An Introduction to Data Sciences

Gabriel Peyré
CNRS & DMA, École Normale Supérieure, PSL

CHAPTER 2

Image processing

September 9, 2026

A digital image is an array of numbers, so changing an image means computing with those numbers. This chapter develops simple methods for storing images, removing noise, detecting edges and manipulating colors. Each example connects a visible effect with an explicit formula and explains the information that the operation preserves or loses.

2.1 Pixels and grayscale values

A grayscale image with n rows and p columns contains np *pixels*. In the eight-bit representation considered here, each pixel stores an integer between 0 and $255 = 2^8 - 1$. Zero is displayed as black, 255 as white, and the intermediate values as shades of gray. Other bit depths and floating-point representations are also used.

The flower in Figure 2.1 has $256 \times 256 = 65\,536$ pixels. The enlargement shows a 5×5 patch: each displayed shade corresponds to one of the numbers in the array. A photograph may contain millions of pixels, but the same representation applies.

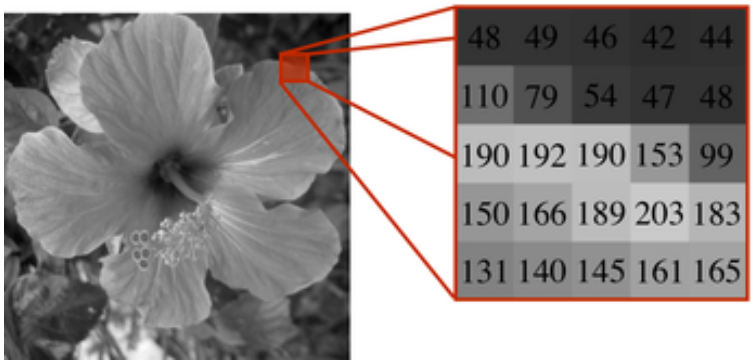


Figure 2.1. An image and a magnified 5×5 patch of pixel values.

2.2 Storing an image

2.2.1 Binary representation and storage units

An integer is represented in binary by decomposing it into powers of two. For example,

$$179 = 2^7 + 2^5 + 2^4 + 2^1 + 2^0 = \underbrace{(10110011)_2}_{\text{binary representation}} .$$

Reading from 2^7 down to 2^0, a bit is one when the corresponding power occurs and zero otherwise. Eight bits suffice for every integer from 0 to 255.

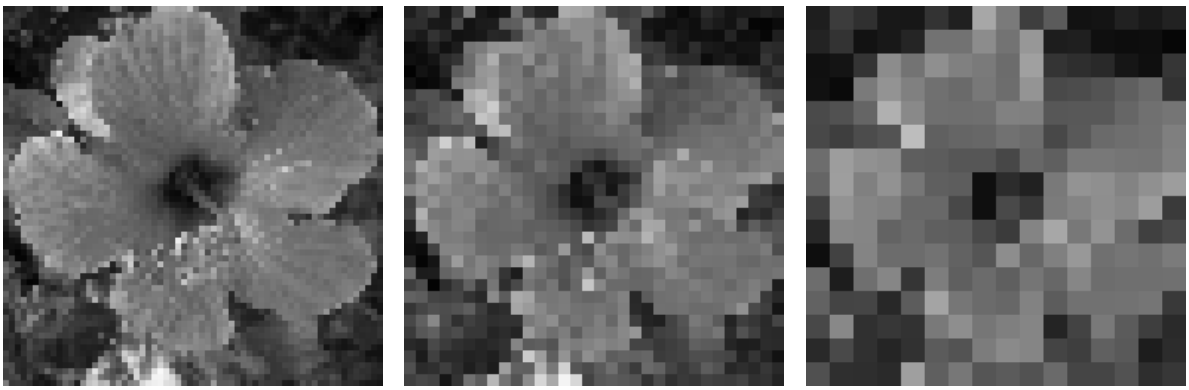
An uncompressed eight-bit grayscale image therefore requires $8np$ bits, or np *bytes*, for its pixel values. The 256×256 flower uses

$$256 \times 256 \times 8 = 524\,288 \text{ bits} = 65\,536 \text{ bytes}.$$

This is 65.536 kB in decimal units, or 64 KiB in binary units: 1 kB is 1000 bytes, whereas 1 KiB is 1024 bytes. For comparison, a 240×240 image would require 57 600 bytes, or 56.25 KiB. File headers and metadata add to these payload sizes.

2.2.2 Subsampling

Reducing the number of pixels reduces storage. Keeping every fourth row and every fourth column divides the pixel count by $4^2 = 16$, when both dimensions are divisible by four. Figure 2.2 shows increasingly coarse grids.



One row/column in 4 One row/column in 8 One row/column in 16

Figure 2.2. Subsampling on successively coarser grids.

Discarding rows and columns can lose fine detail and create false patterns, an effect called *aliasing*. A practical resizing method usually smooths the image before reducing its sampling rate. The next chapter explains the mathematical reason for this precaution.

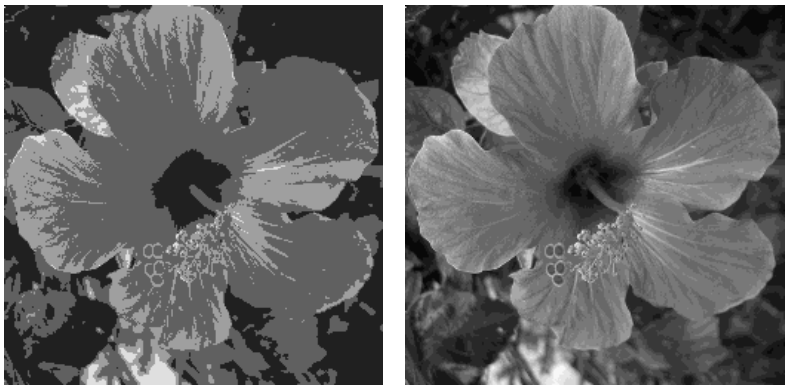
2.2.3 Quantization

Storage can also be reduced by using fewer possible values per pixel. To retain four levels, replace an input value $a \in \{0, \dots, 255\}$ by

$$q(a) = \left\lfloor \frac{a}{64} \right\rfloor \in \{0, 1, 2, 3\}.$$

Thus 0–63 map to 0, 64–127 to 1, 128–191 to 2, and 192–255 to 3. When displayed on a scale from black to white, these indices correspond respectively to black, dark gray, light gray and white; they can be rendered using the values $85q(a)$.

Four indices require two bits, rather than eight, giving a factor-four saving if the indices are packed into the file. Merely storing the indices in eight-bit containers would not achieve that saving. Figure 2.3 compares four and sixteen levels.



4 gray levels 16 gray levels

Figure 2.3. Quantization with two different numbers of levels.

Quantization is generally irreversible: many original values share the same index. Lossless compression, by contrast, preserves every input value while exploiting redundancy in its representation. JPEG 2000 uses wavelets and offers both lossless and lossy modes; it is one example of a codec, not a universally best choice for every image and criterion.[1]

2.3 Removing noise

2.3.1 Local averaging

Noise produces unwanted fluctuations in pixel values. It is particularly visible in low-light photographs, although its distribution depends on the sensor and acquisition process. A simple denoising method replaces each pixel by the mean of a surrounding window.

For the 3×3 neighborhood in Figure 2.4, replace the central value a by

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

The displayed values give

$$\frac{190 + 192 + 79 + 54 + 47 + 153 + 203 + 189 + 166}{9} = \frac{1273}{9} \simeq 141.44.$$

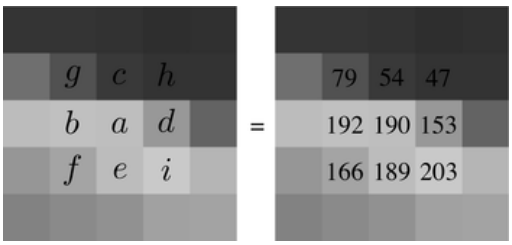
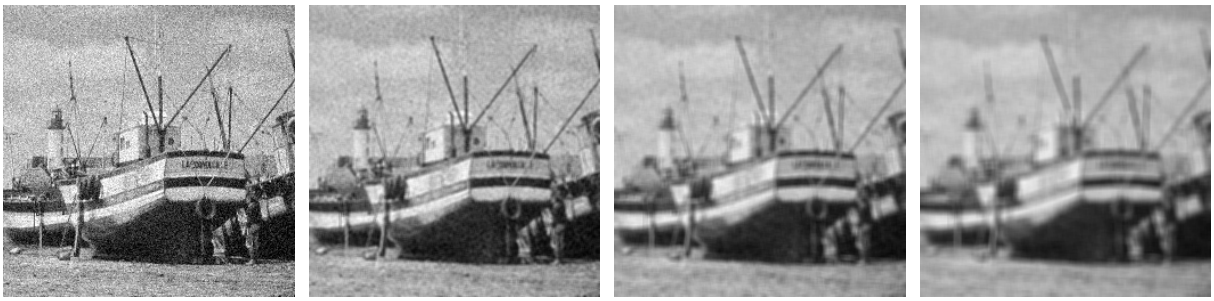


Figure 2.4. A 3×3 neighborhood used for local averaging and median filtering.

All averages must be computed from the same input image. Updating pixels in place and then reusing those updated values defines a different filter. At the image boundary, one must also choose a rule, such as reflecting the image or restricting and renormalizing the window.

Why does averaging help? If a locally constant signal is corrupted by independent, zero-mean noise of variance σ^2, the average of m pixels has noise variance σ^2/m. Near an edge, however, the underlying signal is not constant: averaging mixes distinct regions and introduces blur. A larger window trades more noise reduction for more loss of detail.



Noisy image | 3×3 mean | 5×5 mean | 7×7 mean

Figure 2.5. Local averaging reduces noise but blurs edges as the window grows.

[1] https://jpeg.org/jpeg2000/

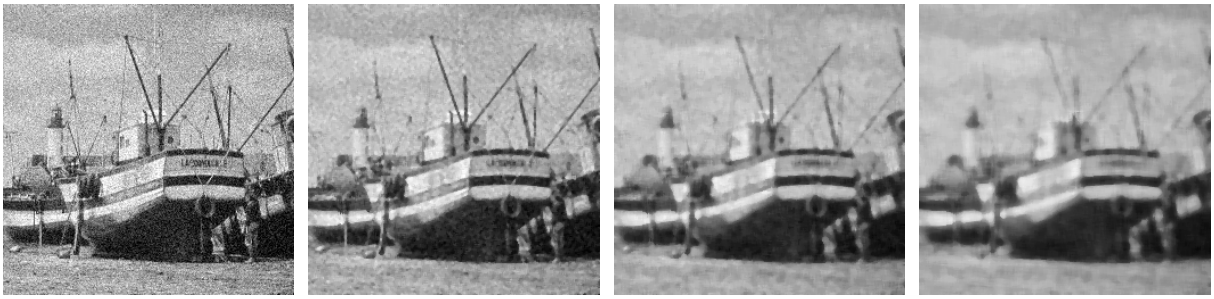
2.3.2 Local median

The *median* of an odd number of values is the middle value after sorting. The nine values above, in increasing order, are

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

so their median is 166. A median filter replaces each pixel by the median of its input neighborhood.

The median is resistant to isolated extreme values and often preserves sharp transitions better than the mean. It is particularly useful for impulse noise, but it is not uniformly better for every noise model or image. Large windows can erase thin objects and small details. Figure 2.6 illustrates this tradeoff.



Noisy image | 3×3 median | 5×5 median | 7×7 median

Figure 2.6. Median filtering with increasing window size.

2.4 Detecting edges

Edges are locations where pixel values change rapidly, for instance at a boundary between a dark object and a bright background. A simple detector compares the neighbors on opposite sides of a pixel.

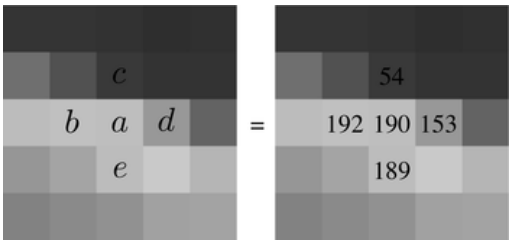


Figure 2.7. The central pixel a and its left, upper, right and lower neighbors b, c, d, e.

With the arrangement of Figure 2.7, define

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

For unit pixel spacing, $\ell/2$ is the magnitude of the gradient estimated by centered differences. The values in the figure yield

$$\ell = \sqrt{(192-153)^2 + (54-189)^2} = \sqrt{19\,746} \simeq 140.52.$$

We have $\ell = 0$ exactly when $b = d$ and $c = e$. This does not imply that all five values are equal: the detector can miss some fine oscillations. A large value indicates a strong local variation, which may come from an object boundary, texture or noise.

For display, the value is clipped to $\min(\ell, 255)$ and rounded. The minimum is needed because ℓ can exceed 255. Figure 2.8 displays large responses in white. Denoising before differentiation can help because differences also amplify noise.

2.5 Color images

Original image Edge response ℓ

Figure 2.8. Edge detection by differences between opposite neighbors.

2.5.1 RGB channels and additive color

An RGB image has three channels, representing red, green and blue. They are separate arrays, but are generally *not statistically independent*. In an eight-bit representation, a pixel is a triplet $(r, g, b) \in \{0, \ldots, 255\}^3$ and requires 24 bits before compression.

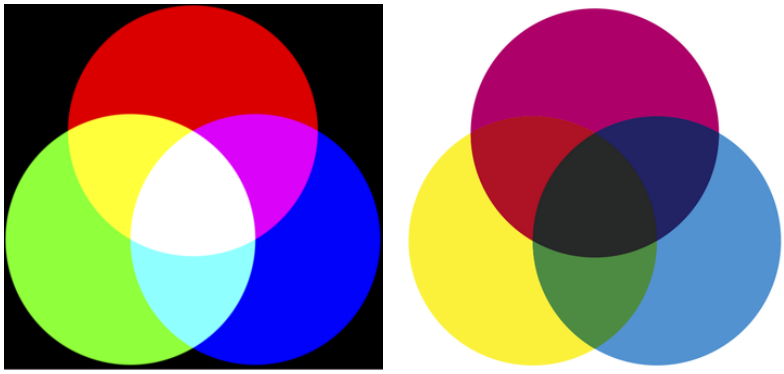
The values $(255, 0, 0)$, $(0, 255, 0)$ and $(0, 0, 255)$ display red, green and blue respectively. Combining emitted light follows *additive* color mixing: red plus green, $(255, 255, 0)$, gives yellow; red plus blue, $(255, 0, 255)$, gives magenta. White is $(255, 255, 255)$ and black is $(0, 0, 0)$.

Color image Mean-channel grayscale

Figure 2.9. A color image and a grayscale representation.

Red channel Green channel Blue channel

Figure 2.10. The three RGB channels, displayed in their corresponding colors.



Additive mixing Subtractive mixing

Figure 2.11. Idealized additive and subtractive color mixing.

2.5.2 CMY channels and subtractive color

An idealized complementary representation uses cyan, magenta and yellow:

$$c = 255 - r, \qquad m = 255 - g, \qquad y = 255 - b.$$

For example, blue $(r, g, b) = (0, 0, 255)$ corresponds to $(c, m, y) = (255, 255, 0)$. These variables describe the *subtractive* action of ideal pigments on white light: cyan absorbs red, magenta absorbs green, and yellow absorbs blue.

Cyan channel Magenta channel Yellow channel

Figure 2.12. The complementary CMY channels, shown against white backgrounds.

Real printers use device-dependent color conversions, usually with an additional black channel (CMYK). A screen still emits RGB light, so CMY data must be converted for display. Neither a fixed set of display primaries nor a fixed set of inks reproduces every visible color.

2.6 Changing brightness and contrast

2.6.1 A grayscale intensity

A simple way to obtain a grayscale image is to average the three channels:

$$a = \frac{r + g + b}{3}.$$

This is the intensity used in the elementary examples here. It is not the physical *luminance*, which weights linear-light components according to visual sensitivity. In particular, an arithmetic mean of encoded RGB values is not a calibrated measurement of emitted light.[2]

[2]For the definition of relative luminance in sRGB, see https://www.w3.org/TR/WCAG22/#dfn-relative-luminance.

2.6.2 Pointwise changes in grayscale

Replacing a by $255 - a$ exchanges black and white and produces a negative. To darken an image, use $a \mapsto a^2/255$; to brighten it, use $a \mapsto \sqrt{255a}$. Both maps preserve the interval $[0, 255]$.

Negative | Square: darker | Square root: brighter

Figure 2.13. Three pointwise transformations of grayscale intensity.

The square and square-root maps are inverses as functions on the real interval. After rounding to integer pixel values, they need not undo one another exactly. Their common generalization is the power transformation

$$\widetilde{a} = 255 \left(\frac{a}{255} \right)^{\gamma}, \qquad \gamma > 0,$$

with $\widetilde{a} = 0$ when $a = 0$. For $0 < a < 255$, the image becomes brighter when $\gamma < 1$ and darker when $\gamma > 1$. The cases $\gamma = 1/2$ and $\gamma = 2$ give the square root and square, respectively; $\gamma = 1$ leaves the values unchanged.

2.6.3 A color transformation

Applying unrelated changes to the three channels can alter colors strongly. One elementary alternative changes the mean intensity a while keeping the residual $(r - a, g - a, b - a)$ fixed. Define

$$\begin{aligned} \widetilde{r} &= \operatorname{clip}(r + \widetilde{a} - a), \\ \widetilde{g} &= \operatorname{clip}(g + \widetilde{a} - a), \\ \widetilde{b} &= \operatorname{clip}(b + \widetilde{a} - a), \end{aligned} \qquad \operatorname{clip}(u) = \max(0, \min(255, u)).$$

Without clipping, the new mean is $\widetilde{a}$ and the differences between channels are preserved. Clipping can change both properties, and this construction does not guarantee exact preservation of perceived hue or saturation. Values are rounded only after the computation.

$\gamma = 0.5$ | 0.75 | 1 | 1.5 | 2 | 3

Figure 2.14. Changing a color image through a power transformation of its mean intensity.

2.7 Images as matrices

2.7.1 Transposition and rotation

Write an image as a matrix $A = (a_{i,j}) \in \mathbb{R}^{n \times p}$. Rows are indexed from top to bottom by $1 \leqslant i \leqslant n$ and columns from left to right by $1 \leqslant j \leqslant p$. Its transpose $B = A^{\mathsf{T}} \in \mathbb{R}^{p \times n}$ satisfies

$$b_{j,i} = a_{i,j}.$$

It exchanges the two coordinate directions and reflects the image across the diagonal direction. A clockwise quarter-turn produces another $p \times n$ matrix C given by

$$c_{j,i} = a_{n-i+1,j}, \qquad 1 \leqslant j \leqslant p, \quad 1 \leqslant i \leqslant n.$$

Equivalently, the original pixel (i, j) moves to $(j, n-i+1)$. These operations permute pixel values and therefore lose no information. Apply the same permutation to all channels of a color image.

A A^{T} Clockwise rotation

Figure 2.15. Transposition and rotation by a quarter-turn.

2.7.2 A crossfade between two images

For images A and B of the same dimensions and channel format, define

$$C_t = (1-t)A + tB, \qquad 0 \leqslant t \leqslant 1.$$

This formula is applied to each pixel and channel. The endpoints are $C_0 = A$ and $C_1 = B$; intermediate values produce a *crossfade*. Because the weights are nonnegative and sum to one, values stay in $[0, 255]$ before rounding. This blends colors at fixed locations; it does not move objects from one location to another.

$t = 0$ 0.2 0.4 0.6 0.8 1

Figure 2.16. Linear interpolation from image A to image B.

Further reading. Reproducible examples are available in the Numerical Tours [1]. This chapter draws in part on the author's article *Le traitement numérique des images* in *Images des mathématiques*.

Glossary

- **Bit and byte**: a binary digit; a group of eight bits.
- **Pixel**: one entry of an image array.
- **Channel**: one component array of a multichannel image.
- **Quantization**: replacing values by indices from a finite set.
- **Noise**: unwanted fluctuations introduced by acquisition or processing.
- **Mean**: the sum of values divided by their number.
- **Median**: the central sorted value for an odd-sized set.
- **Edge**: a rapid spatial change in image values.
- **Blur**: a loss of sharp transitions and fine detail.
- **Contrast**: differences between light and dark regions.
- **Luminance**: a photometric quantity defined using visual sensitivity.
- **Matrix**: a rectangular array $(a_{i,j})$.
- **Resolution**: a term that may refer to pixel dimensions or to spatial resolving power; these are distinct notions.
- **Crossfade**: a weighted average of two images at fixed pixel locations.

References

[1] Gabriel Peyré. The numerical tours of signal processing. *Computing in Science & Engineering*, 13(4):94–97, 2011. doi: 10.1109/MCSE.2011.71. URL https://www.numerical-tours.com/.