ℓ can exceed the maximum displayable value (255, which corresponds to white). These values are displayed with black when $\ell = 0$, in white when ℓ is high, and gray levels are used for the intermediate values. It can be seen that in the image on the right, the outlines of the objects appear white, as they correspond to large values of ℓ.

Original Image Contour map ℓ

Figure 2.8: *Edge detection.*

2.5 Color Images

2.5.1 RGB Space

A color image is actually composed of three independent images, in order to represent the red, green, and blue. Each of these three images is called a color channel. This representation in red, green and blue mimics the human visual system. Figure 2.10 shows the three constituent channels of the image shown on the left of Figure 2.9.

Each pixel of the color image thus contains three numbers (r, v, b), each being an integer between 0 and 255. If the pixel is equal to $(r, v, b) = (255, 0, 0)$, it contains only information red, and is displayed as red. Similarly, the pixels of $(0, 255, 0)$ and $(0, 0, 255)$ are respectively displayed green and blue.

A color image can be displayed on the screen. from its three channels (r, v, b) using the rules of additive color synthesis. These rules correspond to the way in which light rays combine, hence the

Original image Luminance

Figure 2.9: *Color image.*

Red canal Green canal Blue canal

Figure 2.10: *Color channels*

qualifier "additive". Figure 2.11, left, shows the composition rules this additive synthesis of colors. For example, a pixel with the values $(r, v, b) = (255, 0, 255)$ is a mixture of red and green, displayed as yellow.

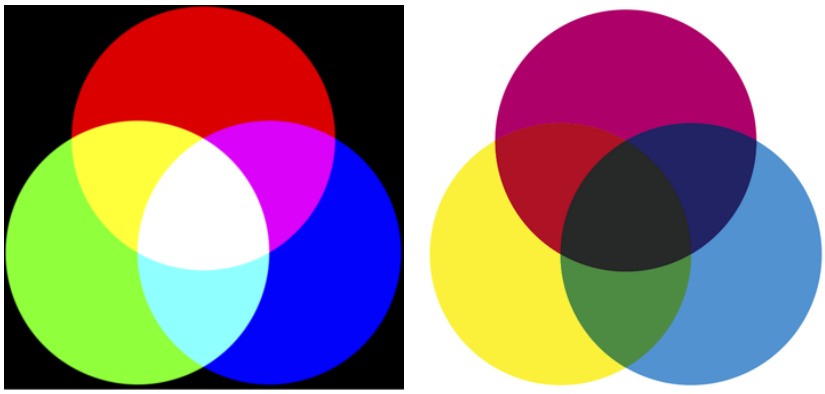
2.5.2 CMJ Space

Another common representation for color images uses as background colors cyan, magenta and yellow. It is calculated the three numbers (c, m, j) corresponding to each of these three channels from the red, green and blue channels (r, v, b) as follows

$$c = 255 - r, \quad m = 255 - v, \quad j = 255 - b.$$

For example, a pixel of pure blue $(r, v, b) = (0, 0, 255)$ will become $(c, m, j) = (255, 255, 0)$. Figure 2.12 shows the three channels (c, m, j) of a color image.

In order to display a color image on the screen from the three channels (c, m, j), the subtractive color synthesis must be used. Figure 2.11, right, shows the composition rules of this subtractive



Additive synthesis Subtractive synthesis

Figure 2.11: *Color Synthesis*

Cyan canal Magenta canal Yellow canal

Figure 2.12: *Channels CMJ*

synthesis. They correspond in painting to the absorption of light by colored pigments, hence the qualifier "subtractive". Cyan, magenta and yellow are called primary colors.

It is thus possible to store on a hard disk a color image by storing the three channels, corresponding to the (r, g, b) or (c, m, j). One can change color images in a similar way as graylevel image, by changing each channel.

2.6 Changing the Contrast of an Image

2.6.1 Luminance

One calculates a grayscale image from a color image as the mean of the three channels. Thus, for each pixel, a value

$$a = \frac{r + v + b}{3}$$

is computed which is called luminance of the color. Figure 2.9 shows the transition from a color image to a luminance image in grayscales.

2.6.2 Grayscale contrast manipulations

It is possible to make various changes to the image in order to modify his contrast. We consider here a grayscale image. A simple manipulation consists in replacing each value a of a pixel of an image by $255 - a$, which corresponds to the opposite gray intensity. The white becomes black and vice-et-versa, giving a similar effect to that of the negative of film for cameras, see figure 2.13, left.

The image is lightened or darkened using an increasing function from $[0, 255]$ to itself, which is applied to the a values of the pixels. One can darken the image by using the square function. More precisely, we define the new value of a pixel of the image as $a^2/255$ (see figure 2.13 in the center). Since the result is not generally an integer, it is rounded to the nearest integer. Similarly, for lightening the image, the value a of each pixel is replaced by the rounding of $\sqrt{255a}$. Figure 2.13, on the right, shows the obtained result. It will be noted that these two operations (square lightening and square root darkening) are inverse to one another.

Negative Square Square root

Figure 2.13: *Changing the contrast.*

2.6.3 Manipulations of Color Contrast

In order to manipulate the contrast of a color image, it is important to respect the color tones as much as possible. It is therefore simpler to manipulate only the luminance component $a = (r + v + b)/3$, while maintaining the residue $(r - a, v - a, b - a)$ constant. For example, a change in contrast can be defined by raising the luminance a to the $ga > 0$ in order to obtain

$$\tilde{a} = 255 \times \left(\frac{a}{255}\right)^{\gamma} = 255 \times \exp\left(\gamma \times \ln\left(\frac{a}{255}\right)\right),$$

(with Convention $\tilde{a} = 0$ when $a = 0$). It is noted that for $\gamma = 1/2$ (respectively $\gamma = 2$) the contrast change is found by squaring (respectively square root) introduced in the previous section. And of course, for $\gamma = 1$, the luminance is unchanged.

This change in contrast is then reflected on the color image by defining three channels $(\tilde{r}, \tilde{v}, \tilde{b})$ of a new image by

$$\begin{cases} \tilde{r} = \max(0, \min(255, r + \tilde{a} - a)), \\ \tilde{v} = \max(0, \min(255, v + \tilde{a} - a)), \\ \tilde{b} = \max(0, \min(255, b + \tilde{a} - a)). \end{cases}$$

It is important to take the maximum with 0 and the minimum with 255 so that the result remains in the range [0.255], and is displayed correctly. Figure 2.14 shows the result for different values of γ. For $\gamma < 1$, the image looks clearer, while for $\gamma > 1$, the image is darkened.

$\gamma = 0,5$ $\gamma = 0,75$ $\gamma = 1$ $\gamma = 1,5$ $\gamma = 2$ $\gamma = 3$

Figure 2.14: *Changing the contrast of a color image.*

2.7 Images and Matrices

2.7.1 Symmetry and Rotation

An image is an array of numbers, with n lines and p columns. It is therefore easy to perform some geometric transformations on the image. The values of the pixels that make up this table (denoted A) can be represented as $A = (a_{i,j})_{i,j}$ or index i describes the set of numbers $\{1, \ldots, n\}$ (the integers between 1 and n) and the index j the numbers $\{1, \ldots, p\}$. $a_{i,j}$ is said to be the value of the pixel at position (i, j).

The array of pixels thus indexed is represented as

$$A = \begin{pmatrix} a_{1,1} & & & & a_{1,p} \\ & & \vdots & & \\ & & a_{i-1,j} & & \\ \ldots & a_{i,j-1} & a_{i,j} & a_{i,j+1} & \ldots \\ & & a_{i+1,j} & & \\ & & \vdots & & \\ a_{n,1} & & & & a_{n,p} \end{pmatrix},$$

This corresponds to the representation of the image in the form of a matrix. Transposing this matrix corresponds to symmetry with respect to the main diagonal. This transposition is carried out on each of the three color components (see figure 2.15, on the left).

It is also possible to carry out a rotation by a quarter turn clockwise on the image. This is obtained by defining a matrix $C = (c_{i,j})_{j,i}$ of p lines and n columns by $c_{j,i} = a_{n-i+1,j}$. Figure 2.15, right, shows the action of this rotation on an image.

2.7.2 Interpolation Between Two Images

It is possible to carry out a transition between two images A and B. It is therefore assumed that the two images have the same number n of lines and the same number p of columns. $A = (a_{i,j})_{i,j}$ the pixels of image A and $B = (b_{i,j})_{i,j}$ the pixels of the image B.

Matrice A | Matrix B (transposed) | Matrix C (rotation)

Figure 2.15: *Transpose and rotate.*

For a value t set between 0 and 1, the image $C = (c_{i,j})_{i,j}$ is defined as

$$c_{i,j} = (1-t)a_{i,j} + tb_{i,j}.$$

It is the formula of a linear interpolation between the two images. For a color image, this formula is applied to each of the channels R, V and B.

It can be seen that for $t = 0$, the image C is equal to the image A. For $t = 1$, the image C is equal to the image B. When the value t increases from 0 to 1, one thus obtains a fading effect, since the image, which at first is close to image A resembles more and more the image B. Figure 2.16 shows the result obtained for 6 values of t distributed between 0 and 1.

Image A, t=0 | t=0.2 | t=0.4 | t=0.6 | t=0.8 | Image B, t=1

Figure 2.16: *Linear interpolation.*

Conclusion

Mathematical processing of images is a very active field, where the theoretical advances are obtained using fast computational algorithms. These algorithms have important applications for the manipulation of digital contents. This article, however, only scratched the surface of the immense list of treatments that can be subjected to an image. We refer to the website *A Numerical Tour of Signal Processing*[1] for many more examples of image processing and links to other resources available online.

[1] http://www.numerical-tours.com/

Glossary

- **random**: unpredictable value, such as noise disturbing images of bad qualities.
- **Bit**: a basic unit for storing information in the form of 0 and 1 in a computer.
- **Channel**: one of three elementary images that make up a color image.
- **Edges**: the area of an image where the values of the pixels change rapidely, which corresponds to the contours of the objects that make up the image.
- **Noise**: small perturbations that degrade the quality of an image.
- **Square**: the square b of a value a is $a \times a$. It is noted a^2.
- **Contrast**: informal quantity that indicates how much difference there is between light and dark areas of an image.
- **Image compression**: a method to reduce the amount of memory required to store an image on the hard disk.
- **Binary coding**: writing of numeric values using only 0 and 1.
- **Blur**: degradation of an image that makes the contours of objects unclear, and therefore difficult to locate precisely.
- **Fade**: linear interpolation between two images.
- **Color image**: a set of three grayscale images, which can be displayed on a color screen.
- **Digital image**: an array of values that can be displayed on the screen by assigning a gray level to each value.
- **Inverse**: operation that returns an image to its original state.
- **JPEG-2000**: recent image compression method that uses a wavelet transform.
- **Luminance**: average of the different channels in an image, which indicates the light output of the pixel.
- **Matrix**: array of values, represented as $(a_{i,j})_{i,j}$.
- **Median**: central value when sorting a set of values.
- **Average**: the average of a set of values is their sum divided by their number.
- **Grayscale**: grayscale used to display a digital image on the screen.
- **integers**: numbers 0, 1, 2, 3, 4 ...
- **Byte**: set of eight consecutive bits.
- **Wavelets**: image transformation that is used by image compression JPEG-2000 method.
- **Ascending order**: classifying a set of values from the smallest to the largest.
- **Pixel**: a single element in the array of values that corresponds to a digital image.
- **Quantization**: a method to reduce the set of possible values of a digital image.
- **square root**: the square root b of a positive value a is the positive value b such that $a = b \times b$. It is denoted $\sqrt{a}$.
- **Resolution**: the size of an image (number of pixels).
- **Exposed**: photograph of a scene too dark for which the photographic lens did not stay open long enough.
- **Additive synthesis**: rule to construct any color from the three colors red, green and blue. This is the rule that governs the mixing of the colors of light beams when illuminating a white wall.
- **Subtractive synthesis**: rule to construct any color from the three cyan, magenta and yellow colors. This is the rule that governs the mixing of colors in paint.

Chapter 3

Sparsity, Inverse Problems and Compressed Sensing

Current standards for compressing music, image or video (MP3, JPG, or MPEG) all use methods derived from non-linear approximation. These methods compute an approximation of the initial data using a linear combination of a small number of elementary functions (such as sinusoids or wavelets). These methods, initially used for approximation, denoising or compression, have been applied more recently to more difficult problems, such as increasing the resolution or inversion of operators in medical imaging. These extensions require the resolution of large-scale optimization problems, and are the subject of intense research activity. One of the most recent advances in this field, compressed sampling, uses the theory of random matrices in order to obtain theoretical guarantees for the performance of these techniques. Compressed sampling allows Claude Shannon's theory of sampling and compression to be considered from a new angle. The compressibility of the data allows for simultaneous sampling and compression.

This chapter presents the key mathematical concepts that have allowed evolution from classical Shannon sampling to compressed sampling. The notion of sparse decomposition, which makes it possible to formalize the idea of compressibility of information, is the main thread.

3.1 Traditional Sampling

In the digital world, most data (sound, image, video, etc.) are discretized in order to store, transmit and modify them. From an analog signal, which is represented by a continuous function $s \mapsto \tilde{f}(s)$, the measurement device calculates a set of discretized values $f = (f_q)_{q=1}^{Q} \in \mathbb{R}^Q$. Thus, Q is the number of time samples for an audio piece or the number of pixels for an image. Figure 3.1 shows examples of discretized data. In the case of an image, $\tilde{f}(s)$ represents the amount of light arriving at a point $s \in \mathbb{R}^2$ of the camera's focal plane, and $f_q = \int_{c_q} \tilde{f}(s)\mathrm{d}s$ is the total amount of light illuminating the c_q surface of a CCD sensor indexed by q. For simplicity, here we assume scalar data (eg mono sound, grayscale image, or video), but the techniques described here may extend to vector data (stereo sound, color image).

It is the theory developed by Claude Shannon [20] that laid the foundation for sampling (the use of a discrete vector f to faithfully represent a continuous function $\tilde{f}$) but also those of lossless compression. We will see how current research has made it possible to build on these foundations lossy compression methods (i.e with a slight degradation of the quality), as well as to revisit the

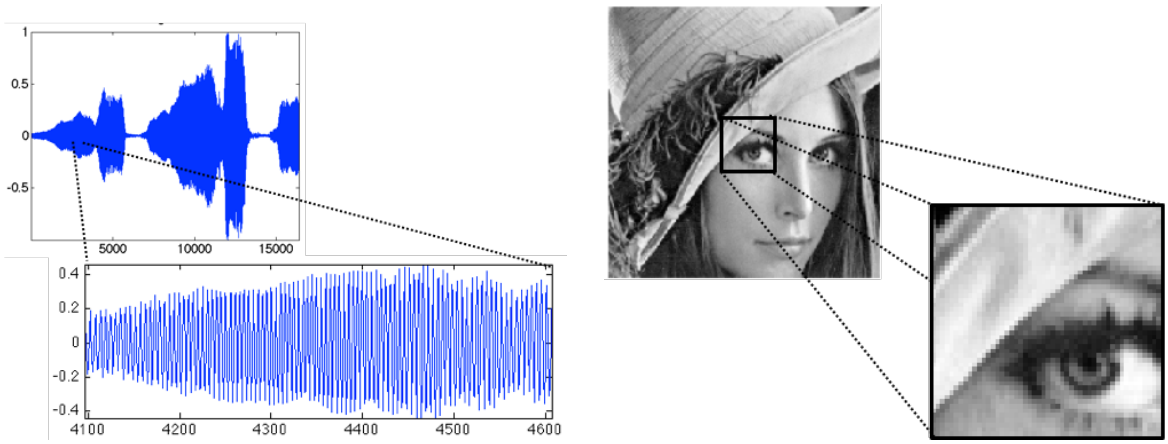


Figure 3.1: Examples of a sound signal (1D data) and an image (2D data) discretized.

conventional sampling to give rise to the idea of compressed sampling.

3.2 Nonlinear Approximation and Compression

3.2.1 Nonlinear Approximation

The size Q of these data is generally very large (of the order of million for an image, of the billion for a video) and it is necessary to calculate a more economical representation in order to be able to store f or to transmit it on a network. All modern lossy compression methods (MP3, JPEG, MPEG, etc.) use sparse decompositions (that is composed of few non-zero coefficients) in a dictionary $\Psi = (\psi_n)_{n=1}^N$ composed of elemental atoms $\psi_n \in \mathbb{R}^Q$. It is thus sought to approach f with the aid of a linear combination

$$f \approx \Psi x \stackrel{\text{def.}}{=} \sum_{n=1}^{N} x_n \psi_n \in \mathbb{R}^Q$$

where the $x = (x_n)_{n=1}^N \in \mathbb{R}^N$ are the coefficients that will be stored or transmitted. In order for this representation to be economical, and for storage to take up little space, it is necessary that a maximum of coefficients x_n be zero, so that only the non-zero coefficients have to be stored. Given a budget $M > 0$ of non-zero coefficients, the best possible combination is sought in order to approximate ℓ^2 the initial data. The aim is to solve the optimization problem

$$x^\star \in \operatorname*{argmin}_{x \in \mathbb{R}^N} \{\|f - \Psi x\|_2 \; ; \; \|x\|_0 \leqslant M\} \quad \text{where} \quad \|f\|_2^2 \stackrel{\text{def.}}{=} \sum_{q=1}^{Q} |f_q|^2. \tag{3.1}$$

Here we have noted $\|x\|_0 \stackrel{\text{def.}}{=} \sharp\{n \; ; \; x_n \neq 0\}$ the number of non-zero coefficients of x, which is a counting measure often referred to by language abuse as the "pseudo-norm" ℓ^0 (which is not a standard!). This abuse of language will be explained in section 3.3, see in particular figure 3.4.

The problem (3.1) is in general impossible to solve: it is a combinatorial problem, which, without further hypothesis on Ψ, requires the exploration of all combinations of M coefficients non-zero. It has been proved that this problem is indeed NP-difficult [16].

f $\Psi x^\star, M = N/4$ $\Psi x^\star, M = N/8$ $\Psi x^\star, M = N/16$

Figure 3.2: Approximate Examples $f \approx \Psi x^\star$ with $M = \|x^\star\|_0$ which varies, for a $f \in \mathbb{R}^N$ image of $N = 256^2$ pixels.

3.2.2 Approximation in an orthonormal basis

There is however a simple case, which is very useful for compression: this is the case where Ψ is an orthonormal basis of $\mathbb{R}^Q$, ie $Q = N$ and

$$\langle \psi_n, \psi_{n'} \rangle = \begin{cases} 1 & \text{si} \quad n = n', \\ 0 & \text{sinon.} \end{cases} \qquad \text{where} \quad \langle f, g \rangle \overset{\text{def.}}{=} \sum_{q=1}^{Q} f_q g_q.$$

This case is the one most often encountered for data compression, using for example discrete Fourier orthogonal bases, local cosines (used for MP3, JPG and MPG) and wavelets (used for JPEG2000), see the book [14]. In this case, we have the identity of Parseval which corresponds to the decomposition of f in an orthonormal basis

$$f = \sum_{n=1}^{N} \langle f, \psi_n \rangle \psi_n \quad \text{et} \quad \|f - \Psi x\|_2^2 = \sum_{n=1}^{N} |\langle f, \psi_n \rangle - x_n|^2. \tag{3.2}$$

These formulas show that the solution of (3.1) is very simple to calculate. Indeed, to minimize $\|f - \Psi x\|_2$, for each non-zero x_n, one should choose $x_n = \langle f, \psi_n \rangle$. And since we set a maximum budget of M non-zero coefficients, we must choose the M largest coefficients $|\langle f, \psi_n \rangle|$ in the formula (3.2). Mathematically, if we note $|\langle f, \psi_{n_1} \rangle| \geqslant |\langle f, \psi_{n_2} \rangle| \geqslant \ldots$ a sorting of the coefficients in descending order, then a $x^\star$ solution of (3.1) is given by

$$x_n^\star = \begin{cases} \langle f, \psi_n \rangle & \text{si} \quad \in \{n_1, \ldots, n_M\}, \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The figure 3.2 shows approximations $f \approx \Psi x^\star$, with a variable number $M = \|x^\star\|_0$ of coefficients. These approximations are performed using an orthogonal base of wavelets Ψ, called the Daubechies 4 base, which are similar to the functions used in the JPEG2000 image compression standard, and are popular because there is a fast algorithm for calculate the scalar products $(\langle f, \psi_n \rangle)_n$ with a computation time proportional to Q (see the book [14, Chapter 7] for a complete description of the theory of wavelets). It can be seen that the quality of the reconstructed image $\Psi x^\star$ degrades when M decreases, but one can still considerably reduce the amount of information to be stored (the M/Q compression ratio is small), while maintaining an acceptable visual quality. This fundamental observation corresponds to the fact (observed in practice) that natural images are very well

Image originale f Φf (flou) Φf (masquage)

Figure 3.3: Observation (noiseless, $w = 0$) $y = \Phi f$ in the case of a convolution ($\Phi f = \varphi \star f$ is a convolution against a low pass filter φ) and missing data ($\Phi = \mathrm{diag}(\mu_q)_{q=1}^{Q}$ is a masking operator).

approximated by a "sparse" linear combination of the form $\Psi x^\star$ with $\|x^\star\|_0 \leqslant M$. It is important to note that, although the calculation of $\Psi x^\star$ from $x^\star$ is a linear formula, the calculation of $x^\star$ from f is *non-linear*, as can be seen in the formula (3.3). The transition from f to its approximation $\Psi x^\star$ is called a non-linear approximation. The theoretical justification of this observation is the object of the study of nonlinear approximation theory, which seeks to prove that $\|f - \Psi x^\star\|$ decreases rapidly when M increases under certain assumptions of regularity over f, for instance assuming that the image is piecewise smooth, see [14, Chap. 9].

In order to obtain an complete compression algorithm, it is then necessary to use a technique making it possible to convert the M coefficients $(x_{n_1}, \ldots, x_{n_2})$ into binary writing and also to store the non-zero indices $(n_1, \ldots, n_M)$. This is done simply using techniques derived from information theory, in particular entropy coding methods, see [14, Chap. 10].

3.3 Inverse Problems and Sparsity

3.3.1 Inverse Problems

Before f data can be stored, it is most of the time necessary to carry out a preliminary restoration step, which consists in improving the quality of the data from observations of low quality, that is to say –9 of low resolution, possibly blurred , entangled with errors and noisy. In order to take into account the whole chain of data formation, we model mathematically the acquisition process in the form

$$y = \Phi f + w \in \mathbb{R}^P \tag{3.4}$$

where $y \in \mathbb{R}^P$ are the P observations measured by the device, $w \in \mathbb{R}^P$ is a measurement noise (unknown), $f \in \mathbb{R}^Q$ is the image (unknown) that one wishes to recover, and $\Phi : \mathbb{R}^Q \to \mathbb{R}^P$ is an operator modeling the acquisition apparatus, and which is assumed to be "linear". This means that Φ may be considered as a (gigantic) matrix $\Phi \in \mathbb{R}^(P \times Q)$. It is important to note that most of the time this matrix Φ is never explicitly stored, it is manipulated implicitly by means of fast operations (convolution, masking, etc.).

This model, which may seem rather restrictive (in particular the hypothesis of linearity) makes it possible to model a surprising quantity of situations that one meets in practice. For example,

- Denoising: $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$, $P = Q$ and one is in the (simplest) situation in which one only seeks to remove the noise w ;
- Deconvolution: (see Figure (3.3), center) $\Phi f = \varphi \star f$ is a convolution by a filter φ modeling for example the blur of a camera (either a blur of shake or a blur due to development) ;
- Missing data: (see Figure (3.3), right) $\Phi = \mathrm{diag}(\mu_q)_{q=1}^Q$ is a diagonal masking operator, such as $\mu_q = 1$ if the data indexed by q (for example, one pixel) is observed, and $\mu_q = 0$ if the data is missing;
- tomographic imagery: Φ is a more complex linear operator, calculating integrals along straight lines (the Radon transform), see [14, Sect. 2.4].

There are many other examples (in medical imaging, seismic, astrophysics, etc.), and in each case, calculating a good approximation of f from y is very difficult. Indeed, with the exception of denoising (ie $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$), the formula $\Phi^{-1}y = f + \Phi^{-1}w$ can not be used either because Φ is not invertible (for example for missing data), or because Φ has very small eigenvalues (for deconvolution or tomography), so that $\Phi^{-1}w$ is going to be very large, and thus $\Phi^{-1}y$ is a very bad approximation of f.

3.3.2 Sparse Regularization

To remedy this problem, we need to replace Φ^{-1} with an approximate "inverse" which takes into account additional assumptions about the f signal we are looking for. Recent methods, which give the best results on complex data, use an approximate inverse which is nonlinear. This may seem contradictory because Φ is linear, but the use of nonlinear methods is crucial to take advantage of realistic assumptions about complex data such as images. Based on the approximation and compression techniques discussed in the previous section, current methods seek to exploit the fact that one can approach f with a sparse approximation Ψx with $\|x\|_0 \leqslant M$. Given a parameter $M > 0$, we will look to approximate f by $f^\star = \Psi x^\star$ where $x^\star$ is a solution of

$$x^\star \in \operatorname*{argmin}_{x \in \mathbb{R}^N} \{\|y - \Phi\Psi x\|_2 \; ; \; \|x\|_0 \leqslant M\} \tag{3.5}$$

We see that (3.5) is quasi-identical to (3.1), except that $f \in \mathbb{R}^Q$ (unknown) has been replaced by $y \in \mathbb{R}^P$, and that matrix $\Psi \in \mathbb{R}^{Q \times N}$ by matrix product $\Phi\Psi \in \mathbb{R}^{P \times N}$. In the particular case of denoising, $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$, the problems (3.2) and (3.5) are equivalent and have the same solution, so that the nonlinear approximation solves the denoising problem .

In the case of any operator Φ, the problem (3.5) is however an optimization problem extremely difficult to solve. Indeed, even if Ψ is an orthonormal basis, in general (except in the case of denoising $\Phi = \mathrm{Id}_{\mathbb{R}^Q}$), the matrix $\Phi\Psi$ is not orthogonal, so that the formula (3.3) is not applicable, and (3.5) is a NP-difficult combinatorial search problem.

3.3.3 ℓ^1 Regularization

The approximation of the solutions of the problem (3.5) using efficient methods is one of the most active subjects of research in data processing (and more generally in applied mathematics, imagery, statistics and machine learning). There are many methods, including greedy algorithms (see, for example, [15]) and convex relaxation methods. We will focus on this second class of methods. One

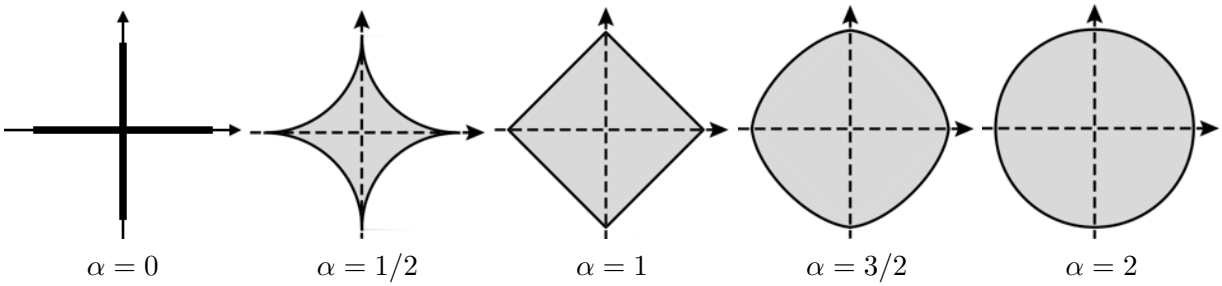


Figure 3.4: Balls B_α for different values of α.

way (heuristic) to introduce these techniques is to replace $\|\cdot\|_0$ in the (3.5) problem with the function $\|\cdot\|_\alpha^\alpha$, which is set to $\alpha > 0$ by

$$\|x\|_\alpha^\alpha \stackrel{\text{def.}}{=} \sum_{n=1}^N |x_n|^\alpha.$$

Figure 3.4 shows in the (unrealistic but convenient to draw) case of $N = 2$ coefficients, the balls of the $B_\alpha \stackrel{\text{def.}}{=} \{x \,;\, \|x\|_\alpha \leqslant 1\}$ units associated with these functional $\|\cdot\|_\alpha$. It can thus be seen that B_α "tend" to the "unit ball" associated with the counting measure $\|\cdot\|_0$ as α tends to 0,

$$B_\alpha \stackrel{\alpha\to 0}{\longrightarrow} B_0 \stackrel{\text{def.}}{=} \left\{x \in [-1,1]^N \,;\, \|x\|_0 \leqslant 1\right\},$$

the convergence of these sets (which is well visualized in the figure) being in the sense for example of the Hausdorff distance. The limiting ball B_0 is composed of extremely sparse vectors, since they are composed of a single non-zero component.

One then has to take into account two conflicting points to choose a value of α:

- In order to have a functional enforcing the sparsity of vectors, we want to use a value of α as low as possible to replace $\|\cdot\|_0$ by $\|\cdot\|_\alpha$.
- In order to calculate the solution of (3.5) with $\|\cdot\|_\alpha$ instead of $\|\cdot\|_0$, it is important that the $\|\cdot\|_\alpha$ be *convex*. The convexity is indeed essential in order to obtain a problem that is not NP-difficile and to be able to benefit from fast calculation algorithms. These algorithms find an exact solution $x^\star$ in polynomial time or quickly converge to this solution.

The convexity constraint of $\|\cdot\|_\alpha$ requires that the set B_α be convex, which equivalently means that $\|\cdot\|_\alpha$ must be a *norm*. This imposes that $\alpha \geqslant 1$. Taking these two constraints into account leads naturally to the choice $\alpha = 1$, so that we consider the convex optimization problem (that is, seeks to minimize a convex function on a convex set)

$$x^\star \in \operatorname*{argmin}_{x\in\mathbb{R}^N} \left\{ \|y - \Phi\Psi x\|_2 \,;\, \|x\|_1 = \sum_{n=1}^N |x_n| \leqslant \tau \right\}, \tag{3.6}$$

so that the retrieved image is defined as $f^\star = \Psi x^\star$. It may be noted that a parameter $\tau > 0$ was used here, which plays a role similar to parameter M which appears in (3.5). The question of choosing this parameter τ is crucial. If the noise w is small, then we want that $\Phi f^\star = \Phi\Psi x^\star$ be close to y, and so we will choose τ big. On the contrary, if the noise w is important, in order to

f Original Observations y Reconstruction $f^\star$

Figure 3.5: Examples of reconstruction with missing data, $\Phi = \mathrm{diag}(\mu_q)_{q=1}^{Q}$ with $\mu_q \in \{0,1\}$ and a number of observed data $\sharp\{q\ ;\ \mu_q = 1\} = 10\%$.

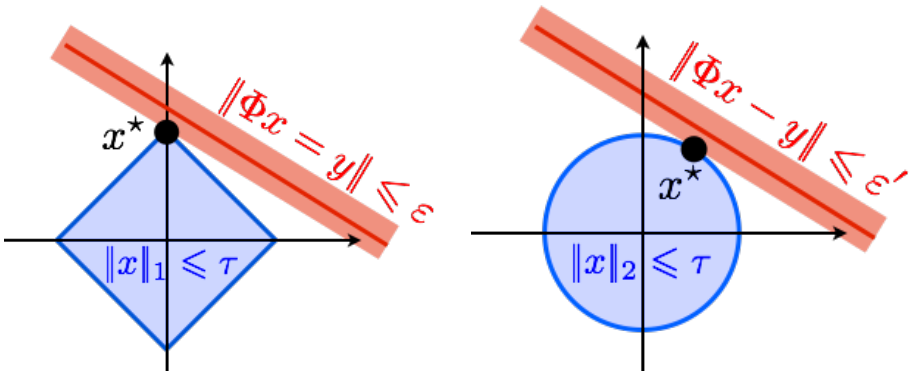
obtain a greater denoising effect, the value of τ is reduced. The choice of a τ "optimal" is a difficult search problem, and there is no universal response, the existing strategies strongly depend on the Φ operator as well as the atoms family Ψ .

The problem (3.6) was initially proposed by engineers in the fields of seismic imaging (see for example [19]), and it was introduced jointly in signal processing under the name "basis pursuit" [6] and in statistics under the name "Lasso" [22].

The problem (3.6), although convex, remains a difficult problem to solve because of the non-differentiability of $\|\cdot\|_1$ and large data size (N is very large). This is the price to pay for getting good quality results. As will be explained in the following paragraph, it is in fact the nondifferentiability of $\|\cdot\|_1$ which makes it possible to obtain sparsity. The development of efficient algorithms to solve (3.6) is a very active field of research, and we refer to [23, section 6] for a review of these methods. Figure 3.5 shows an example of missing data interpolation performed by solving (3.6) in a Ψ family of translationally invariant wavelets.

3.3.4 From Intuition to Theory

The figure 3.6 shows intuitively why the $x^\star$ solution calculated by replacing $\|\cdot\|_0$ by $\|\cdot\|_\alpha$ in (3.5) is better (in the sense that it is more sparse) if we choose $\alpha = 1$ (that is, if we solve (3.6)) than if we choose $\alpha = 2$ (a similar conclusion is obtained for other values of $\alpha > 1$). The figure is made in the (very simple) case of $N = 2$ coefficients and $P = 1$ observations. The crucial point, which makes the solution of (3.6) sparse, is that the ball B_1 associated with the ℓ^1 standard is "pointed" so that the $x^\star$ solution is located along the axes. This is not the case for ball B_2 associated with standard ℓ^2, which gives a $x^\star$ solution that is not along the axes, and therefore is not sparing. This phenomenon, already visible in dimension 2, is actually accentuated when the dimension increases, so that the approximation obtained by replacing $\|\cdot\|_0$ by $\|\cdot\|_1$ becomes better in large dimension. This phenomenon is called the "blessing of dimensionality" by David Donoho [8]: although the data become very expensive and complex to treat, there are effective analysis and processing techniques when they are sufficiently sparse. To make this intuition rigorous, however, is difficult, and this is the object of research still in progress for operators Φ such as convolutions [3, 12]. The analysis in the case of the operators that one meets for example in medical imaging is an open mathematical problem.



ℓ^1 minimisation ℓ^2 minimisation

Figure 3.6: Comparison of the minimization with constraints of type $\|x\|_\alpha \leqslant \tau$ for $\alpha \in \{1, 2\}$. A $x^\star$ solution is obtained when a tube $\{x \; ; \; \|\Phi x - y\| \leqslant \varepsilon\}$ is sufficiently large (ie gradually growing ε) as it is tangent in $x^\star$ to the ball $\{x \; ; \; \|x\|_\alpha \leqslant \tau\}$.

3.4 Compressed sampling

There exists a particular class of operators Φ for which it is possible to analyze very precisely the performances obtained when we solve (3.6). This is the case where Φ is drawn randomly according to some distributions of random matrices. Using random matrices may seem strange, because the operators mentioned above (convolution, tomography, etc.) are not at all. In fact, this choice is motivated by a concrete application proposed jointly by Candès, Tao and Romberg [2] as well as Donoho [9], and which is commonly called "compressed sensing" in English).

3.4.1 Single Pixel Camera

In order to make illustrate the exposition, we will discuss the "single pixel camera" prototype developed at Rice University [11], and which is illustrated by the figure 3.7 (left). It is an important research problem of developing a new class of cameras allowing to realize both the sampling and the compactness of an image. Instead of first sampling very finely (ie with very large Q) the analog signal $\tilde{f}$ to obtain a $f \in \mathbb{R}^Q$ image then compressing enormously (ie with M small) using (3.3), we would like to dispose directly of an economic representation $y \in \mathbb{R}^P$ of the image, with a budget P as close to M and such that one is able to "decompress" y to obtain a good approximation of the image f.

The "single-pixel" hardware performs the compressed sampling of an observed scene $\tilde{f}$ (the letter "R" in Figure 3.7), which is a continuous function indicating the amount of light $\tilde{f}(s)$ reaching each point $s \in \mathbb{R}^2$ of the focal plane of the camera. To do this, the light is focused against a set of Q micro-mirrors aligned on the focal plane. These micro-mirrors are not sensors. Unlike conventional sampling (described in Section 3.1), they do not record any information, but they can each be positioned to reflect or absorb light. To realize the complete recording, one very quickly changes P times the configurations of the micro-mirrors. For $p = 1, \ldots, P$, $\Phi_{p,q} \in \{0, 1\}$ is used, depending on whether the micrometer at position q has been placed in the absorbing (0) or reflective (value 1) position at step p of the acquisition. The total light reflected at step p is then accumulated into a single sensor (hence the name "single pixel", in fact it is rather a "single sensor"), denoted "PD" in the figure, which achieves a linear sum of the reflected intensities to obtain the recorded

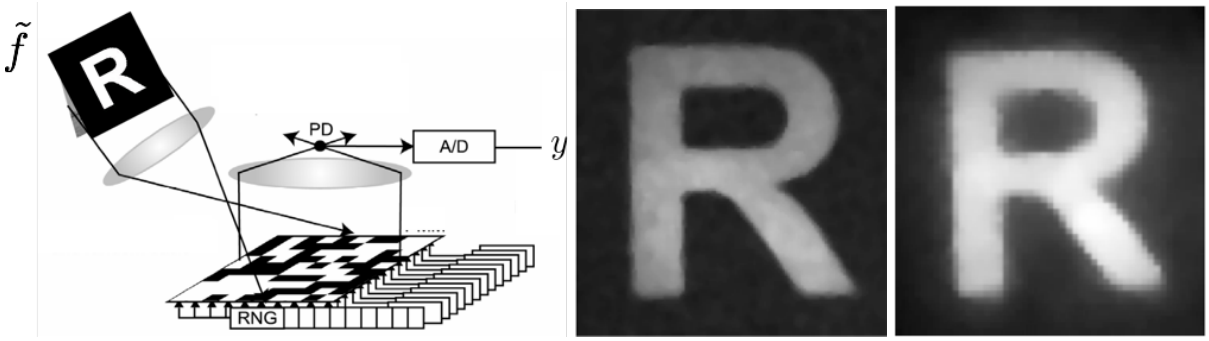


Diagram of the device f $f^\star$, $P/Q = 6$

Figure 3.7: Left: diagram of the single-pixel acquisition method. Center: image $f \in \mathbb{R}^Q$ "ideal" observed in the focal plane of the micro-mirrors. Right: image $f^\star = \Psi x^\star$ reconstructed from observation $y \in \mathbb{R}^P$ with a compression factor $P/Q = 6$.

$y_p \in \mathbb{R}$ value. In the end, if the light intensity arriving on the surface c_q of the mirror indexed by $f_q = \int_{c_q} \tilde{f}(s)\mathrm{d}s$ is denoted (as in the 3.1 section) as q, the equation that connects the discrete image $f \in \mathbb{R}^Q$ "seen through the mirrors" to P measures $y \in \mathbb{R}^P$ is

$$\forall\, p = 1, \ldots, P, \quad y_p = \sum_q \Phi_{p,n} \int_{c_n} \tilde{f}(s)\mathrm{d}s = (\Phi f)_p,$$

which corresponds exactly to (3.4). It is important to note that the mirrors do not record anything, so in particular the f discrete image is never calculated or recorded, since the device directly calculates the compressed representation y from the analog signal $\tilde{f}$. The term w models here the acquisition imperfections (measurement noise). The compressed sampling therefore corresponds to the transition from the observed scene $\tilde{f}$ to the compressed vector y. The "decompression" corresponds to the resolution of an inverse problem, whose goal is to find a good approximation of f (the discrete image "ideal" as seen by the micro-mirrors) from y.

3.4.2 Theoretical Guarantees

An important feature of this inverse problem is that one can choose, as desired, the configurations of the micro-mirrors, which amounts to saying that one can choose freely the matrix $\Phi \in \{0,1\}^{P\times Q}$. The question is therefore to make the best choice, so that the inverse problem can be solved effectively. If we make the hypothesis that the signal f to be reconstructed is compressible in an orthonormal basis Ψ (that is to say that $f \approx \Psi x_0$ with $M \overset{\text{def.}}{=} \|x_0\|_0$ small) then several recent works, starting with [2, 9], showed that method (3.6) is effective if Φ is chosen as a realization of some random matrices. For the single-pixel camera, each $\Phi_{p,n}$ can then be randomly drawn with a probability of $1/2$ for the values 0 and 1. In practice, a pseudo-random generator is used, so that both the person who compresses the data and the person who is going to decompress them knows the matrix Φ perfectly (because they can communicate the seed of the generator). The figure 3.7 (right) shows an example of reconstruction obtained for the case of the single-pixel apparatus with such a random choice of matrix Φ, with Ψ a translation-invariant family of wavelets (see [14, Section 5.2] for a description of this family).

It has been shown by [2, 9] that there exists a constant C such that if $f = \Psi x_0$ where x_0 are the coefficients of the image to be retrieved, where Ψ is an orthogonal basis therefore in particular $Q = N$), and if the number P of measurements satisfies

$$\frac{P}{M} \geqslant C \log\left(\frac{N}{M}\right) \quad \text{where} \quad M \stackrel{\text{def.}}{=} \|x_0\|_0 \tag{3.7}$$

then a solution $f^\star = \Psi x^\star$ computed by (3.6) tends to f when the w noise tends to 0 and τ tends to$+\infty$. This result is true "with high probability" on the random drawing of the matrix Φ, that is to say a probability tending rapidly towards 1 when N increases. In particular, if there is no noise, $w = 0$, taking $\tau \to +\infty$, the method makes it possible to find exactly f if P satisfies (3.7). This theory also allows to take account of "compressible" data, ie if we only assume that f is close to (but not necessarily equal to) Ψx_0 with $M \stackrel{\text{def.}}{=} \|x_0\|_0$ small.

Intuitively, this theoretical result means that compressed sampling can do almost "as well" by calculating $\Psi x^\star$ from y (solving (3.6)) than a usual compression method (MP3, JPEG , JPEG2000, MPEG, etc.) that would know exactly the f signal and calculate the best approximation Ψx_0 with $M \stackrel{\text{def.}}{=} \|x_0\|_0$ coefficients (solving (3.1) via the formula (3.2)). The precise meaning of the qualifier "equally" corresponds to the $C \log(N/M)$ multiplying factor, which bounds P/M. This factor corresponds to the "extra cost" of the compressed sampling method (which calculates P measurements) compared to a usual compression method (which calculates M coefficients). Despite this additional cost, the compressed sampling method has many advantages: saving time and energy (at the same time sampling and compression), "democratic" coding (all y_n coefficients play the same role, and therefore none has a dominant role, unlike the coding of the coefficients of x_0 which have an importance proportional to their amplitude), coding automatically encrypted (if Φ is not known, f can not be found from y). The value of the C constant depends on the meaning given to "with high probability". If this probability bears only Φ, but must be true for all x_0 (worst case analysis), then it is very large (see [10]). If, on the other hand, if the high probability is both on Φ and x_0 (so that the theoretical result is true for almost all the signals) then it can be shown that for example, for $N/P = 4$, we have $C \log(N/M) \sim 4$ (see [4]), which remains a significant overhead but is acceptable for some applications.

The "single pixel" camera is a particular application of the compressed sampling technique. Applications to photography are limited because the CCD sensors of cameras are powerful and inexpensive. Compressed sampling is likely to have an impact on applications where measurements are difficult to acquire or costly. Another source of potential applications is medical imaging, for example by magnetic resonance imaging (MRI). In these fields, however, it is impossible to obtain totally random matrices, so that the theory of compressed sampling can not be applied directly. Encouraging results on these applications have however been obtained, see for example [1, 5].

Conclusion

Recent advances in data analysis have made it possible to extend the scope of compression in order to deal with difficult inverse problems in imaging, but also in other fields (recommendation system, network analysis, etc.). These advances have been made possible by the use of a very broad spectrum of techniques in applied mathematics, covering both harmonic analysis, nonlinear approximation, non-smooth optimization and probability, but also analysis and PDEs (which were not mentioned in this article). Sparse methods associated with ℓ^1 regularization are only the tip

of the iceberg, and more advanced regularizations make it possible to obtain better results by taking into account the complex geometric structures of the data. For more details on these latest advances, we recommend reading the article [23], as well as visiting the web site "Numerical Tours of Signal Processing" [17], which features many computer codes to carry out the numerical experiments presented here, as well as many others.

Acknowledgments

I would like to thank Charles Dossal, Jalal Fadili, Samuel Vaiter, Stéphane Seuret and the anonymous reviewer for their invaluable help.

Bibliography

[1] B. Adcock, A. C. Hansen, C. Poon, and B. Roman. Breaking the coherence barrier: asymptotic incoherence and asymptotic sparsity in compressed sensing. *CoRR*, abs/1302.0561, 2013.

[2] E. J. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[3] E.J. Candès and C. Fernandez-Granda. Towards a mathematical theory of super-resolution. *Communications on Pure and Applied Mathematics*, 67(6):906–956, 2014.

[4] V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. Willsky. The convex geometry of linear inverse problems. *Foundations of Computational Mathematics*, 12(6):805–849, 2012.

[5] N. Chauffert, P. Ciuciu, J. Kahn, and P. Weiss. Variable density sampling with continuous trajectories. *SIAM Journal on Imaging Sciences*, 7(4):1962–1992, 2014.

[6] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1):33–61, 1999.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[8] D. L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *Lecture "Math Challenges of the 21st Century"*, 2000.

[9] D. L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[10] C. Dossal, G. Peyré, and J. Fadili. A numerical exploration of compressed sampling recovery. *Linear Algebra and Applications*, 432(7):1663–1679, 2010.

[11] M. F. Duarte, M. A. Davenport, D. Takbar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, March 2008.

[12] V. Duval and G. Peyré. Exact support recovery for sparse spikes deconvolution. *Foundations of Computational Mathematics*, 15(5):1315–1355, 2015.

[13] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

[14] S. G. Mallat. *A wavelet tour of signal processing*. Elsevier/Academic Press, Amsterdam, third edition, 2009.

[15] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[16] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[17] G. Peyré. The numerical tours of signal processing - advanced computational signal and image processing, www.numerical-tours.com. *IEEE Computing in Science and Engineering*, 13(4):94–97, 2011.

[18] J. Rissanen and G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, 1979.

[19] F. Santosa and W.W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330, 1986.

[20] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[21] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers*, 37(1):10–21, 1949.

[22] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B. Methodological*, 58(1):267–288, 1996.

[23] S. Vaiter, G. Peyré, and J. Fadili. *Low Complexity Regularization of Linear Inverse Problems*, chapter Sampling Theory, a Renaissance, pages 103–153. Springer-Birkhäuser, 2015.