Image Analysis Fundamentals

Aaron Ponti

Fall 2023

Contents

1 Introduction

1.1 Digital image processing

Digital image processing is any form of signal processing that takes an **image** as an input and delivers either another image, that is corrected or enhanced in some way, or some **measurements** or **parameters**.

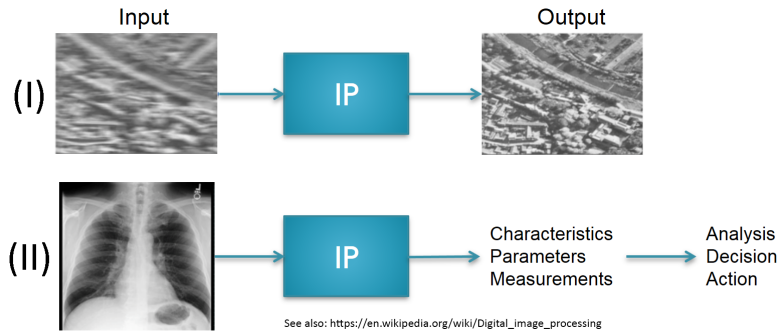


Figure 1: Digital image processing

If the output of image processing is a set of measurements, the type and number of subsequent steps will vary based on the specific application. We may perform **statistical analysis** of the extracted **features**, take some **decision** based on the outcome, or initiate further **actions**. Frequently, multiple iterations of type (I) processing will be required to produce a modified image that is conducive to the extraction of model parameters or statistical measures in type (II) processing.

1.2 Typical scientific workflow

In this course we are interested in **applying image processing techniques to extract useful information from images**.

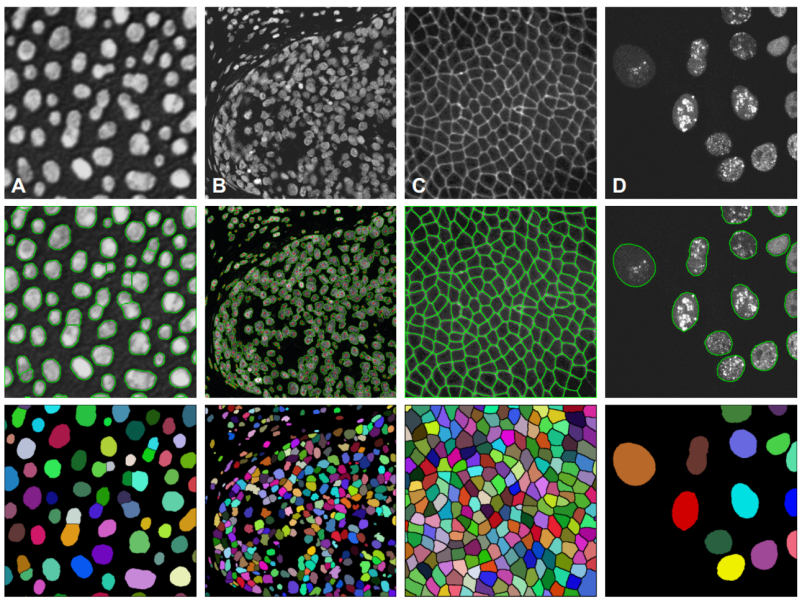


Figure 2: Example analyses of images for scientific applications

In our scientific applications, the standard workflow commences with an instrument generating a **signal**, which is then stored on a computer hard drive for subsequent manipulation. Our focus is primarily on one or more images captured through a light microscope. Depending on the nature of the data acquisition and the analytical requirements, various operations may need to be executed on the image(s) prior to extracting quantitative measurements.

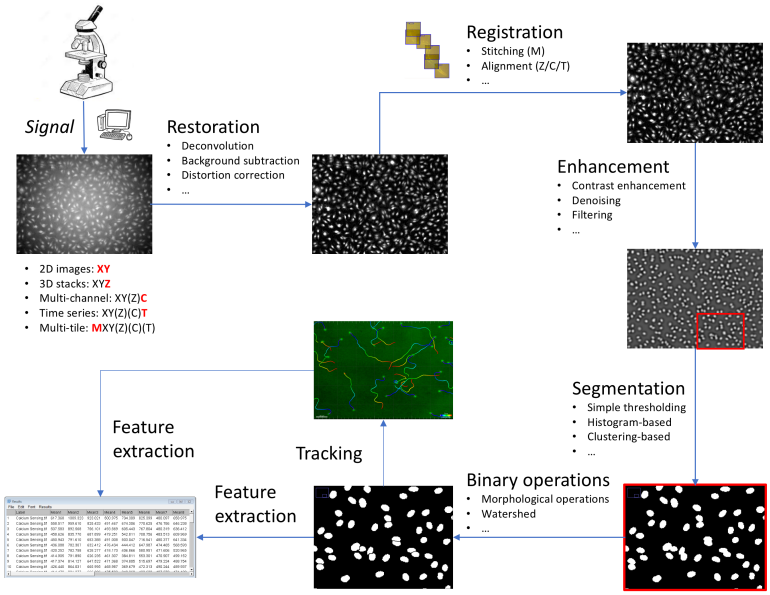


Figure 3: Typical workflow

Features of interest are isolated from pertinent **objects** identified within the images—for instance, individual cells in a microscopy image. Utilizing these measured features, they may be categorized into various distinct **classes**. Additionally, **descriptive analysis** could be conducted on certain features, such as calculating the mean size and size variance of the cells. Alternatively, **inferential analysis** may be employed to evaluate statistical hypotheses, ascertain confidence intervals for key parameters, or examine correlations between two or more of these parameters.

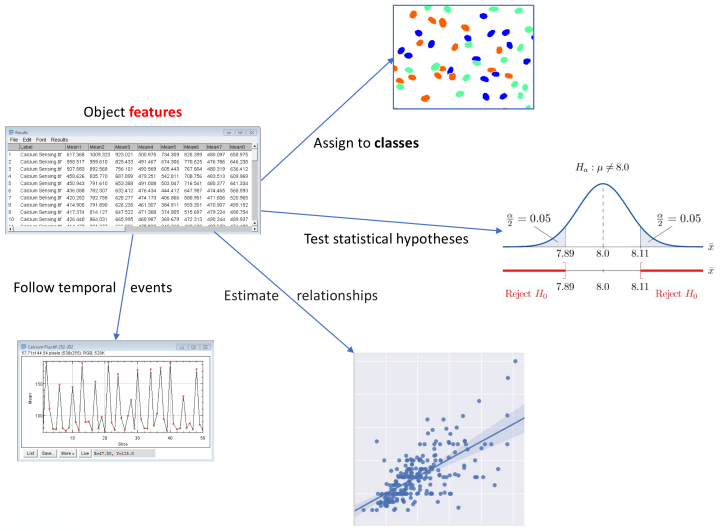


Figure 4: Examples of analysis

1.3 Image processing classes

Image processing is roughly divided into a few classes of operations. In this section, we will quickly introduce them. In the remainder of the course, we will then deepen our discussion.

1.3.1 Image enhancement

Image enhancement is the often subjective manipulation or transformation of an image with the aim of increasing its usefulness or visual appearance.

Keywords associated to image enhancement are *point-wise image operations, histogram operations*, *spatial and frequency domain filtering*, *pseudo coloring*.

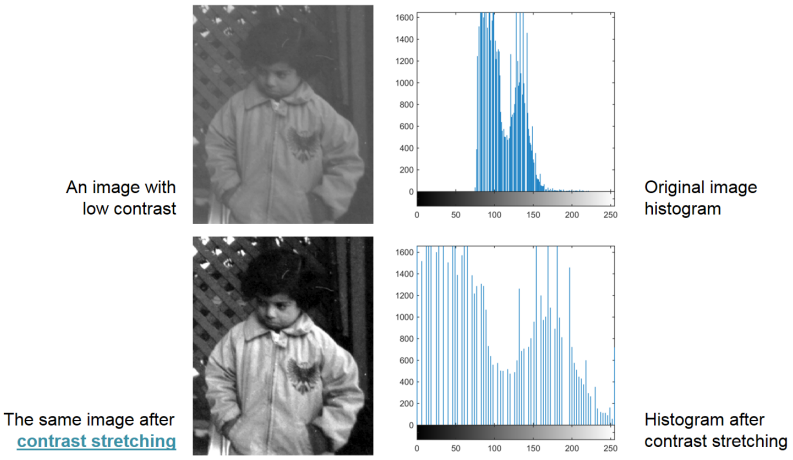


Figure 5: Image enhancement

In this example, the intensity contrast of the original picture is stretched to give a more pleasant image.

1.3.2 Image restoration

In contrast to image enhancement, **image restoration** makes use of a model of the degradation to correct the image based on some optimality criterion.

Keywords associated to image restoration are *noise smoothing*, *deconvolution*, *error concealment*, *inpainting*, *super-resolution*, *reconstruction from projections*.

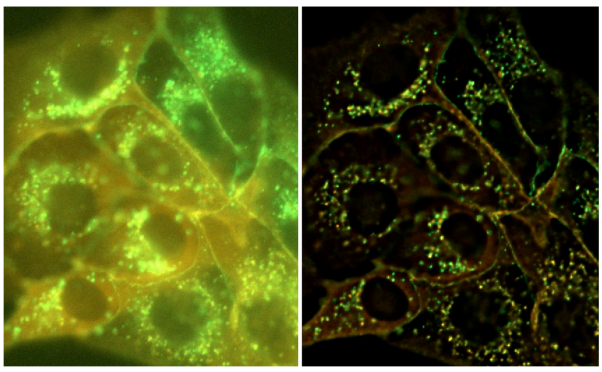


Figure 6: Deconvolution of cell-cell junctions of MDCK cells (Dr. Johan de Rooij, Hubrecht Institute, Utrecht, The Netherlands (source: https://www.svi.nl)

In this example, the blur and haze of a widefield microscopy image are corrected by **deconvolution**.

1.3.3 Image registration

Image registration is the process of transforming different sets of data into one coordinate system. Data may be multiple photographs, data from different sensors, times, depths, or viewpoints.

Keywords associated to image registration are *intensity-based*, *feature-based*, *spatial domain methods*, *frequency domain methods (phase correlation)*, *single-modality*, *multiple-modality*, *stitching*.

One example of image registration is the panorama function (or stitching) of the camera applications on our smartphones; or multiple stage positions on a light microscope. In correlative microscopy, image registration aligns (and overlays) the images taken on a light microscope to those taken on an electron microscope.

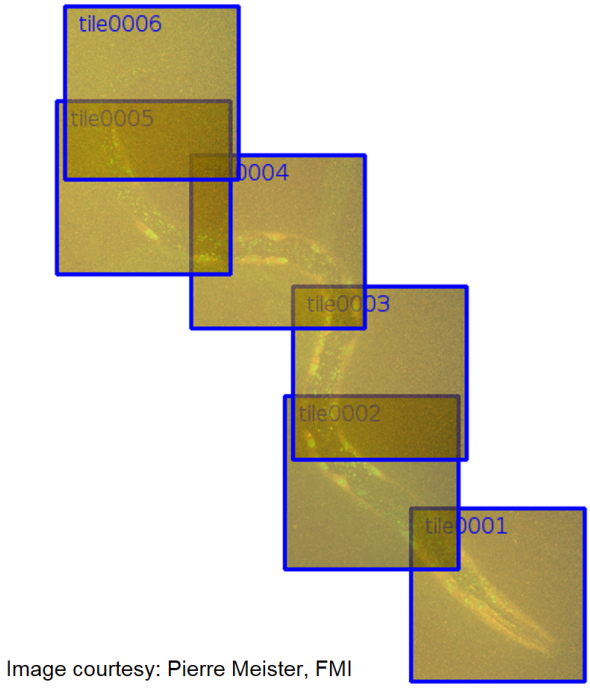


Figure 7: Image stitching

1.3.4 Image segmentation and classification

Image segmentation and **classification** are the division of an image into constituent, meaningful regions and the successive characterization of those regions.

Keywords associated to image segmentation and classification are *edge segmentation*, *thresholding*, *histogram-based segmentation*, *region growing*, *splitting and merging*, *watershed*, *K-means algorithms*, *convolutional neural networks (U-Net, ...)*, *statistical methods*, *supervised classification*.

An example of segmentation and classification could be automatically recognizing cells in a microscopy image and assigning them to the different phases of their cell cycle.

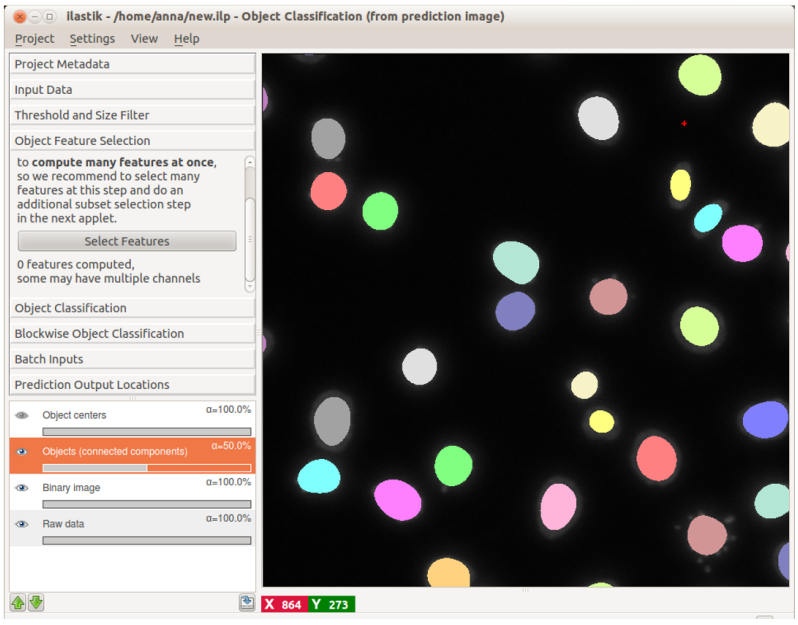


Figure 8: Image segmentation and classification

1.3.5 Image tracking

Image tracking is the process of determining motion vectors that describe the transformation from one image to another (**motion estimation**); it can also apply to extracted objects (**tracking**).

Keywords associated to image tracking are *phase correlation*, *block matching*, *optical flow*, *object tracking*, *single-particle tracking*.

Figure 9: Motion estimation and particle tracking

1.3.6 Image compression

Image compression is the minimization of the number of bits in representing an image, either in a **lossless** or **lossy** fashion.

Keywords associated to image compression are *Huffman*, *arithmetic coding*, *dictionary methods*, *PCM*, *DPCM*, *JPEG*, *MPEG*.

Resolution	Bandwidth
HDTV 1080p @ 25 fps **uncompressed**	100 - 130 MB/s (∼1Gbit/s)
HDTV 1080p @ 25 fps **compressed** (YouTube, Netflix)	0.65MB/s (∼5Mb/s) (Netflix HD requirements)

Streaming movies over the internet is mainly possible thanks to the use of (lossy) compression techniques. Without such compression, the bandwidth requirements would incur prohibitively high costs.

It's important to distinguish between the two main types of compression:

- **Lossless compression**: this category of data compression algorithms enables the **perfect reconstruction** of the original signal from the compressed data.
  - Common lossless file formats include **TIFF**, **PNG**, and specialized formats for microscopy, such as **ND2**, **CZI**, **LIF**, **LSM**, **STK**, ...
- **Lossy compression**: this approach allows for only an approximate reconstruction only of the original data, albeit with generally superior compression ratios and reduced file sizes.
  - Notable lossy file formats are **JPEG**, **JPEG2000** for images, and **MPEG-1/2/4**, **H.264**, **H.265** for video.

For scientific applications requiring quantitative analysis, it is imperative to utilize **lossless compression** exclusively.

1.3.7 Image visualization

Image visualization is the transformation, selection, or representation of data from simulations or experiments, with an implicit or explicit geometric structure, to allow the exploration, analysis, and understanding of the data. Proper visualization of image data is essential to allow its exploration, analysis and understanding. Also essential is the display of derived measurements along with the raw data.

Keywords associated with image visualization are *volume rendering*, *MIP rendering*, *surface rendering*, *look-up tables*, *color spaces*.

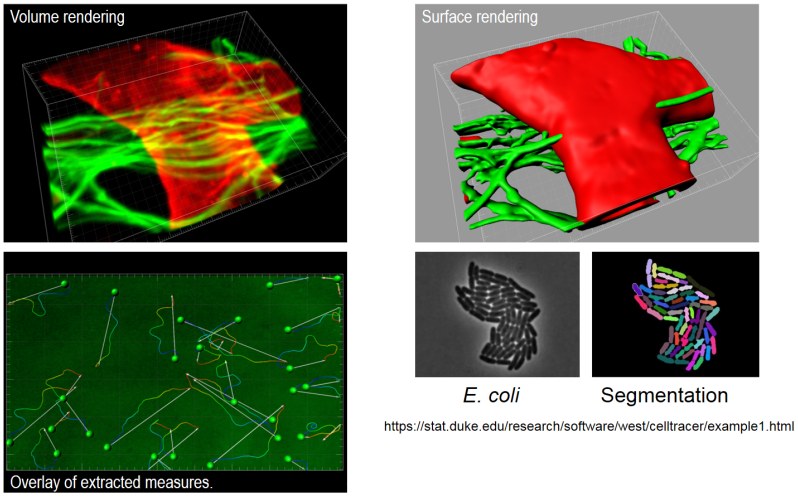


Figure 10: Some image visualization modalities

2 Signals

2.1 What is a signal?

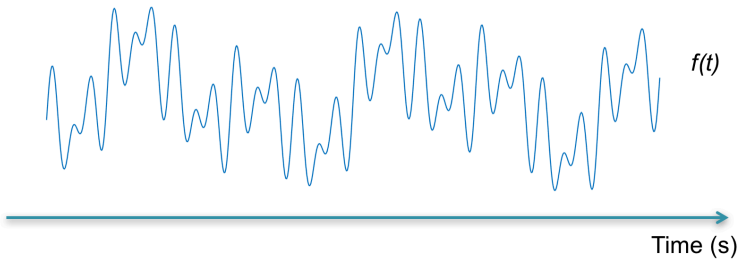


Figure 11: A signal

A **signal** is referred to as a function f conveying **information** about some phenomenon of interest. In more physical terms, a signal is a quantity exhibiting **variation** in space and/or time. The concepts of variation and information are linked in information theory, and *no variation* implies *no information*.

2.2 Analog and digital signals (1D)

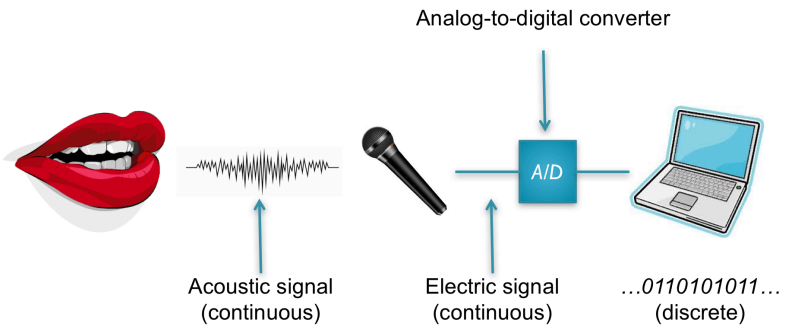


Figure 12: Analog to digital conversion

For numerous applications, it's essential to convert a real-world, **analog** (*continuous*) signal into a format amenable to computational processing or storage.

To illustrate, consider a one-dimensional example such as an audio signal.

When we speak, we generate an *acoustic* signal, which is inherently **analog and continuous**. A microphone translates these air pressure variations into an *electrical* signal, which also remains analog and continuous. This signal then undergoes a transformation via an **analog-to-digital converter**, converting it into a **numeric, discrete** form through a two-stage process that we will discuss next. This digitalized signal can now be stored on a computer hard-drive, for instance.

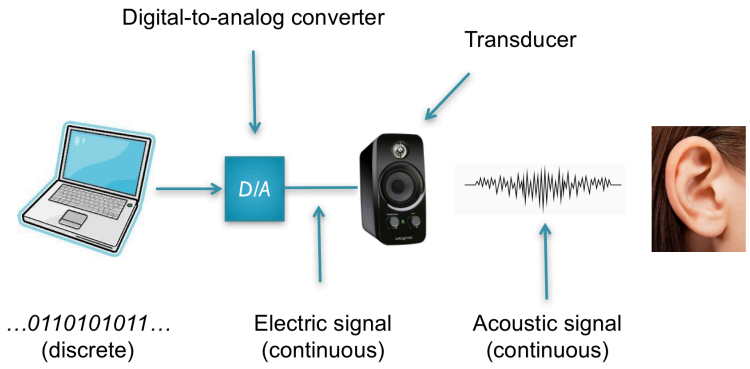


Figure 13: Digital to analog conversion

To revert the numeric representation of the signal to an audible sound, the process must be inverted.

The discrete signal is first transformed by a **digital-to-analog converter** into an electrical signal. This electrical signal is then channeled to a **transducer**, which converts the electrical current back into variations in air pressure. As a result, an audible sound is generated.

2.3 Analog-to-digital conversion

The analog-to-digital conversion is a two-step process:

- **Sampling** converts a **continuous** signal into a **discrete** one.
- **Quantization** discretizes the **amplitidue** of the signal.

2.3.1 Sampling

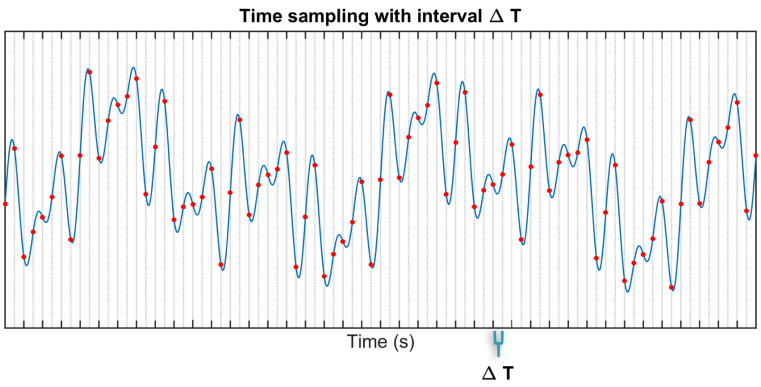


Figure 14: Sampling

Sampling refers to the act of capturing instantaneous measurements of the target signal at specific, discrete time intervals denoted by Δt. In the graph above, the continuous blue curve represents the original signal, while the red dots signify the points at which the signal is sampled at these discrete time intervals.

2.3.2 Quantization

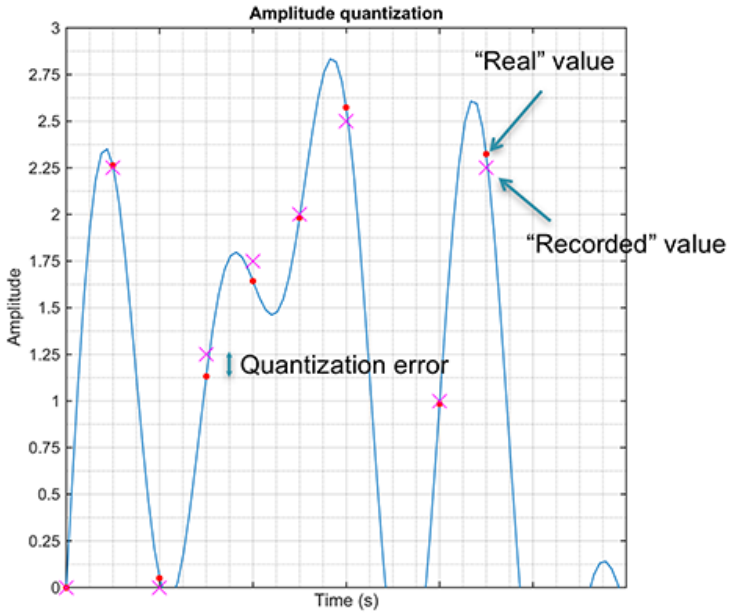


Figure 15: Quantization

The signal is not only continuous in time but also in **amplitude**. This necessitates that the amplitude must also be approximated using a **finite number** of discrete values. The discrepancy between the actual amplitude and its recorded approximation is termed as the **quantization error**.

2.4 Analog and digital signals (2D)

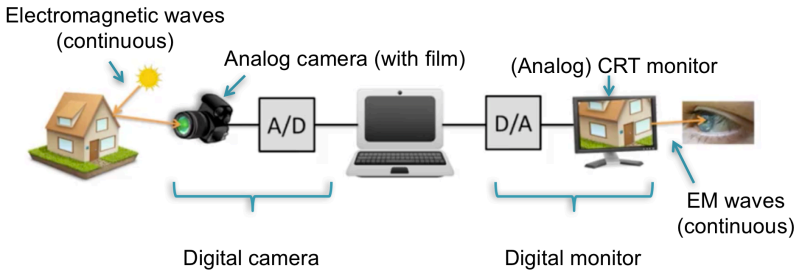


Figure 16: A 2-D signal

Two-dimensional signals largely exhibit similar characteristics to their one-dimensional equivalents.

Light, in the form of continuous electromagnetic waves, interacts with objects and is captured by an analog camera, which converts it into an electrical current. This current is subsequently sampled and quantized by an analog-to-digital converter for digital storage. To recreate the original signal, a digital-to-analog converter generates an electrical current that a CRT monitor directs towards phosphorescent pixels on its screen, thereby rendering the images we perceive. In contemporary technology, digital cameras and monitors often incorporate these converters internally.

As we progress with the course, we will discuss the topic of **fluorescence**,where the signal of interest is not merely *reflected* but is actually *generated* by the object itself.

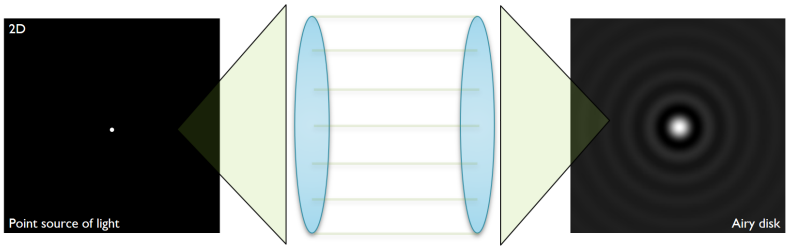


Figure 17: Fluorescence

2.5 Digital images

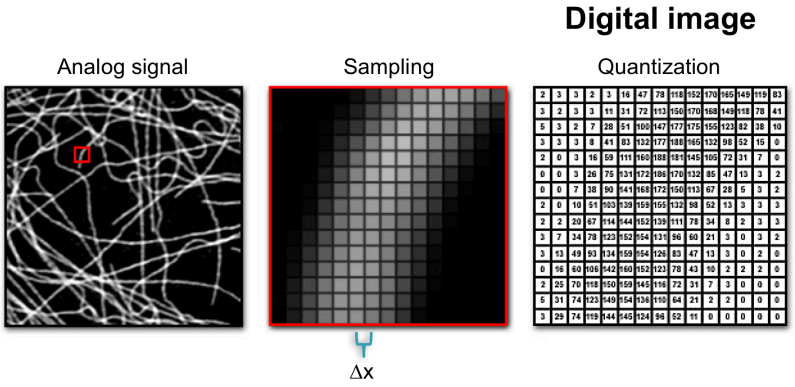


Figure 18: A digital image

An **image** is essentially the representation of brightness or color signals and can be described as a function f of a two-dimensional location (x, y). In the example above, the continuous image of a biological filamentous structure on the left panel is sampled at a spatial interval Δx in both the x and y directions. The middle panel provides a magnified view of the area enclosed by the red square on the left, illustrating how the continuous light intensities are quantized into a **fixed number of discrete values** (right panel).

The discretization process requires decisions regarding the dimensions of the image array (NxM) as well as the number of discrete values G to be used for quantization.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & ...f(0,N-1) \\ f(1,0) & f(1,1) & ...f(1,N-1) \\ ... & ... & ... \\ f(M-1,0) & f(M-1,1) & ...f(M-1,N-1) \end{bmatrix}$$

In digital image processing applications, M, N, and G are usually powers of two.

2.5.1 Pixels and voxels

In a two-dimensional image denoted by $I(x, y)$, each element within the grid or array is referred to as a **pixel**, an abbreviation for **picture element**. In numerous applications, such as biomedical imaging, three-dimensional images are frequently utilized. In such a 3D image, represented as $I(x, y, z)$, the equivalent of a pixel is termed a **voxel**, derived from **volume element**.

2.5.2 Resolution considerations

The **resolution** of an image serves as an indicator of the *fidelity* with which the original scene is represented, and it comprises two distinct aspects.

The **spatial resolution** of a digital image is intrinsically linked to the number of pixels (NxM) employed to capture the original signal. Consequently, it bears a close relationship to the applied **sampling**.

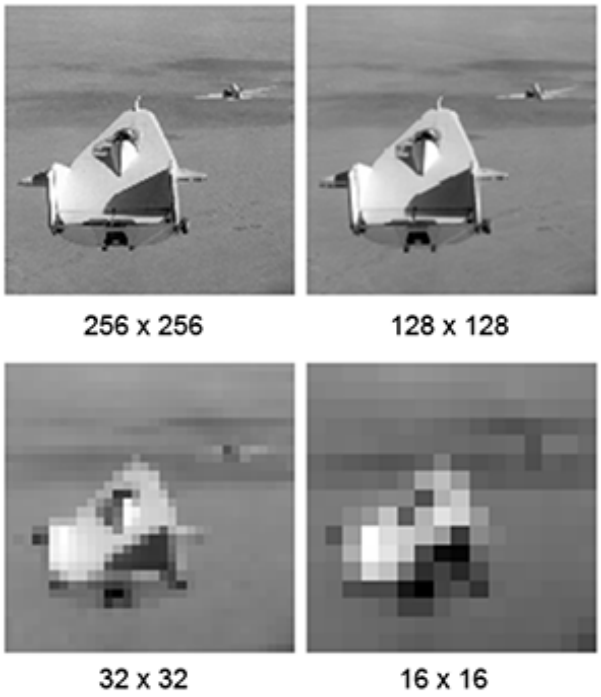


Figure 19: Spatial resolution

The **grayscale resolution** is determined by the **bit depth**, which refers to the number of **bits** (and correspondingly, the range of *gray values*, *scales*, or *levels*) used to represent the **dynamic range** of the signal. As such, it is intimately associated with the **digitization** employed.

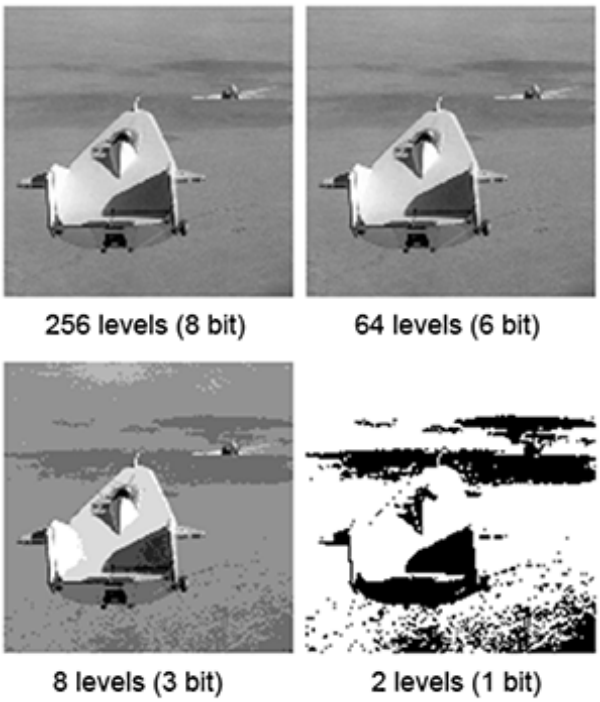


Figure 20: Grayscale resolution

The relation between bit depth b and number of gray levels G is given by $G = 2^b$.

3 Systems

Having introduced the concept of signals, we now turn our attention to **systems** that interact with these signals, focusing specifically on **linear shift-invariant systems**.

To establish some notation, we will represent a discrete signal $x(n_1, n_2)$ either as a table of numbers (as shown on the left) or within a coordinate system (as depicted on the right).

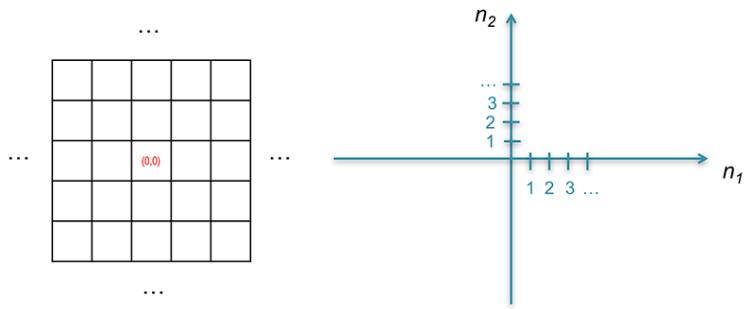


Figure 21: Signal representations

Here, $x(n_1, n_2)$ signifies the value of the signal at the coordinates (n_1, n_2), where n_1 and n_2 serve as the indices in the table or along the two orthogonal axis, respectively.

A **system** takes an input signal $x(n_1, n_2)$ and subjects it to a transformation T, resulting in an output signal $y(n_1, n_2)$.

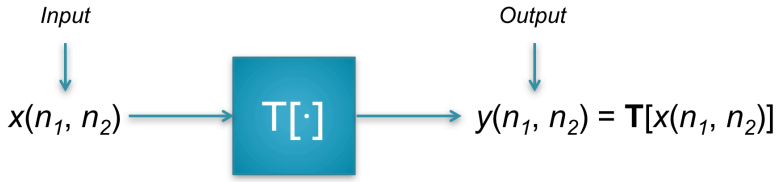


Figure 22: A system

A straightforward example of a system could be $255 - x(n_1, n_2)$ that, when applied to an 8-bit signal (see below), effectively inverts the image.

A more complex example involves a system that assigns at each position $y(n_1, n_2)$ the median value from a **neighborhood** N of pixels surrounding $x(n_1, n_2)$.

The transformation $T[]$ can encompass a wide variety alterations of the input signal $x(n_1, n_2)$.

Systems characterized by $T[]$ can exhibit a range of properties. In out course, we will focus on a specific class of systems that hold two very important and useful attributes: they are **linear** and **spatially- or shift-invariant**.

3.1 Linear systems

A system $T[]$ is deemed **linear** if it satisfies the following condition:

$$T[a_1 x_1(n_1, n_2) + a_2 x_2(n_1, n_2)] = a_1 T[x_1(n_1, n_2)] + a_2 T[x_2(n_1, n_2)]$$

In layman's terms, this equation signifies that the transformation of a sum of multiple signals is equivalent to the sum of the individual transformed signals. Alternatively, one could say that a linear system allows for its decomposition into constituent elements that can be processed independently, with the results subsequently aggregated.

3.2 Shift-invariant systems

A system is classified as **shift-invariant** if it adheres to the following principle: given a transform $T[]$, a shift of the input by (k_1, k_2) results in an equivalent shift of the output.

$$T[x_1(n_1 - k_1, n_2 - k_2)] = y(n_1 - k_1, n_2 - k_2)$$

In simpler terms, the location of the coordinate system's origin is inconsequential; for a given value of x, the resultant value of y will remain consistent.

3.3 Discrete unit impulse

Let us examine a specific type of input signal, referred to as a **discrete unit impulse**, denoted by $\delta(n_1, n_2)$.

$$\delta\,(n_1,n_2) = \begin{cases} 1, & if\ n_1 = n_2 = 0 \\ 0, & otherwise \end{cases}$$

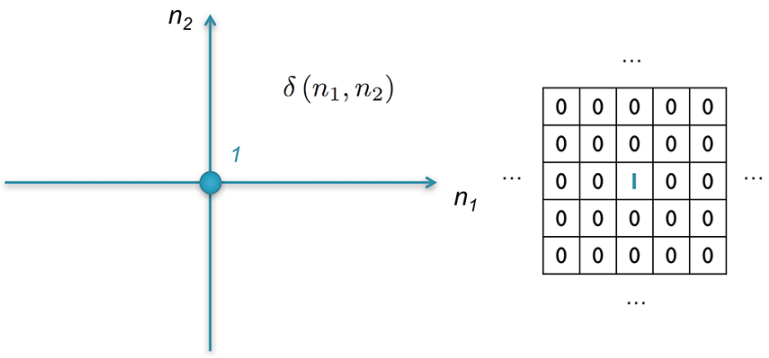


Figure 23: The discrete unit impulse

The function $\delta(n_1, n_2)$ is characterized by being zero at all points except for a single location, specifically at the origin of the axes, where it takes the value of 1.

When a discrete unit impulse is input into a system, the resultant output is known as the system's impulse response, denoted by $h(n_1, n_2)$.

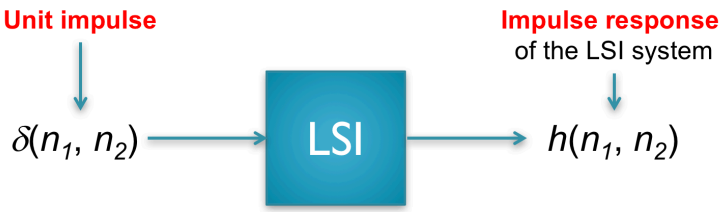


Figure 24: Impulse response of a system

One of the most salient features of linear-shift invariant (LSI) systems is that the impulse response to a unit impulse provides a comprehensive description of the system. In other words, once $h(n_1, n_2)$ is known, it becomes possible to determine the system's response to **any given input signal** $x(n_1, n_2)$.

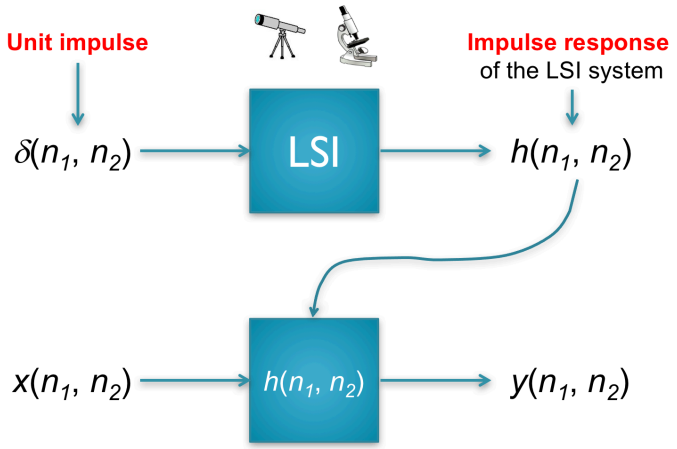


Figure 25: The impulse response characterizes the response to any signal

This concept extends beyond mere theoretical constructs; the impulse response of a system can actually be *measured* in practical settings. For example, one could assess the impulse response of the Hubble telescope, which orbits outside Earth's atmosphere, by directing it at a distant star set against the dark sky. Similarly, the impulse response of a microscope could be determined by focusing it on a single fluorescent GFP molecule within a specimen.

3.4 Convolution

LSI systems can effectively be characterized and implemented efficiently in the spatial domain via the process of **convolution**.

$$y(n_1, n_2) = x(n_1, n_2) \circledast h(n_1, n_2)$$

$$y\left(n_1, n_2\right) = x\left(n_1, n_2\right) \circledast h\left(n_1, n_2\right) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x\left(k_1, k_2\right) h\left(n_1 - k_1, n_2 - k_2\right)$$

Figure 26: Convolution

Convolution is a mathematical operation involving two functions x and h that yields a third function y. The resultant function y is the integral (or sum) of the pointwise multiplication of x and h, considered as a function of the extent to which one of the original functions is shifted.

Despite its seemingly complex nature, convolution is more approachable than it appears. To illustrate, consider a straightforward example in one dimension. It's important to note that the negative sign serves to invert the impulse response.

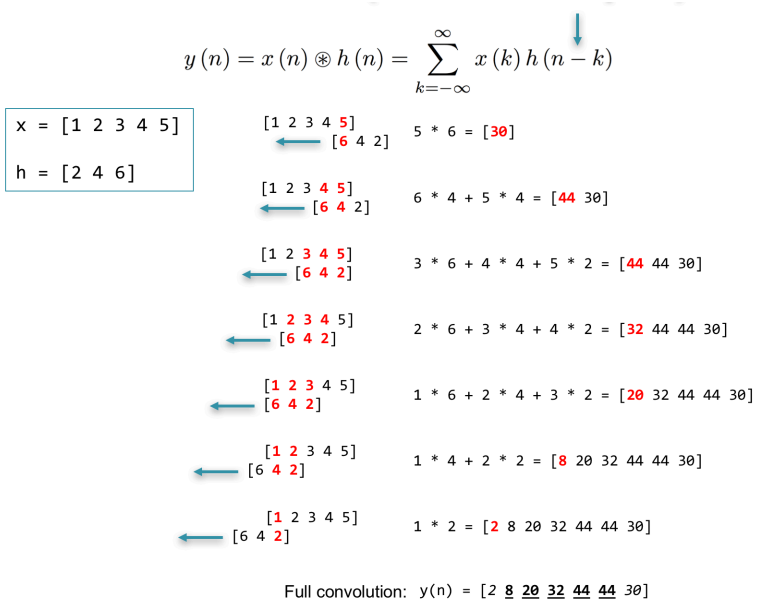


Figure 27: Convolution example

The complete convolution operation results in a size defined by `length(x) + length(h) - 1`. The portion of the convolution that fully incorporates the *support* of the impulse response corresponds to the central `length(x)` elements. The border regions, which only partially include the impulse response, have a size determined by `ceil(length(h) / 2 - 1)`.

3.5 Summary

To summarize the attributes of LSI systems, let's consider a straightforward, graphical example. Again, the impulse response of the system to the $\delta(n_1, n_2)$ function is represented by $h(n_1, n_2)$.

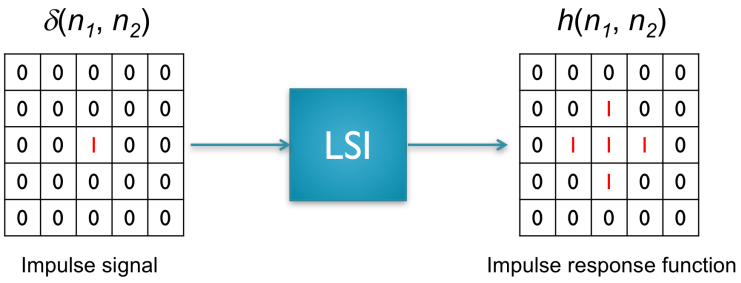


Figure 28: Impulse response of an LSI system

When the input signal $x(n_1, n_2)$ below is processed through the LSI system, the output $y(n_1, n_2)$ is generated. We've deliberately chosen a rather unremarkable input to facilitate the visualization of the LSI system's properties.

Each δ function in the input elicits its own unique response, and this response is not influenced by the spatial location of the δ function.

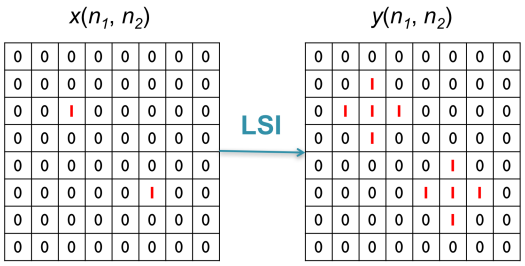


Figure 29: Shift invariant response

When the two signals are moved closer together, the resultant output is simply a linear combination (specifically, a weighted sum) of the spatially-shifted impulse response functions.

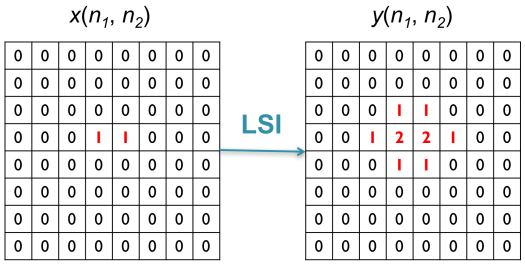


Figure 30: Linear (additive) response

4 Fundamental definition and tools

4.1 Data types

After sampling and quantization, image pixels are stored on our computer systems in various data formats. The most prevalent types of these formats are the **integer** and **floating-point** classes.

4.1.1 Integer values

Integers are stored in a binary format, where a **bit** can only assume one of two possible values: 0 or 1. A single bit is sufficient to denote whether a pixel is black (0) or white (1). To represent a broader range of numbers, bits are packaged into longer sequences. For instance, with 2 bits we can represent up to 4 values $(0,\ 1,\ 2,\ 3)$:

$$\begin{array}{l} 00 \rightarrow 0 \\ 01 \rightarrow 1 \\ 10 \rightarrow 2 \\ 11 \rightarrow 3 \end{array}$$

With 8 bits, it becomes possible to encode 256 unique values, and with 16 bits, one can encode 65536 distinct values. The number of unique integer numbers that can be encoded with b bits is 2^b, spanning a range from 0 to $2^b - 1$. This number b is referred to as the **bit depth**.

In contemporary computers, bits are typically processed in units of 8, known as **bytes**. Consequently, a pixel with an 8-bit intensity encoding will occupy 1 byte of memory, while a pixel with a 16-bit intensity encoding will occupy 2×8 bits $= 2$ bytes.

In certain representations, one bit is allocated to indicate the **sign** of the integer value being encoded. For example, an **unsigned 8 bit integer** has a range of $0 \rightarrow 255$, whereas a **signed 8 bit integer**, whicu uses 7 bits for the number and 1 bit for the sign, has a range of $-128 \rightarrow 127$. Similarly, a **signed 16-bit integer** has range $-32768 \rightarrow 32767$.

4.1.2 Floating-point values

At times, image operations require storing values that are not easily or efficiently represented as integers. For example, ratiometric values like 0.2 or 6.3, or exceedingly large numbers such as $3.2 \cdot 10^{15}$, are either infeasible to represent or would require an impractically large bit depth.

Given that the set of real numbers is infinite, a binary representation would necessitate an infinite number of bits. Therefore, a standardized approach for the representation, exchange, and computation of real numbers is essential. This standardization is provided by the **IEEE Standard for Floating-Point Arithmetic (IEEE 754)**. You can read more about it on Wikipedia: https://en.wikipedia.org/wiki/IEEE_754.

A **floating-point number** is characterized by a sign bit, an exponent field, and a mantissa. The table below outlines the bit allocation for these components, as well as the range and precision (at 1.0) for both single- and double-precision numbers.

Precision	Sign	Exponent	Mantissa	Total	Range	Precision
Single	1	8	23	32	$1.17^{-38} - 3.40^{38}$	1.19^{-7}
Double	1	11	52	64	$2.22^{-308} - 1.80^{308}$	2.22^{-16}

4.1.3 Limitation of bit representations

The necessity to use bits for representing quantized signals arises from the inherent limitations of computer systems. Specifically, computers must store continuous signals using a finite representation, both in terms of temporal or spatial dimensions and in the number of quantization levels. These constraints can lead to issues if not properly managed.

4.1.3.1 Integer overflow and underflow

Integer overflow occurs when an attempt is made to store a numerical value that exceeds the bit capacity of the designated data type. The behavior upon overflow varies depending on the programming language and, often, the specific data type in use.

MATLAB, for instance, employs a mechanism known as **clipping**. When a value surpasses the maximum limit defined by the bit depth of the image, it is truncated or "clipped" to that maximum. For example, a value of 362 would be clipped to 255 if stored in an unsigned 8-bit integer image. Conversely, negative values (for instance, -10) will experience **underflow** and be clipped to zero.

Python, by default, does not offer native support for 8- or 16-bit integer types. For such functionality, external libraries like NumPy, which is implemented in C, are commonly used. When utilizing NumPy in Python—a practice widely supported in scientific computing—**integer-wraparound** occurs. In this scenario, values exceeding the maximum limit wrap around to become smaller values ($256 \rightarrow 0$, $257 \rightarrow 1$, ...), while values that underflow wrap around to become larger values($-1 \rightarrow 255$, $-2 \rightarrow 254$, ...). Be careful when working with these data types.

4.1.4 Rounding (down)

Storing a floating-point value like 3.2 as an integer results in truncation of the fractional part, rounding the value down to 3[1]. When converting an image from an unsigned 16-bit integer format to an unsigned 8-bit integer format, a loss of precision is inevitable unless the original intensity range is already within the $0 - 255$ range. Consider an original intensity range of $10,000$ ($\min = 0$, $\max = 10000$); this range must be compressed to fit within the 8-bit limit. Given that the ratio between the two ranges is $10000/256 \approx 39$, approximately 39 distinct values in the original 16-bit image will be mapped to a single value in the 8-bit image. This compression results in a significant reduction in grayscale resolution.

4.2 Colors

The world is full of color, and we usually want our photos to be colorful too. But is color always essential for conveying the essence of an image? In scientific contexts, the answer is often no. The key information is

[1] It's important to note that instead of simply truncating the fractional part, proper rounding methods can be employed. This would shift the values by half an interval, although it would result in the maximum value being uniquely represented. This approach can mitigate some of the loss in grayscale resolution.

frequently tied to the intensity of the image, which correlates with the magnitude of a particular signal. Such images are typically stored in grayscale format, with pixel values represented as 8- or 16-bit integers, or as 32-bit or 64-bit floating-point numbers.

However, when color does play a role, we need an efficient method to digitally represent it. Given that modern computer displays can render millions of colors, it's crucial to have a compact system for encoding these hues.

4.2.1 Look-up tables (LUTs)

In biomedical imaging, it is common practice to apply false colors to intensity images while preserving the original 8- or 16-bit range. For instance, in the right panel of the figure below, a color (look-up) table was applied to a scanning electron microscopy image featuring a soybean cyst nematode and its egg. The addition of color not only enhances the visual appeal but also makes the micrograph more accessible and understandable for those without specialized knowledge of the imaged sample.

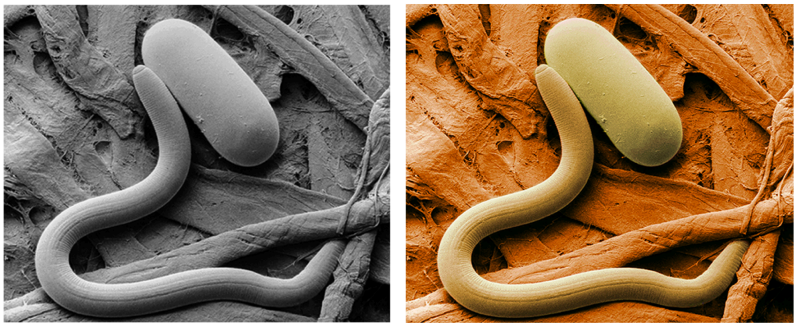
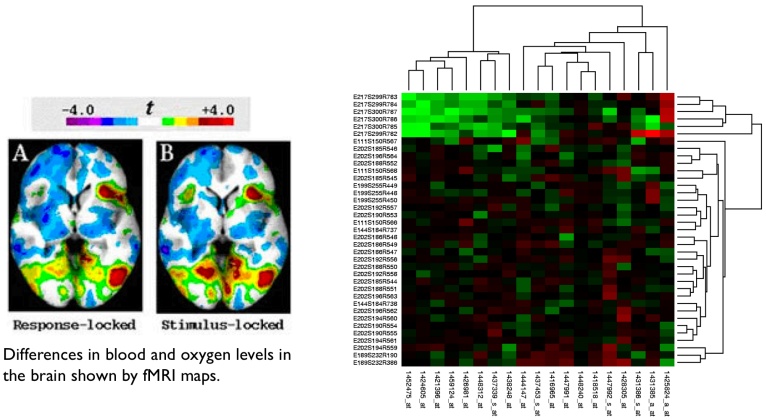


Figure 31: Lookup table for visibility

Colors also serve as a powerful tool for simplifying data interpretation by associating distinct colors with specific values or value ranges. Take, for example, the functional magnetic resonance imaging (fMRI) maps shown on the left. The use of color makes it instantly clear where in the brain pressure variations exist, and whether these variations are positive or negative. This immediate visual cue aids in quicker and more intuitive understanding of the data.



Differences in blood and oxygen levels in the brain shown by fMRI maps.

Figure 32: Lookup table for data interpretation

In the figure on the right, shades of green and red are used to indicate increased or decreased gene expression between different conditions, respectively. A black color signifies no change in gene expression. This color-coding provides an immediate visual guide for understanding variations in gene activity.

4.2.2 Multi-channel images

In scientific settings like light microscopy, **multi-channel images** are quite common. In these cases, each "channel" captures light emitted from a different fluorophore—molecules that fluoresce under certain condi-

tions. For example, GFP (Green Fluorescent Protein) emits light in the green spectrum, DAPI in the blue, Rhodamine B in the red, and YFP (Yellow Fluorescent Protein) in the yellow spectrum. When these channels are combined into a single **composite image**, it's often desirable to maintain the original colors of each channel. This not only helps in distinguishing between the different elements but also aids in studying potential colocalization, or overlapping, of these fluorescent proteins. Despite this, it's important to note that each individual channel is essentially a gray-scale image based on intensity.

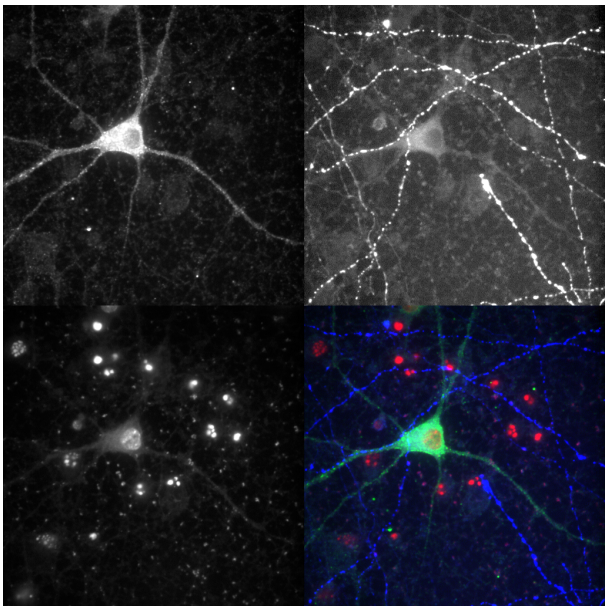


Figure 33: Composite image from three intensity images (*channels*)

4.2.3 Color spaces

A **color model** serves as a conceptual framework that outlines how colors can be represented using a set of numerical values, usually arranged as tuples of three or four components. On the other hand, a **color space** provides the specifics for interpreting these numerical components.

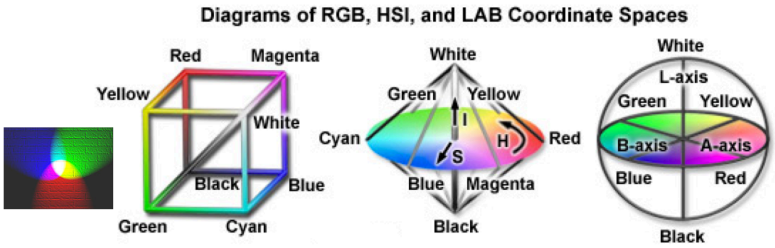


Figure 34: Common color spaces

Various color spaces exist, each with its own unique characteristics.

RGB (Red, Green, Blue): this is an additive color model primarily used in electronic displays like computer monitors and TVs. In this model, varying amounts of Red, Green, and Blue light are combined to produce a wide range of colors.

Figure 35: The RGB model. Notice that R + G + B = W (white)

HSL (Hue, Saturation, Lightness) and HSV (Hue, Saturation, Value) or HSB (Hue, Saturation, Brightness): these are cylindrical-coordinate representations of the RGB color model. They are designed to be more perceptually intuitive than the Cartesian (grid-based) representation of RGB.

Lab (CIELAB): this color space aims to be perceptually uniform, meaning that the perceptual difference between colors is consistent across the color space. The 'L' component in Lab closely aligns with human perception of lightness, making it a good approximation of how humans see color.

Each of these color spaces has its own advantages and is suited for specific applications, from digital media creation to scientific research.

4.3 Image histogram

The **histogram** of a gray-scale image serves as a statistical representation that shows the distribution of gray levels across the image. In this representation, the x-axis corresponds to the different gray levels, ranging from 0 to $K-1$ (where K is the number of possible gray levels), and the y-axis represents the frequency of each gray level, ranging from 0 to the total number of pixels $M \times N$ in the image.

The **frequency of occurrence** $H_f(k)$, in the plot below, tells us how many pixels in the image have a gray level of k. As we already discussed, the **number of gray levels** in an b-bit image is $K = 2^b$. For an b-bit image, $K = 256$.

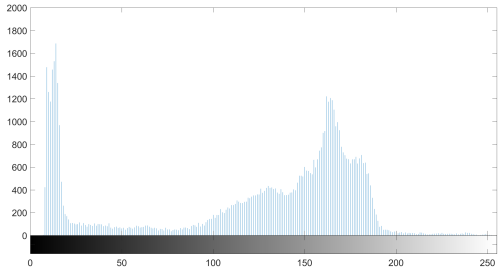


Figure 36: Image histogram

The histogram is a crucial tool for **image enhancement** for several reasons.

Insight into Image Characteristics: the shape of the histogram can provide quick insights into the image's characteristics. For example, if most of the data is concentrated on the left side of the histogram, the image is likely dark. Conversely, if the data is skewed to the right, the image is likely bright.

Contrast: a histogram that is spread out across the gray levels usually indicates good contrast in the image.

Dynamic Range: this is the range between the darkest and brightest parts of the image. A wider dynamic range usually means a more detailed image.

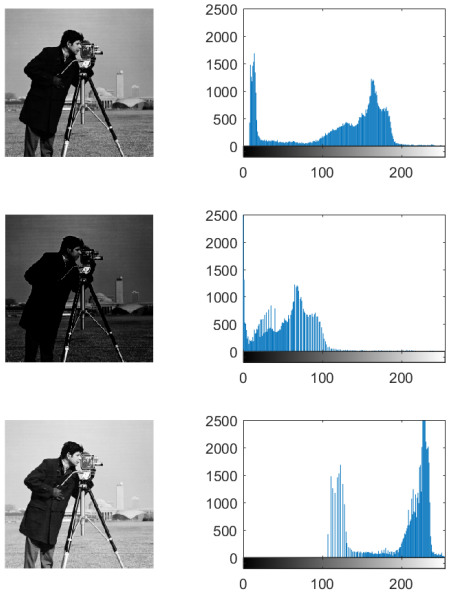


Figure 37: Histogram comparisons

For color images, particularly those in the RGB color model, the concept of a histogram extends naturally.

RGB Histogram: each of the Red, Green, and Blue channels has its own histogram, allowing for a more nuanced understanding of color distribution within the image.

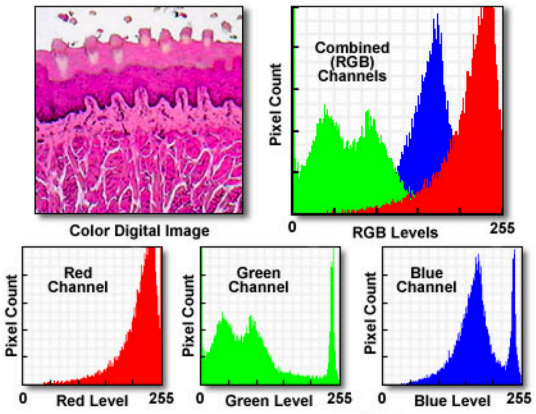


Figure 38: The RGB histogram(s)

4.4 Point operations

In digital imaging, **point operations** serve as a fundamental class of image transformations. These operations apply a function h to each pixel in the image $f(\boldsymbol{n})$ to produce a new image $g(\boldsymbol{n})$. Here, $\boldsymbol{n}$ represents the coordinates of a pixel or voxel in the image, which can be $\boldsymbol{n} = [x, y]^T$ in 2D, or $\boldsymbol{n} = [x, y, z]^T$ in 3D.

Mathematically, a point operation can be expressed as $g(\boldsymbol{n}) = h[f(\boldsymbol{n})]$. This operation alters the gray-value distribution of the image but doesn't affect the spatial arrangement of objects within the image. In essence, point operations serve as a **system** or **transformation** that maps the input image $f(\boldsymbol{n})$ to the output image $g(\boldsymbol{n})$.

4.4.1 Linear point operations

Linear point operations are a specific type of point operation defined by a linear equation: $g(\boldsymbol{n}) = S * f(\boldsymbol{n}) + o$, where S is a scaling factor and o is an *offset*. The scaling factor S stretches or compresses the histogram of the image, while the offset o shifts it.

4.4.1.1 Image negative

The **image negative** is a special case of a linear point operation where the histogram of the image is inverted. It is defined as: $g(\boldsymbol{n}) = -1 * f(\boldsymbol{n}) + (K-1)$, where $S = -1$ and $o = (K-1)$, with K being the number of possible gray levels in the image.

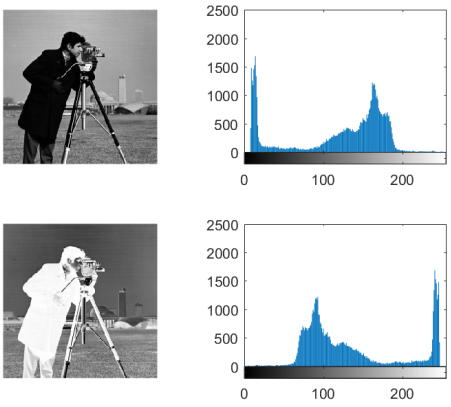


Figure 39: Image negative

The image negative operation flips the histogram, making dark areas light and light areas dark. This can be useful for enhancing certain features in images, especially in medical imaging or other fields where contrast is crucial.

4.4.1.2 Full-scale histogram stretch

The **full-scale histogram stretch**, also known as **contrast stretch**, is a specific type of linear point operation aimed at enhancing the contrast of an image. This operation expands the histogram of the image to span the entire range of possible gray levels $0, ..., K-1$.

Given that $A = \min(f(\boldsymbol{n}))$ is the minimum gray level and $B = \max(f(\boldsymbol{n}))$ is the maximum gray level in the image $f(\boldsymbol{n})$, the full-scale histogram stretch is defined as $g(\boldsymbol{n}) = \frac{K-1}{B-A} f(\boldsymbol{n}) + (-A\frac{K-1}{B-A})$, where $S = \frac{K-1}{B-A}$ and $o = -A\frac{K-1}{B-A}$.

An equivalent but simpler expression for the full-scale histogram stretch can be written as: $g(\boldsymbol{n}) = (K-1)\frac{f(\boldsymbol{n})-A}{B-A}$.

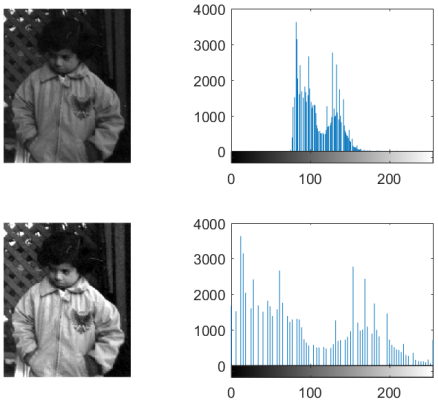


Figure 40: Full-scale histogram stretch

This operation is particularly useful for images with poor contrast, where the histogram is concentrated within a narrow range of gray levels. By stretching the histogram across the full range, the contrast of the image is significantly improved, making it easier to discern details.

4.4.2 Non-linear point operations

Non-linear point operations involve a non-linear function h that transforms the pixel values of the input image $f(\boldsymbol{n})$ to produce the output image $g(\boldsymbol{n})$. These types of operations are particularly useful for specialized image enhancement and manipulation tasks, as they offer more flexibility than linear operations in shaping the output.

4.4.2.1 Logarithmic point operations

The **logarithmic point operation** is a combination of both nonlinear and linear functions. Specifically, it first applies a logarithmic transformation to the pixel values of the image, and then follows it up with a full-scale histogram stretch. This operation serves to compress the image's histogram in a non-linear fashion.

Mathematically, it can be represented as $g(\boldsymbol{n}) = FSHS(log[1 + f(\boldsymbol{n})])$.

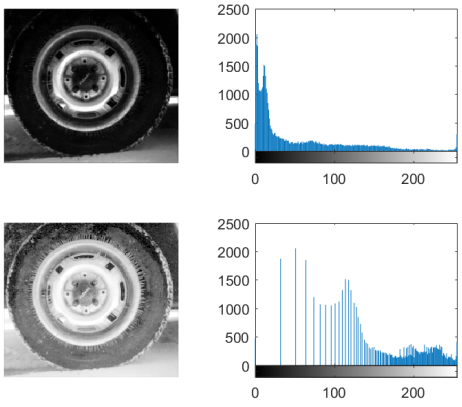


Figure 41: Logarithmic point operation

A logarithmic point operation is commonly used on the Fourier transform of an image to enhance the visibility of its high-frequency components. This makes it easier to analyze and interpret the frequency spectrum of the image.

4.4.2.2 Histogram equalization

In a manner similar to full-scale histogram equalization, **histogram equalization** (also known as **histogram leveling**) aims not only to span the entire gray-scale range but also to redistribute pixel intensities to achieve a histogram that is as uniformly distributed as possible.

To fully grasp histogram equalization, we need to expand upon the concept of the histogram. We introduce the **normalized image histogram**, denoted as p_f, which is defined as $p_f(k) = \frac{1}{NM} H_f(k)$, for $k = 0, ..., K-1$, where NM represents the total pixel count in the image. The value $p_f(k)$ tells us the proportion of pixels in the image with the gray value k.

The function $p_f(k)$ satisfies the condition $\sum_{k=0}^{K-1} p_f(k) = 1$ and can be viewed as the **probability mass function** of the gray levels in the image f.

From $p_f(k)$, we can calculate the **cumulative probability mass function**, denoted as $t(k)$, using the equation $t(k) = \sum_{j=0}^{k} p_f(j)$.

The transformed histogram $s(k) = T[p_f(k)]$ of the equalized image $g(\boldsymbol{n})$ is then obtained using the formula (that we do not need to derive in this course): $s(k) = (K-1)t(k) = (K-1)\sum_{j=0}^{k} p_f(j)$.

The values from $s(k)$ are subsequently used to replace the original intensity values k in the resulting image. The entire histogram equalization algorithm is given by the following pseudo-code:

```
# Given the image I with normalized histogram P and size M (rows) x N (cols).
# L is the max intensity (L = 255 for 8 bits). We calculate the equalized image H.

# Cumulative histogram
T[0] = P[0]
for k = 1 to L
    T[k] = T[k - 1] + P[k]

# Equalized histogram S
for k = 0 to L
    S[k] = round(L * T[k])

# Calculate the equalized image intensities -> final image H
for y = 0 to (M - 1)
    for x = 0 to (N - 1)
        r = I[y, x]
        H[y, x] = S[r]
```

Given that image intensities are typically represented as integer values, the equalized intensities $s(k)$ are **rounded** to the closest integer. This rounding is also practically necessary for array indexing.

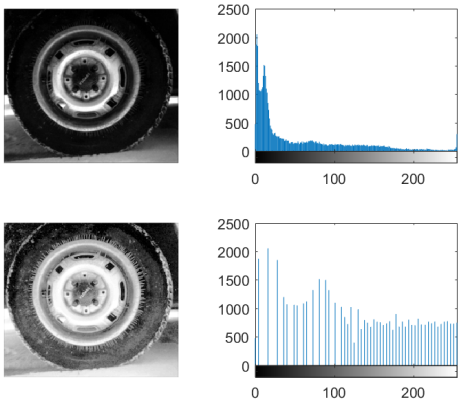


Figure 42: Histogram equalization

Histogram matching is a more advanced form of histogram equalization that adjusts the image intensities to align with a specific target **probability distribution**. In contrast, histogram equalization aims to match the **uniform distribution**.

4.4.3 Arithmetic operations

Arithmetic operations are point-based operations, meaning they are applied pixel-by-pixel, involving multiple images. Given a set of images f_1, f_2, f_3, ..., f_n, **point-wise image addition** is defined as $f_1+f2+f_3+...+f_n$. The **difference** between any two images f_i and f_j is denoted as f_i-f_j. The **point-wise product** is represented as $f_1 \cdot f2 \cdot f_3 \cdot ... \cdot f_n$, and the **point-wise image quotient** as f_i/f_n, assuming that $f_n \neq 0$ for all pixels.

4.4.3.1 Example: image averaging for noise reduction

For images suffering from a low **signal-to-noise ratio** (SNR), averaging multiple captures can serve as an effective method for reducing noise while preserving the original signal. The most prevalent form of noise is **additive**, where the captured image f consists of both *signal* (s) and *noise* (n) components that linearly combine as $f = s + n$. The noise n is characterized by a **zero mean** and **some level of fluctuation** around zero. If the signal remains constant across multiple captures, averaging the pixels won't alter the signal value but will diminish the noise impact. Because the noise measurements are zero-mean and independent, $\frac{1}{k}\sum_{j=1}^{k} n_j \approx 0$ for larger and larger number of images k.

Hence, the image with reduced noise can be calculated as $\hat{f} = (f_1 + f_2 + f_3 + ... + f_k)/k)$, where each $f_i = s_i + n_i$.

4.4.3.2 Example: image masking

In the case of two images f_1 and f_2, suppose f_1 is a binary image resulting from segmentation (where the background pixels have an intensity of 0 and the object pixels have an intensity of 1) and f_2 is a gray-value image. Performing point-wise multiplication $f_1 \cdot f_2$ will retain the original intensity values of the objects in the image, while effectively setting the pixel intensities outside of these segmented objects to 0.

4.5 Pixel neighborhoods

Transformations or systems can also operate on **neighborhoods of pixels** rather than individual pixels. In this approach, the impact of the transformation on a specific pixel is determined not only by the pixel's own value but also by the values of its surrounding neighbors. In two dimensions, the commonly used neighborhoods are 4-connected and 8-connected, while 6-connected neighborhoods are less frequent. In three dimensions, the neighborhoods can be 6-connected, 18-connected, or 26-connected.

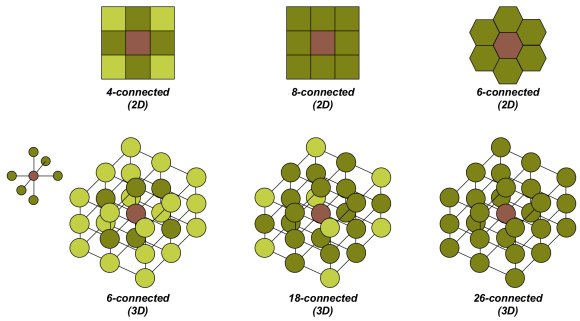


Figure 43: Pixel neighborhoods

4.6 (Convolution) kernels

The concept of a **(convolution) kernel** serves as an advanced form of pixel neighborhood and plays a crucial role in image filtering. The kernel is a compact matrix of numerical values, or weights, that is used to convolve with an image to produce a filtered version of that image.

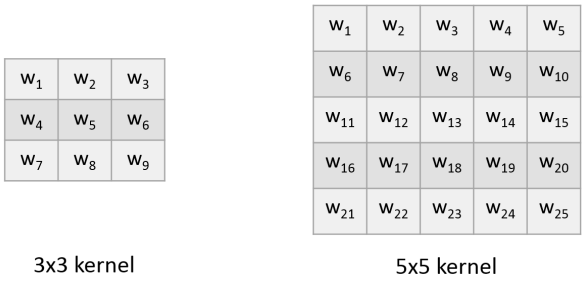


Figure 44: Convolution kernels

5 Filtering in the spatial domain

Filtering is often employed for the purpose of **image enhancement**. Common applications include *smoothing* an image to reduce noise or irregularities while maintaining the integrity of the signal. Other uses involve *sharpening*, *contrast amplification*, *target-matching*, and *feature accentuation*.

The term **spatial domain** pertains to the actual image plane, and methods operating in this domain directly modify the image's pixels. In contrast, a *transform domain* initially changes the image into an alternative representation, which is then manipulated; the modified image is reverted back to the spatial domain through an inverse transform. The **frequency domain** is as very important example of transform domain. It is important to mention that there is a very close correspondence between the spatial and the frequency domain, but the details of this correspondence are beyond the scope of this introductory course.

A spatial filter is composed of a **neighborhood** and a **specified operation** that is executed at each pixel location in the original image. For each pixel coordinate (n_1, n_2), the operation is carried out on the pixels within the neighborhood N, and the outcome is placed at the corresponding position (n_1, n_2) in the resulting image.

If the operation performed on the neighborhood is **linear**, the filter is termed a **linear spatial filter**. If not, the filter is categorized as **non-linear**.

5.1 Linear filtering through convolution

In the spatial domain, linear filters are executed through the process of **convolution**. Given an image $f(x, y)$ with dimensions A x B and a **kernel** (or **mask**) $h(x, y)$ with dimensions C x D, the convolution is mathematically expressed as:

$$f(x, y) \circledast h(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n).$$

Here, $M = A + C - 1$ and $N = B + D - 1$ determine the dimensions of the complete convolution. Typically, the border regions with incomplete kernel support are omitted, retaining only the central $A \times B$ pixels from the convolution outcome.

To create a $C \times D$ spatial filter mask h, one must define its $C \cdot D$ mask coefficients. These coefficients dictate the series of multiplications executed within the vicinity of each pixel in $f(x, y)$, thereby determining the filter's operation.

The **size** of the kernel establishes the **support** of the filter. Larger kernels exert a more pronounced filtering effect on the image compared to their smaller-sized counterparts

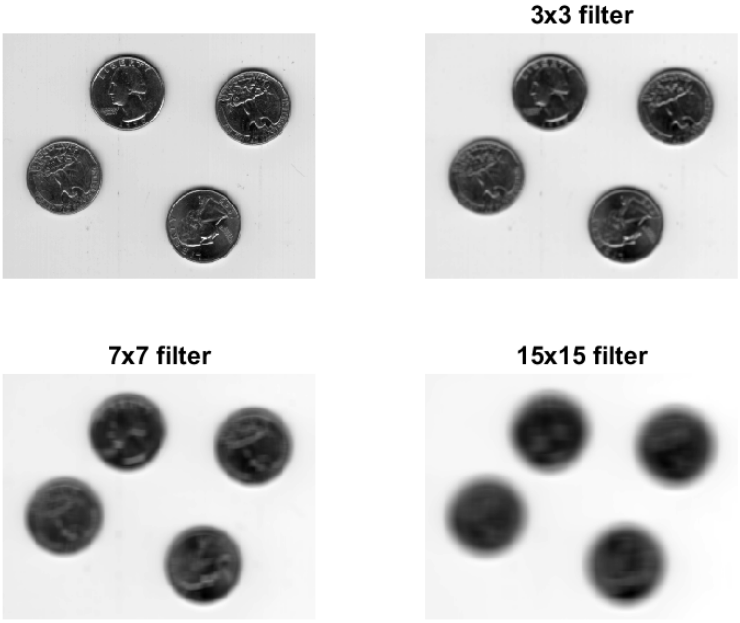


Figure 45: Effect of kernel size

5.1.1 Average filter

The **average filter** computes the mean intensity of all pixels within a neighborhood for each pixel pixel (x, y) in the input image f. This mean value is then stored at the corresponding position (x, y) in the output image g.

For a 3×3 neighborhood surrounding the pixel (x, y), the average of the 9 pixel values is determined as follows:

$$\frac{1}{9}\left(f(x-1,y-1)+f(x-1,y)+f(x-1,y+1)+f(x,y-1)+\cdots+f(x+1,y)+f(x+1,y+1)\right)$$

This averaging operation can be accomplished for each position (x, y) in the image by a convolution with a 3×3 kernel defined as:

$$h = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

In cases where the weights within the kernel vary, the result is a **weighted average** filter.

The illustration in the preceding section displays the original image in the top-left corner, accompanied by three versions that have undergone average filtering. These filtered images employ kernel sizes 3×3, 7×7, and 15×15.

5.1.2 Gaussian filter

Unlike the average filter, the **Gaussian filter** excels at maintaining features of specific scales or sizes. It also performs more favorably in the frequency domain, although we won't delve into that aspect in this course. The weights for the Gaussian kernel are determined using the equation:

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

When $\sigma = 1.0$, this results in the following $5x5$ kernel h:

$$h = \begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix}$$

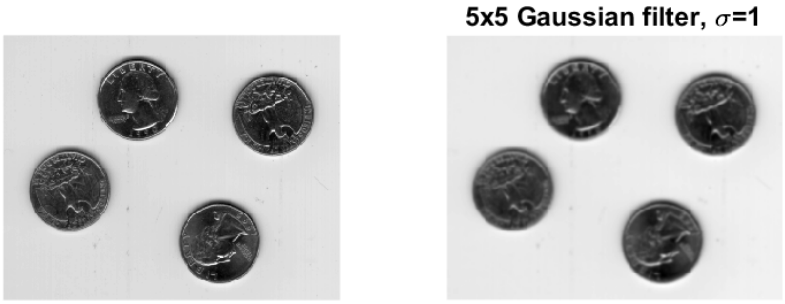


Figure 46: Gaussian filter

5.1.3 Sidebar: understanding image derivatives

In the realm of discrete functions, derivatives are conceptualized through the lens of **differences**. While there are multiple ways to define a discrete derivative, certain fundamental properties must hold true:

- A **first-order derivative**, also known as the **gradient**, should be zero in regions of uniform intensity. Conversely, it should be non-zero at the beginning and throughout intensity ramps.
- A **second-order derivative** should also be zero in areas of constant intensity, non-zero at the start and end of intensity ramps, and zero along ramps with a constant slope.

A straightforward definition of a first-order derivative for a one-dimensional discrete function that meets these criteria is:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

For the second-order derivative, the equation becomes:

$$\frac{\partial^2 f}{\partial x^2} = [f(x+1) - f(x)] - [f(x) - f(x-1)] = f(x+1) + f(x-1) - 2f(x)$$

In the context of a two-dimensional image, there's also a second derivative in the *y*-direction, defined as:

$$\frac{\partial^2 f}{\partial y^2} = f(y+1) + f(y-1) - 2f(y)$$

5.1.4 Sobel filter

The **Sobel Operator** serves as a method for approximating the first derivative of an image in both x and y directions. This is achieved by convolving the image with two distinct kernels:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The **magnitude** of the gradient is computed as:

$$G = \sqrt{G_x^2 + G_y^2}$$

Additionally, the **direction** of the gradient can be determined by:

$$\Theta = \arctan(\frac{G_y}{G_x})$$

The amplitude of the gradient gives a strong response for **edges** in the image.

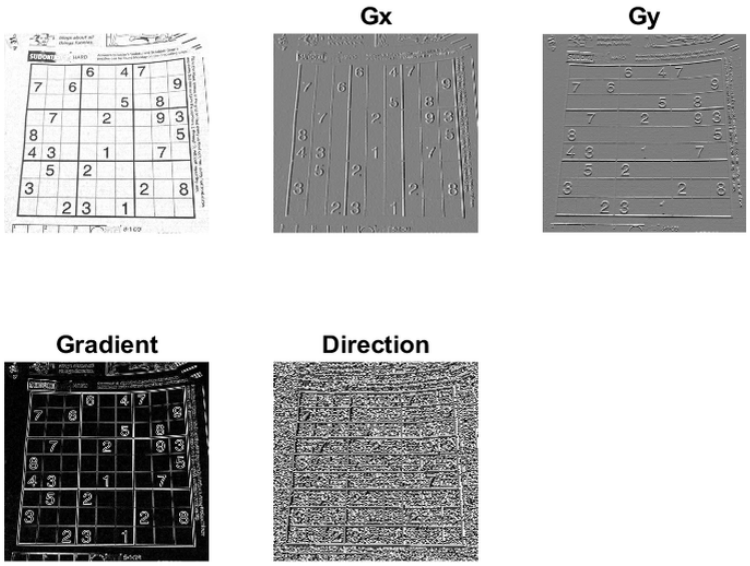


Figure 47: Gradients, gradient magnitude and direction

5.1.5 Laplacian filter

The **Laplacian Filter** is designed around the concept of a second-order derivative, specifically the Laplacian ∇^2 of a two-variable function $f(x, y)$. The Laplacian is mathematically defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Utilizing the definitions from the previous section, this can be expressed as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

The above equation can be implemented using the following convolution kernel:

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Convolution with this kernel yields the second-order derivative of the image along both x and y axes.

To include diagonal directions, two additional terms can be added to the equation, resulting in the following kernel:

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In practical applications, the sign of the Laplacian is often inverted, as shown below:

$$h = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

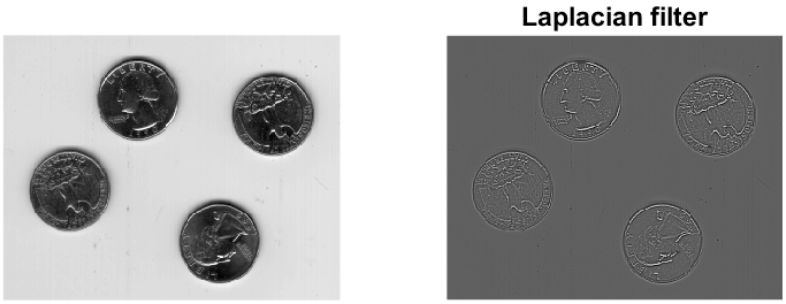


Figure 48: Laplacian filter

Applications of the Laplacian filter are in edge detection, image sharpening, texture analysis, noise reduction, object recognition, and others.

5.1.6 Laplacian of Gaussian filter

The Laplacian filter, while effective for edge detection, is highly susceptible to image noise. Minor fluctuations in pixel intensity can trigger a pronounced response from the Laplacian operator. To mitigate this, it's common to precede the Laplacian with a Gaussian filter to dampen the noise. In practice, these two filters are often combined into a single operation known as the **Laplacian of Gaussian (LoG)** filter.

The integration with the Gaussian kernel offers an additional benefit: by adjusting the σ parameter, we can control the **scale** of the filter. In other words, we can specify the object size that will elicit a strong response from the Laplacian operator. Hence, the LoG filter becomes an effective **blob detector**.

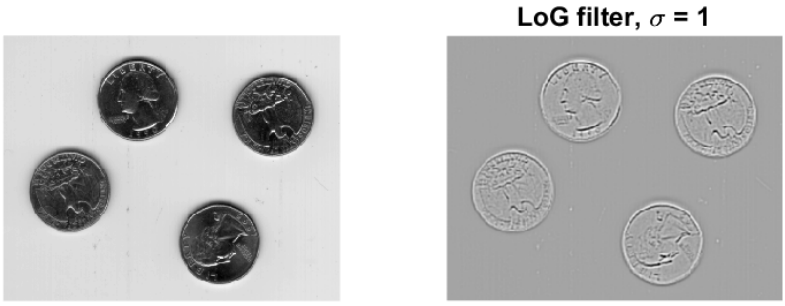


Figure 49: Laplacian of Gaussian filter

5.1.7 Difference of Gaussians filter

The **Difference of Gaussians filter** (**DoG**) employs a kernel derived from the difference between two Gaussian kernels with distinct σ values. When the ratio between σ_2 and σ_1 is close to 1.6, this DoG kernel serves as an approximation to the Laplacian of Gaussian filter with $\sigma = 1$.

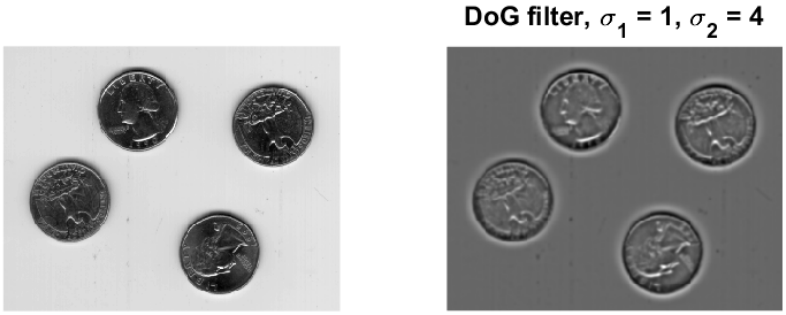


Figure 50: Difference of Gaussians filter

5.2 Non-linear filtering

5.2.0.1 Rank-based filters

Rank-based filters, also known as **order filters**, are a class of non-linear spatial filters. These filters operate by sorting the pixel intensities within a given neighborhood and selecting the pixel intensity that corresponds to a specific rank.

5.2.1 Median filter

Among the rank-based filters, the **median filter** is perhaps the most well-known. This filter replaces each pixel with the median value of the intensities in its neighborhood. The median is the middle value in the sorted list of pixel intensities, ranging from the lowest to the highest. Unlike average filters, median filters are highly resilient to **outliers**.

One of the key applications of the median filter is in the elimination of **salt-and-pepper noise**, making it highly effective for image noise reduction.

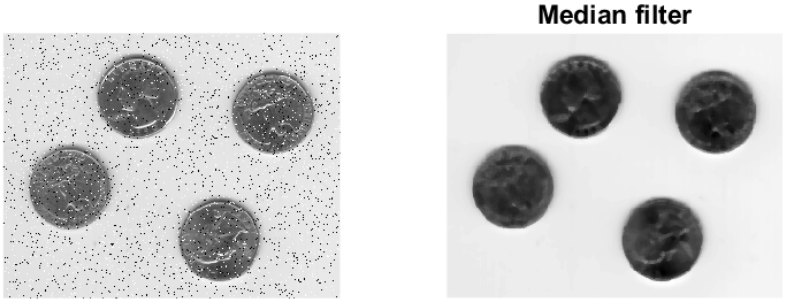


Figure 51: Median filter on salt-and-pepper noise

5.2.2 Minimum intensity filter

The **minimum intensity filter**, commonly referred to as the **min filter**, selects the lowest pixel intensity value within the specified neighborhood for each pixel in the image.

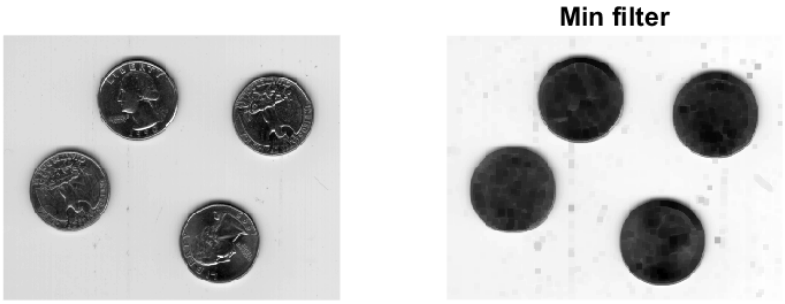


Figure 52: Min filter

5.2.3 Maximum intensity filter

The **maximum intensity filter**, often called the **max filter**, identifies the highest pixel intensity value within the designated neighborhood for each pixel in the image.

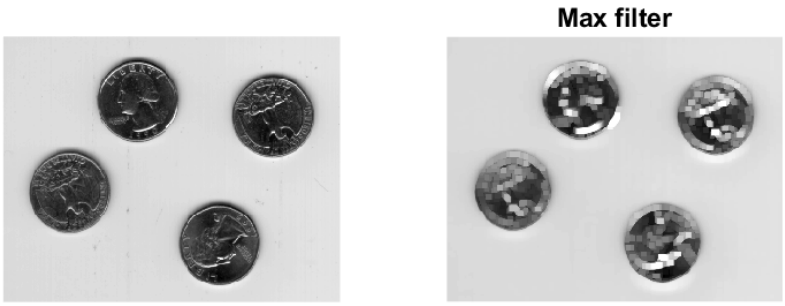


Figure 53: Max filter

6 Filtering in the frequency domain

In the early 19th century, **Jean Baptiste Joseph Fourier** (1768 - 1830) introduced the groundbreaking idea that any **periodic function** could be represented as a **sum of sines and cosines of varying frequencies**, each scaled by a unique coefficient. This summation is now known as a **Fourier series**. Furthermore, even **non-periodic functions** with finite area under their curve can be expressed as an **integral of sines and cosines, each multiplied by a weighting function**. This integral is termed the **Fourier transform**. Both the Fourier series and Fourier transform allow for a complete **reconstruction** of the original function through an **inverse process**, without any loss of information.

When a signal is represented in terms of its Fourier series or Fourier transform, it is said to be in the **frequency domain**, as opposed to the **time domain**. In the time domain, a plot of the signal reveals its temporal variations, with time as the x-axis. Conversely, in the frequency domain, the plot illustrates the distribution of the energy of the signal across various frequency bands, with frequency as the x-axis.

For discrete images, such as those captured by a microscope and digitized, the term **spatial domain** is used instead of the time domain. This is because the focus is on spatial variations between pixels rather than temporal changes. Mathematically, the same concepts and notations apply, albeit extended to two or more dimensions. To transition these discrete, spatially-defined images to the frequency domain, the **discrete Fourier transform** is employed.

What benefits does frequency-domain representation offer? The **convolution theorem** posits a direct equivalence between spatial-domain filtering via convolution and frequency-domain filtering through pixel-wise

multiplication of Fourier-transformed signals and kernels. The result of this multiplication in the frequency domain is then **inverse Fourier transformed** to yield the same outcome as spatial-domain convolution.

Frequency-domain filtering offers several advantages. Firstly, designing filters to target specific frequencies is often more straightforward in the frequency domain than crafting the corresponding convolution kernel in the spatial domain. Secondly, the convolution theorem enables more efficient computations for large images and kernels, thanks to highly optimized implementations of the discrete Fourier transform. Speed-ups can be substantial, sometimes by several orders of magnitude.

It's important to note, however, that only **linear filters** can be implemented in the frequency domain. Given the expansive nature of this subject, it will not be covered further in this course.

7 Segmentation

Segmentation serves as a cornerstone in the realm of image processing, often initiating a cascade of analytical steps. The primary objective of segmentation is to isolate regions of interest within an image for further examination and manipulation. Specifically, segmentation subdivides a digital image into **segments** or **super-pixels**, which are uniform with respect to certain attributes such as luminosity, color, texture, or motion. These segments typically represent either **objects** or their **boundaries** within the image.

Over the years, a myriad of general-purpose algorithms and techniques have emerged to tackle the challenges of image segmentation. However, the absence of a one-size-fits-all solution necessitates the integration of **domain-specific knowledge** to effectively address segmentation challenges within a particular field. Consequently, a priori information becomes indispensable for tailoring segmentation algorithms to accurately delineate the desired segments.

The landscape of segmentation approaches is vast, ranging from rudimentary **thresholding** techniques to sophisticated **(deep) neural network algorithms**. Other methodologies include **histogram-based** techniques, **clustering** algorithms, **edge detection**, **watershed transforms**, **region growing**, **level sets**, and **statistical methods**, among others.

In this course, a selection of these approaches will be briefly explored.

7.1 Thresholding techniques

Thresholding is particularly effective for images with *bimodal histograms*, such as those featuring light objects against a dark background. In this technique, pixels in an image are designated as object pixels if their intensity surpasses a certain **threshold value** t.

Conventionally, object pixels are assigned a value of 1, while background pixels receive a value of 0. An image consisting solely of 0 or 1 values is termed a **binary image**.

Another prevalent approach is to set the background pixels to 0 while retaining the original intensity values for the object pixels, depending on subsequent analyses.

$$B(x,y) = \begin{cases} 1, & if\ I(x,y) > t \\ 0, & if\ I(x,y) \leq t \end{cases}$$

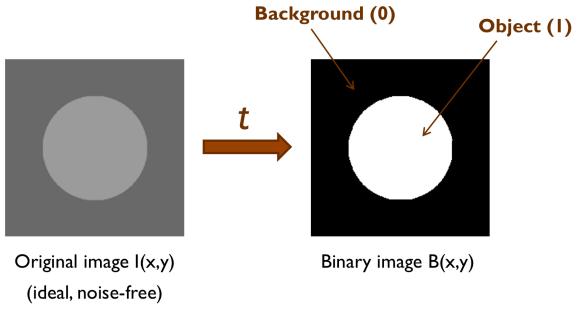


Figure 54: Simple thresholding

The question then arises: how does one select an appropriate threshold? Many image processing applications offer a visualization mode that displays pixel intensities. One can visually select a threshold by pinpointing the minimum intensity that clearly differentiates the object from the background.

However, visual selection is not always feasible or straightforward. In subsequent sections, several classic algorithms for threshold selection will be explored.

7.1.1 Iterative thresholding

Iterative thresholding is a straightforward yet effective algorithm that generally performs well without requiring extensive tuning. It is also relatively robust to noise.

Algorithm:

- Choose an initial threshold t, either randomly or based on a specific method.
- Segment the image I into two sets:
  - $G_1 = \{I > t\}$
  - $G_2 = \{I \leq t\}$
- Calculate the average of each set:
  - $m_1 = \texttt{mean}\,(G_1)$
  - $m_2 = \texttt{mean}\,(G_2)$
- Update the threshold t as the average of m_1 and m_2:
  - $t = (m_1 + m_2)\,/2$
- Repeat steps 2 - 4 until the threshold t stabilizes, indicating *convergence*.

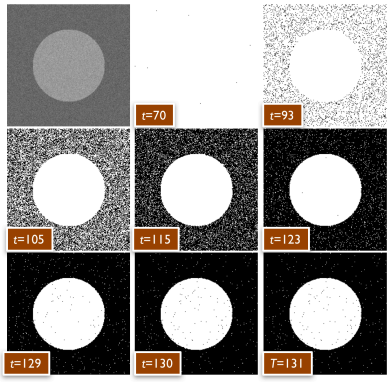


Figure 55: Iterative thresholding

In this example, the algorithm is applied to a somewhat noisy image containing a single object. The initial threshold is arbitrarily set at 70. It's worth noting that, with this initial threshold, the set $G2$ (comprising intensities below the threshold) is nearly empty. As shown in the sequence of images, the threshold stabilizes around a final value of 131 after a few iterations. Since the segmentation relies solely on intensity to distinguish between foreground and background, some darker pixels within the circle are classified as background.

7.1.2 Histogram-based segmentation

Histogram-based segmentation employs the distribution of pixel intensities in the image for segmentation purposes. Specifically, the shape of the histogram is used to identify clusters within the image.

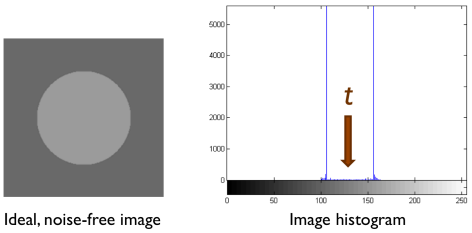


Figure 56: Noise-free image

Noise-Free Image with Two Peaks: When the histogram of the image has two distinct peaks (modes), selecting a threshold becomes straightforward. The threshold can be set at the minimum point between the two peaks.

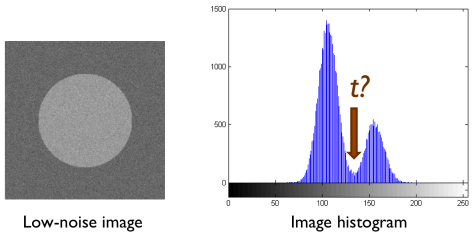


Figure 57: Low-noise image

Moderate Noise with Two Clear Peaks: Even in the presence of moderate noise, if the histogram still shows two clear peaks, determining the threshold remains relatively simple. One can still identify a minimum point between the two peaks to set as the threshold.

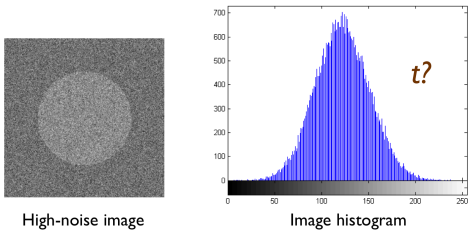


Figure 58: High-noise image

High Noise Levels: In cases where the image is highly noisy, determining the appropriate threshold becomes challenging. The histogram may not have clear peaks, making it difficult to identify a suitable threshold value.

In summary, the effectiveness of histogram-based segmentation depends on the clarity of the peaks in the histogram, which can be influenced by the level of noise in the image.

7.1.3 Otsu's method

Otsu's method is a widely-used technique for histogram-based image segmentation. The algorithm aims to find an optimal threshold that minimizes the intra-class variance, defined as:

$$\sigma_\omega^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Here, $\omega_0(t)$ and $\omega_1(t)$ are the probabilities of the two classes separated by the threshold t, and $\sigma_0^2(t)$ and $\sigma_1^2(t)$ are the variances of these classes.

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i), \omega_1(t) = \sum_{i=t}^{L} p(i)$$

Algorithm steps:

1. Compute the histogram and probabilities p_i of each intensity level 0, ..., L.
2. For each possible threshold t:
    1. calculate $\omega_0(t) = \sum_{i=0}^{t-1} p(i)$ and $\omega_1(t) = \sum_{i=t}^{L} p(i)$
    2. partition the image into two sets $G_0 = \{I \leq t\}$ and $G_1 = \{I > t\}$
    3. calculate $\sigma_0^2(t) = \texttt{var}(G_0)$ and $\sigma_1^2(t) = \texttt{var}(G_1)$
    4. calculate and store $\sigma_\omega^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$
3. Choose t for which $\sigma_\omega^2(t)$ is minimal.

In practice, variations of this algorithm are often used to maximize inter-class variance, but the underlying concept remains the same.

The following figure summarizes the results of segmenting the noise-free, low-noise and high-noise images discussed earlier using Otsu.

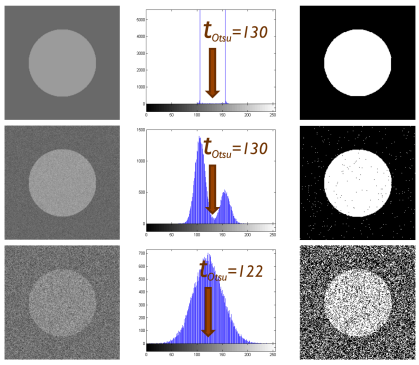


Figure 59: Otsu's method with increasingly noisy images

When the histogram shows two clear peaks, Otsu's method robustly finds the threshold, even in the presence of moderate noise. However, in cases of high noise levels, the threshold may be over or under-estimated. One approach to improve segmentation in noisy images is to pre-filter the image, for example, using a Gaussian kernel.

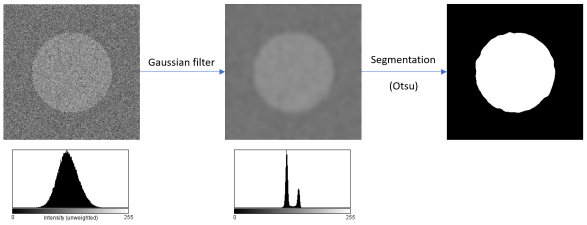


Figure 60: Filtering helps with segmentation of noisy images

This filtering process averages out the noise and narrows the spread of the histogram around the two modes of the underlying intensity distribution.

7.2 k-means clustering

The ***k*-means clustering algorithm** is a popular method for image segmentation. The algorithm aims to partition a set of n observations into k clusters, where each observation is assigned to the cluster with the

nearest mean value.

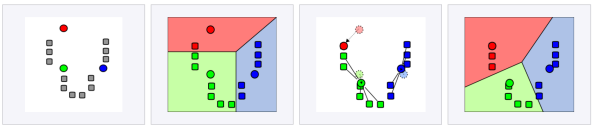


Figure 61: k-means algorithm

Algorithm steps:

1. Define the number of clusters k to be used based on the expected number of classes in the final segmentation.
2. Generate k initial "means" randomly within the data domain (shown in color).
3. Create k clusters by associating each observation (*i.e.*, pixel intensity) with the nearest (cluster) mean.
4. Update the means by calculating the new centroid of each of the k clusters.
5. Repeat steps 3 and 4 until convergence is reached, meaning the means no longer change significantly.

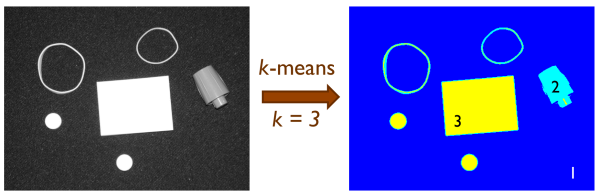


Figure 62: k-means segmentation with three classes

7.3 Adaptive thresholding

The concept of **adaptive thresholding** addresses the limitations of global thresholding methods, especially in cases where the image has varying lighting conditions or textures. By using a local threshold for different regions of the image, adaptive thresholding can handle such complexities more effectively.

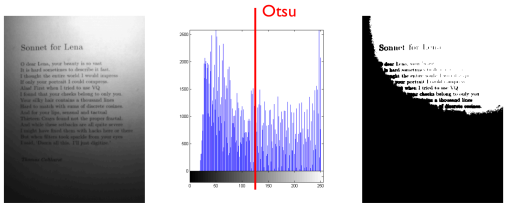


Figure 63: Limitations of global thresholding

Illumination on this sheet of paper is obviously not uniform, with much more light hitting the page on the top than the bottom. The Otsu algorithm will not succeed in extracting the text, since the division between the dark pixels belonging to the text and the light pixels belonging to the paper are blended into the intensity variation across the image due to the bad illumination. The Otsu threshold (in red) will give the meaningless (for our purposes) segmentation on the right panel.

To address such challenges, **adaptive** (also **local** or **dynamic**) thresholding is employed. In adaptive thresholding:

1. A window size and shape (*e.g.*, square, circular) are defined to slide across the image.
2. For each window position, a local statistic, commonly the mean intensity, is calculated for the pixels within the window.
3. The local threshold is then computed as $\texttt{mean}C$, where `mean` is the calculated local mean value and C is a user-defined constant.

4. This local threshold is applied to all pixels within the current window, segmenting them into object or background based on their intensity relative to the local threshold.
5. The window is slid to the next position, and steps 2-4 are repeated until the entire image is processed.

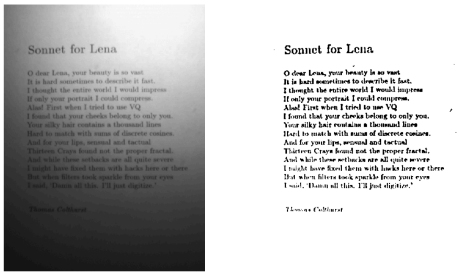


Figure 64: Adaptive thresholding of Sonnet for Lena

By using a local threshold for different regions, adaptive thresholding can effectively handle varying lighting conditions and textures within the image. Segmentation of the Sonnet for Lena using adaptive thresholding delivers the result on the right in the figure above.

7.4 Background subtraction

Background subtraction serves as an alternative to adaptive thresholding, offering a two-step approach: first correct for the background, and then apply a global thresholding technique.

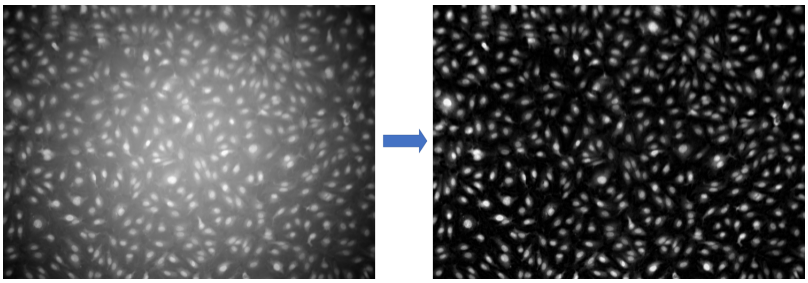


Figure 65: Background subtraction

Since we are familiar with the concept of thresholding, in this section we can focus on the task of determining the background to be subtracted. The following is a list of possible approaches.

Direct Background Image: one of the most straightforward methods, at least computationally, is to acquire a second image that contains only the background pixel intensities. This is particularly useful in cases like microscopy images affected by inhomogeneous illumination. By capturing an image of just the light profile, we can obtain a usable background image. Subtracting this from the original image effectively removes the illumination inhomogeneity.

Rolling Ball Algorithm: if obtaining a separate background image is not feasible, estimating the background from the existing image is the next best option. The Rolling Ball Algorithm serves this purpose by visualizing the image intensities as a 3D landscape. A ball is submerged under this landscape at each pixel position, and once the ball is completely covered, its apex determines the local background intensity. Rolling the ball across the entire image allows for a complete background estimation, which is then subtracted from the original image.

Morphological Opening: another estimation approach involves the use of morphological opening on the original image. A structuring element, slightly larger than the discrete objects in the image (*e.g.*, individual cells), is used. This effectively removes these objects, leaving behind the background, which is then subtracted from the original image.

Large Gaussian Kernel: a large Gaussian kernel can also be used to low-pass filter the original image. This kernel suppresses all objects smaller than its support, effectively leaving only the global intensity trend or background. This filtered version is then subtracted from the original image.

7.5 Edge detection

Edge detection algorithms focus on identifying the *boundaries* of objects in an image rather than the objects themselves. The goal is to find points where there is a sharp change in image brightness, or more formally, where there are *discontinuities*. These discontinuities can arise from variations in depth, surface orientation, material properties, or scene illumination.

The ideal outcome of edge detection would be a set of connected curves that accurately outline object boundaries. However, real-world applications often result in fragmented or false edges due to image noise. To address this, edge detection usually relies on one of two criteria:

- Locations where the **first derivative** of the intensity is larger in magnitude than a certain threshold.
- Locations where the **second derivative** of the intensity has a zero crossing.

To improve the stability of edge detection, images are often pre-smoothed to reduce the impact of noise.

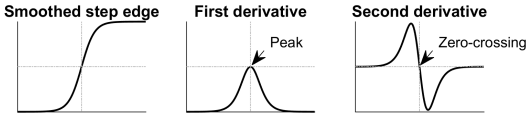


Figure 66: 1-dimensional derivatives

Classical gradient detectors are the **Sobel**, **Prewitt** and **Robert** operators. These are gradient-based edge detectors that use convolution with a pair of 3×3 kernels which approximate the gradient of the image intensity function. The **Canny edge detector** is a multi-step algorithm that involves smoothing the image with a Gaussian filter and then finding the intensity gradients. It also includes post-processing steps to suppress noise and improve edge localization.

Figure 67: canny edge detector

7.6 Blob detection

Blob detection, also known as spot or particle detection, is a technique used for identifying regions in an image where there are local intensity maxima. These maxima are typically found within a radius defined by the estimated blob diameter.

Key parameters for blob detection are:

Tolerance Level: a threshold can be set to select only those local maxima that are significantly higher than the surrounding background, thereby reducing the chance of false positives.

Filtering Step: often, a pre-processing step is applied to enhance features of a particular size or scale. A common filter used for this purpose is the **Laplacian of Gaussian (LoG)** filter.

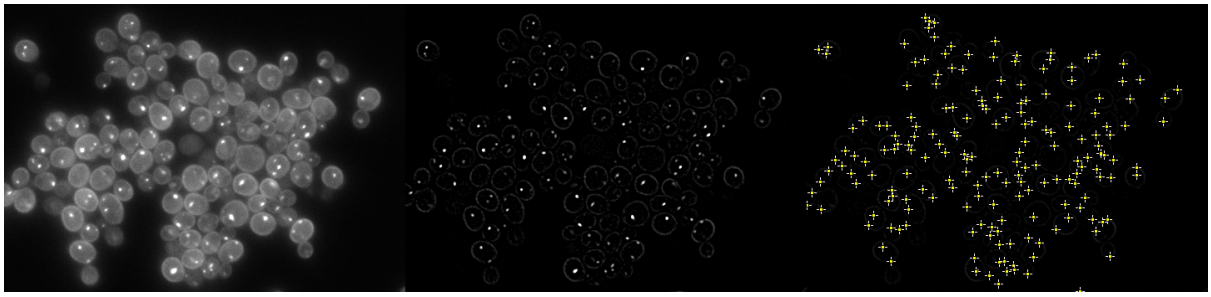


Figure 68: Extracting spots from cells. Image courtesy Fabian Rudolf, D-BSSE, ETH Zurich

In the example image above, which shows cells with weaker and stronger spots, some of the weaker spots are challenging to segment as they are only slightly brighter than the surrounding cytoplasm. By applying a Laplacian of Gaussian filter with a standard deviation of $\sigma = 1$, the cell signal is suppressed while the spot signal is enhanced, making the detection process more robust.

7.7 Connected components

After segmentation, the resulting binary image usually consists of pixels that are either part of the background (with a value of 0) or the foreground (with a value of 1). While thresholding is a crucial step, it's often just an intermediate phase in the overall process of extracting meaningful, quantitative information from images.

In the image below, all the white pixels collectively form what we refer to as the *foreground*. While this is valuable information, it often doesn't go far enough. For example, if the foreground consists of multiple cells, we would ideally like to identify each cell as a separate entity.

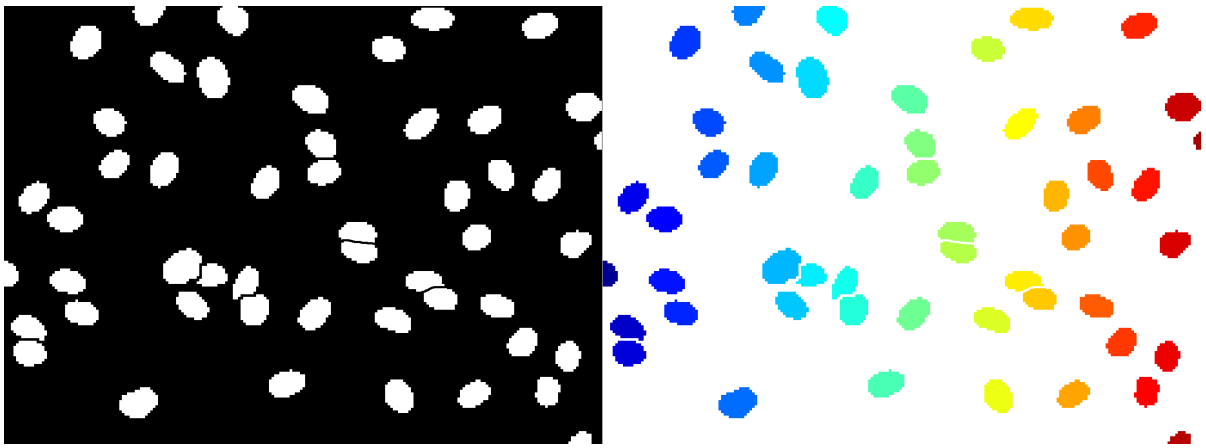


Figure 69: Connected components

In the right panel of the figure, each blob (or cell) has been assigned a unique identity number, represented here by unique colors. This is achieved through a **connected components** algorithm.

The connected components algorithm works by grouping all sets of foreground pixels that are adjacent to each other based on a defined **neighborhood connectivity** criterion. In 2D images, this is often 4- or 8-connectivity, while in 3D images, it could be 6-, 18-, or 26-connectivity. Each of these connected groups forms a separate subset, representing an individual object that is spatially distinct from the others.

Once these subsets are identified, various **features** such as size, shape, and intensity can be extracted for each individual object for further analysis.

8 Morphological operations

Binary (mathematical) image morphology is a powerful class of operations that act on the **shape** of objects in an image.

The foundation of mathematical morphology lies in **set theory**. Sets in mathematical morphology represent **objects** in an image. In **binary** images, sets are member of the two-dimensional integer space Z^2, where each element is as 2-element vector whose coordinates are the (x, y) coordinates of a white (foreground) pixel in the image. Gray-scale images can be represented as sets in Z^3, and the pixels are represented by their coordinates and intensity value (x, y, i). In this section we will concentrate on **binary morphology**.

Mathematical morphology is used in image enhancement, segmentation, restoration, feature detection, shape analysis, and many others.

Image morphology makes use of **structuring elements** of given **shape** to probe an image for properties of interest.

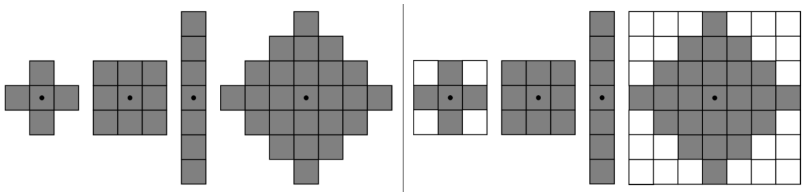


Figure 70: Structuring elements

In the left panel of the picture above, the elements of a structuring element are shaded in gray. The origin of the structuring element is indicated by a dot. When working with images, the structuring element must be a rectangular array and we therefore append enough background elements (in white, in the right panel above) to form a rectangular array. For instance, the padded version of the first structuring element above is represented as follows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The target of a **morphological binary filter** is a **binary image**, where all pixels are either set to 0 if they represent the image *background* or to 1 if they represent some *objects* (or *foreground*).

Morphological filters are applied to a binary image A by moving the padded structuring element B over all pixels in the image (the set of pixels z) and applying a logical operation between B and the pixel neighborhood (also called the *windowed set*) centered at each pixel.

8.1 Erosion and dilation

The operations of **erosion** and **dilation** are fundamental to morphological image processing and are the building blocks of many, more complex algorithms.

8.1.1 Erosion

The **binary erosion** is defined by:

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

The erosion of A by the structuring element B is the set of all displacements z such that B, translated by z, is completely contained in A.

Figure 71: Erosion

The erode filter shrinks the size of the foreground or object in the binary image A. All objects smaller than the structuring element are removed (*i.e.*, set to background).

8.1.2 Dilation

The **binary dilation** is defined by:

$$A \oplus B = \left\{ z \mid (\hat{B})_z \cap A \neq \emptyset \right\}$$

The dilation of A by the structuring element B is the set of all displacements z such that B, translated by z and reflected, so that $B(x, y) = \hat{B}(-x, -y)$, overlaps by at least one element.

Figure 72: Dilation

The dilate filter grows (or thickens) objects in the binary image. The specific manner and extent of this thickening is controlled by the shape of the structuring element used.

8.1.3 Opening and closing

The **opening** of an image A by a structuring element B is a the erosion of A followed by a dilation of the result.

$$A \circ B = (A \ominus B) \oplus B$$

Figure 73: Opening

Opening generally smoothes the contours of objects, eliminates protrusions and breaks narrow linkages.

The **closing** of an image A by a structuring element B is a the dilation of A followed by an erosion of the result.

$$A \bullet B = (A \oplus B) \ominus B$$

Figure 74: Closing

Closing also tends to smooth objects, but in contrast to erosion, it fuses narrow breaks and long, thin gulfs, eliminates small holes, and fill gaps in contours.

8.1.4 Duality of erosion and dilation

Erosion and dilation are **duals** of each other, that is:

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

and

$$(A \oplus B)\, c = A^c \ominus \hat{B}$$

In particular, when the structuring element B is symmetric about its origin (*i.e.*, when $B = \hat{B}$), we can obtain the erosion of an image by simply dilating its background (*i.e.*, A^c) and taking the complement.

8.1.5 Boundary extraction

The boundary set of A, denoted by $\beta(A)$, can be obtained by first eroding A by B and then performing the set difference between A and its erosion.

$$\beta(A) = A - A \ominus B$$

Figure 75: Boundary extraction

8.1.6 Hole filling

A *hole* is a background region surrounded by a connected border of connected pixels. The algorithm for image filling is not trivial, and we will not discuss it in detail here.

Figure 76: Hole filling

8.2 Watershed

Often, objects in an image that are close to each other are difficult to segment, and are fused into individual blobs in the binary image.

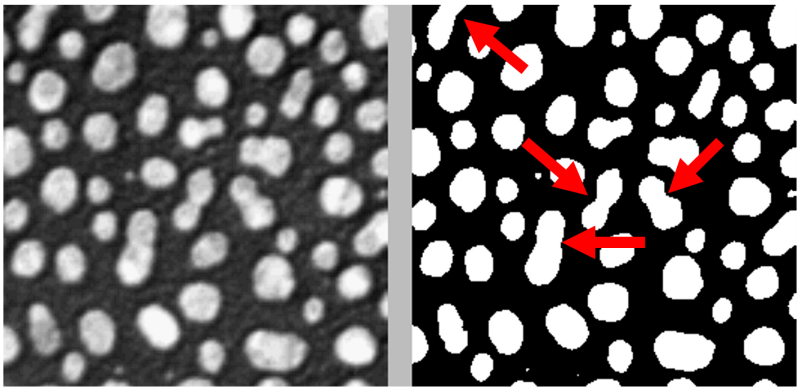


Figure 77: Watershed motivation

As is often the case in image analysis, there are many alternative or complementary methods for partitioning space or segmenting images. While the **Watershed Transform** is a widely used technique for segmentation, other approaches like the **Voronoi Transform** and **Morphological Reconstruction** can be applied effectively.

8.2.1 Watershed transform

The **watershed transform** turns a segmented, black-and-white image into a 3D landscape (using an algorithm that we will discuss presently), with x and y coordinates representing pixel locations and a third dimension for local elevation. If we imagine our landscape on a rainy day, we can define three key sets of points:

1. points at a regional minimum (valleys);
2. points where a droplet of water would slide down towards one and only one minimum (slopes);
3. points where a droplet of water would be equally likely to fall to more than one such minimum (top of hills).

For a particular regional minimum, the set of points satisfying condition **2** is called the **catchment basin** or **watershed** of that minimum. The points satisfying condition **3** form crest lines on the topographic surface and are called **divide lines** or **watershed lines**.

The principal idea of the **watershed transform** is to find the watershed lines. Algorithms for finding the watershed lines are based on a similar metaphor:

- Holes are punched at each of the local minima in the topographic surface.
- The entire topography is flooded from below by letting water raise through the holes at constant rate.
- As the rising water from distinct catchment basins is about to merge at the top of a hill, a dam is built.
- The whole landscape is flooded until only the dams are visible: they represent our **watershed lines**.

To generate a useful topographical surface from the black-and-white mask, we often use the **distance transform** D of the binary segmentation image. The value at each position (x, y) is the distance of that pixels from the **closest background pixel**, or 0 if the pixel is a background pixel itself.

Different distance metrics can be used: the **E**uclidean **D**istance **T**ransform (**EDT**) calculates the exact distance between any two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ in the image as:

$$D_{p_1,p_2} = (\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2})$$

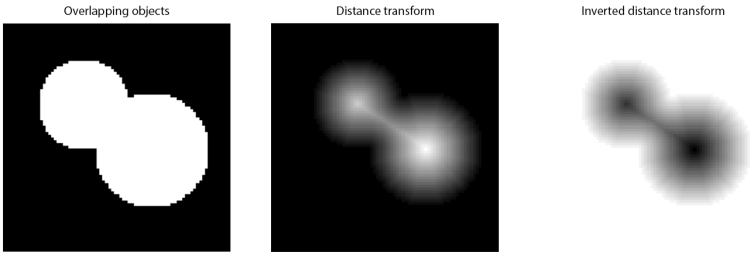


Figure 78: Distance transform

Unfortunately, the distance transform $D(x, y)$ lacks usable local minima for flooding. The solution is to invert the distance transform, converting local maxima into local minima (red dots on the left of the figure below), and enabling the watershed algorithm to separate touching objects effectively.

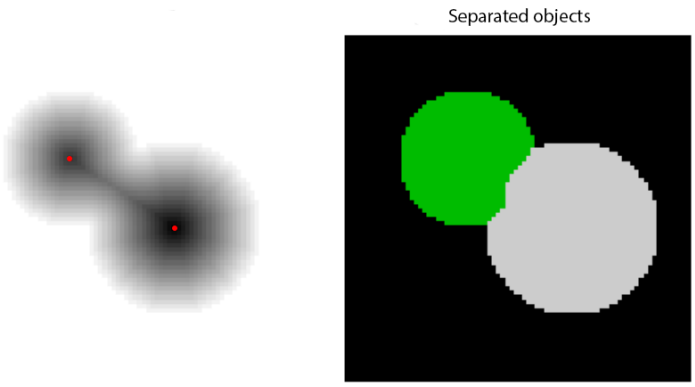


Figure 79: Separated objects

8.2.2 Voronoi transform

In mathematics, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells.

If the distance transform is applied to an inverted binary image, the pixel values give the distance to the closest foreground object. The Voronoi transform divides the image into regions whose separation lines have the same distance to the nearest foreground regions.

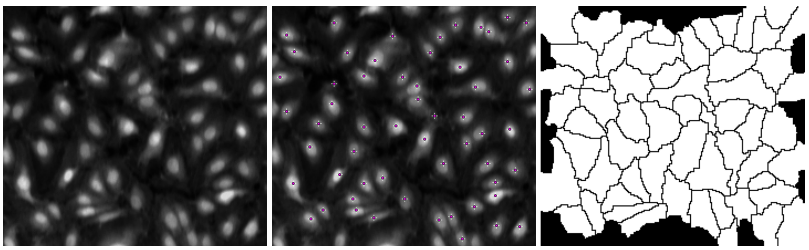


Figure 80: Voronoi transform

8.2.3 Morphological reconstruction

So far we have discussed morphological operations requiring one image A and one structuring element B. The **morphological reconstruction** transformation involves two (binary) images and one structuring element. One image is the **marker** F and contains the starting points or seeds of the transformation. The other image, the **mask** G constraints the transformation. The structuring element B defines the connectivity (4- or 8-connected). The important requirement is that $F \subseteq G$.

Morphological reconstruction will iteratively dilate the marker F under the constraint that the dilation must be contained in the mask G. The end result is that all objects in G that contain a seed in F will be preserved unchanged, whereas the ones without seed will disappear.

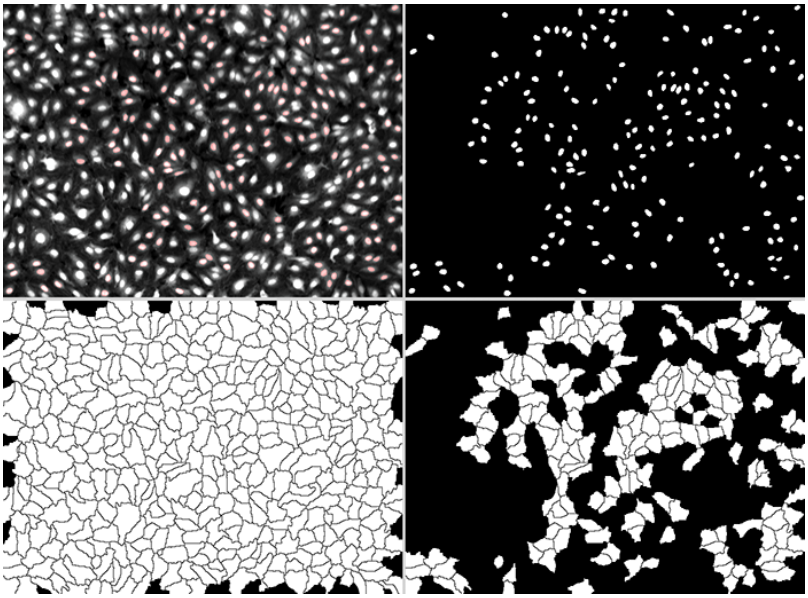


Figure 81: Morphological reconstruction

8.3 Gray-scale morphology

Morphological operations can be extended to gray-scale images and work fundamentally in the same way as their binary counterparts. We define an image $f(x, y)$ and a structuring element $b(x, y)$. The structuring element can either be *flat* (*i.e.*, all its values are the same), or non-flat (*i.e.*, with non-homogeneous values).

The **gray-scale erosion** for a flat structuring element is defined as:

$$[f \ominus b](x, y) = \min_{(s,t)\in b} f(x + s, y + t)$$

The **gray-scale dilation** for a flat structuring element is defined as:

$$[f \oplus b](x, y) = \max_{(s,t)\in b} f(x - s, y - t)$$

In analogy to the binary dilation case, the structuring element is mirrored.

The **gray-scale erosion** for a non-flat structuring element is defined as:

$$[f \ominus b_N](x, y) = \min_{(s,t)\in b_N} f(x + s, y + t) - b_N(s, t)$$

The values of the structuring element are subtracted from the values of the image before the minimum value is calculated and returned.

Finally, the **gray-scale dilation** for a non-flat structuring element defined as:

$$[f \oplus b_N](x, y) = \max_{(s,t)\in b_N} f(x - s, y - t) + b_N(s, t)$$

Here, the values of the mirrored structuring element are added from the values of the image before the maximum value is calculated and returned.

Gray-scale opening and **closing** are defined exactly as their binary counterparts.