

Computer Vision

Matteo Gialazzo

March 14, 2025

Contents

1	Introduction	2
2	Fundamentals of Image Processing and Computer Vision	2
2.1	Images	2
2.1.1	Pinhole camera model	2
2.1.2	Stereo images	3
2.1.3	Stereo correspondence	4
2.1.4	Epipolar geometry	4
2.1.5	Depth of Field (DOF)	4
2.1.6	Lenses	5
2.1.7	Diaphragm	6
2.1.8	Focusing mechanism (manually changing depth of field)	6
2.1.9	Image digitization	7
2.1.10	Camera sensors	7
2.1.11	SNR	7
2.1.12	Dynamic Range (DR)	7
2.2	Image Filtering	8
2.2.1	Noise and image filters	8
2.2.2	Convolution	8
2.2.3	Discrete convolution	9
2.2.4	Practical implementation	9
2.2.5	Mean filter	9
2.2.6	Gaussian filter	10
2.2.7	Median filter	10
2.2.8	Bilateral filter	11
2.2.9	Non-local means filter	11
2.3	Edge detection	13
2.3.1	Non-Maxima Suppression (NMS)	13
2.3.2	Canny's edge detector	13
2.3.3	Second derivative along the gradient & Laplacian	14
2.3.4	Laplacian of Gaussian (LoG)	14
2.4	Feature detection	14
2.4.1	Local invariant features paradigm	15
2.4.2	Properties of good detectors/descriptors	15
2.4.3	Moravec Interest Point Detector	15
2.4.4	Harris Corner Detector	16
2.4.5	Scale-Space	16
2.4.6	Scale and Rotation Invariance Description	19
2.4.7	SIFT Descriptor	19
3	Advanced Topics in Deep Learning for Computer Vision	20

1 Introduction

The difference between computer vision and image processing is the fact that computer vision is the process of extracting information from images, while image processing aims at improving the quality of images. Quite often image processing helps computer vision. The informations we want to extract from the images could be counting, object orientation, object classification, measurements... Computer vision is challenging since we lose depth from the images (3D information becomes 2D), scale and illumination varies, and there's object occlusion (object hiding other objects).

Computer vision started with hand-crafted decision rules that only required few images as example, and evolved to machine learning where the algorithm learns a decision rule, but the training requires hundreds to thousands of images. The big paradigm shift happened with deep learning (e2e learning) which learned both the image representation and the decision rules, but required thousands to millions of words. Deep learning has been enabled by better networks, better hardware and more data.

2 Fundamentals of Image Processing and Computer Vision

2.1 Images

An imaging device gathers the light reflected by 3D objects to create a 2D representation of the scene.

2.1.1 Pinhole camera model

The "pinhole camera" is the simplest camera model we can define. Light goes through the very small pinhole (to not have saturation) and hits the image plane. Geometrically, the image is achieved by drawing straight rays from scene points through the hole up to the image plane.

This simple geometrical model turns out to be a good approximation of the geometry of image formation. However, useful images can hardly be captured by means of a pinhole camera.

The geometric model of image formation in a pinhole camera is known as **perspective projection**.

- M : scene point
- m : corresponding image point
- I : image plane
- C : optical centre (pin hole)
- Optical axis: line through C and orthogonal to I
- c : intersection between optical axis and image plane (image centre or piercing point)
- f : focal length
- F : focal plane

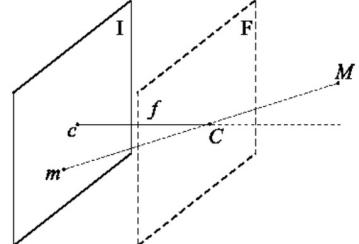


Figure 1: perspective projection

Which, by writing the points as vectors and the plane's coordinates becomes:

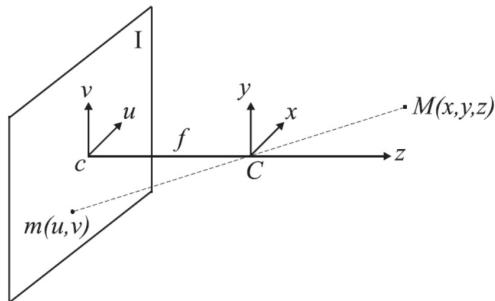


Figure 2: geometric model of image formation

Given the reference frame in 2

- u is the horizontal axis in the image plane.

- v is the vertical axis in the image plane.
- X and Y are the respective axis in the 3D reference system. It's called the **camera reference system** because it is "attached" to the camera.

For the perspective model these axis must be parallel

The equations to map scene points into their corresponding image points are defined as:

$$\frac{u}{x} = -\frac{f}{z} \rightarrow u = -x \frac{f}{z} \quad \frac{v}{y} = -\frac{f}{z} \rightarrow v = -y \frac{f}{z}$$

The minus sign means the axis gets inverted (as we can see in the visualization, and it's what happens in the brain). We can get rid of the sign, since the image plane can be thought of as lying in front rather than behind the optical centre.

Image coordinates are a scaled version of scene coordinates (function of depth), this means that if the distance becomes bigger the object gets smaller. When z increases, since it's at the denominator in both the equations the terms get smaller (object gets smaller). **The larger the focal length the bigger the object is in the image (and viceversa).**

As we previously said, the image formation process deals with mapping a 3D space onto a 2D space, and so to a loss of information. A given scene point is mapped into an image point, but an image point is mapped onto a 3D line. For an image point we can only state that its corresponding scene point lies on a line but cannot disambiguate a specific 3D point along such a line.

2.1.2 Stereo images

The solution is to use multiple images (at least two) to create stereo vision. The human visual system is a stereo vision system. Given correspondences, 3D information can be recovered easily by triangulation. We can use two cameras, or two cameras and an infrared sensor to project guides to align the images.

For standard stereo geometry there are some assumptions we have to make:

- The cameras have parallel (x, y, z) axes.
- The image planes of both cameras are coplanar and aligned.
- Both cameras have identical focal lengths.

Based on this, the transformation between the two reference frames is just a translation, usually horizontal. For stereo vision is also really important to **sense two images at the very same moment**.

The cameras are displaced at a given quantity b called baseline. The **disparity** is the difference between the horizontal coordinates in the left and right images.

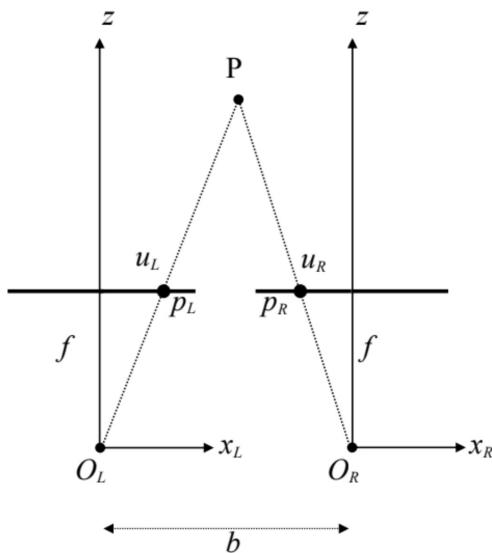


Figure 3: standard stereo geometry

The fundamental relationship in stereo vision is $z = b \cdot \frac{f}{d}$. It's used to calculate the depth of a point in a scene from a pair of stereo images.

In standard stereo geometry since we are given just two 2D images there is no info about the correspondence between two points in the two images. We can recall that the cameras have parallel

axes, and so we know that we can search for the correspondence along the horizontal lines. This task is called stereo matching.

2.1.3 Stereo correspondence

Given a point in one image find it in the other image which is the projection of the same 3D point. Such image points are called corresponding points. **Points farther away have a smaller disparity, while close points have a larger display**



Figure 4: Corresponding points look similar in the two images

The image of a 3D line segment of length L lying in a plane parallel to the image plane at distance z from the optical centre will exhibit a length given by:

$$l = L \frac{f}{z}$$

This relationship is more complicated for an arbitrarily oriented 3D segment, as its position and orientation need to be accounted for as well. For a **given position and orientation, length always shrinks alongside distance**.

Perspective projection maps 3D lines into image lines. **Parallelism between 3D lines is not preserved** (except for lines parallel to the image plane). This is the reason why if we look at a really long road into the distance we have the perception that the road becomes thinner, and the lines of the road intersect in the distance. The images of parallel 3D lines intersect at a point, called **vanishing point**, which isn't necessarily within the image.

If the lines are parallel to the image plane they meet at infinity.

2.1.4 Epipolar geometry

What if the two cameras are no longer aligned? Do we need to search through the whole image? We can project the line related to point P_L in the right plane and search across that line. The issue is that this projection can be computed only if the transformation between the two cameras is known (the relative mapping between the two cameras).

It is almost impossible to build a stereo rig which is perfectly aligned horizontally. Searching through oblique epipolar lines is awkward, and computationally is less efficient. What people do in practice is to convert epipolar geometry to standard geometry with rectification/warping. We warp the images as if they were acquired through a standard geometry, then we can compute and apply to both images a transformation known as rectification.

2.1.5 Depth of Field (DOF)

A scene point is on focus when all its light rays, gathered by the camera, hit the image plane at the same point. In a pinhole device this happens to all scene points because of the very small size of the hole, so that the camera features an infinite Depth of Field (DOF).

The drawback is that such a small aperture allows gathering a very limited amount of light. The image is really sharp, but has a really low light. If a point is projected onto a circle instead

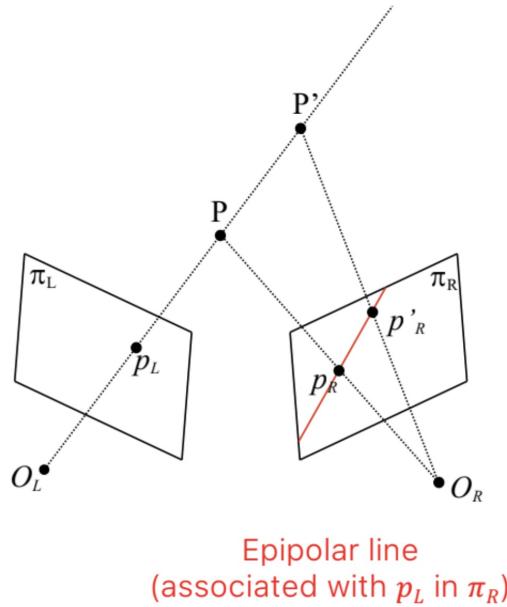


Figure 5: Epipolar line

of a point (bigger pinhole) the image is not sharp (not on focus). If we cannot gather through aperture we have to integrate through time, by using a longer exposure time.

2.1.6 Lenses

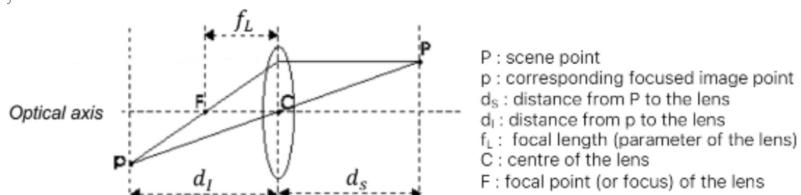
Lenses concentrate light, so we use them to gather more light from a scene point and focus it on a single image point. This enables much smaller exposure times. This way Depth Of Field is no longer infinite, and only points across a limited range of distances can be simultaneously on focus in a given image.

We will consider the approximate model known as thin lens equation, which is useful to graphically determine the position of a focused image point:

- Rays parallel to the optical axis are deflected to pass through F .
- Rays through C are undeflected.

If the image is on focus, the image formation process obeys to the perspective projection model:

- The center of the lens is the optical center.
- The distance v acts as the effective focal length of the projection.



$$\frac{1}{d_s} + \frac{1}{d_l} = \frac{1}{f_l}$$

Figure 6: Scheme of a lens

Choosing the distance of the image plane determines the distance at which scene points appear on focus in the image. Scene points in front and behind the focusing plane will result out-of-focus, thereby appearing in the image as circles, known as circles of confusion, or **blur circles**, rather than points.

The advantage of lenses is to have a small exposure time for capturing moving objects but we pay in terms of Depth Of Field.

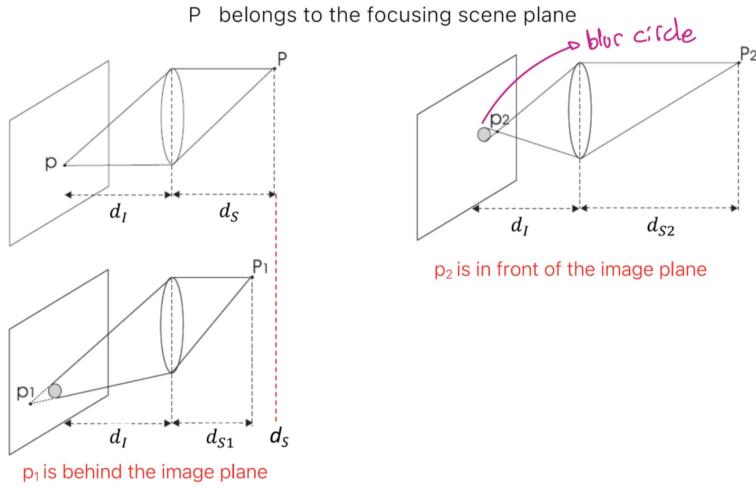


Figure 7: Lens focusing at various distances

2.1.7 Diaphragm

In theory, when imaging a scene through a thin lens, only the points at a certain distance can be on focus, all the others appear blurred into circles. However, as long as the circles are smaller than the size of the photosensing elements (a single pixel), the image will still look on-focus.

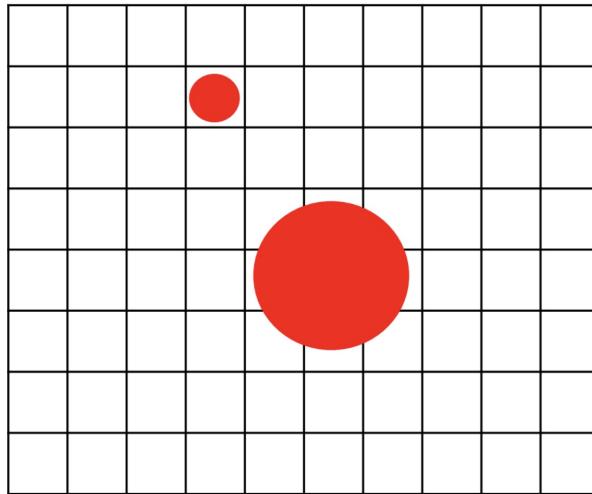


Figure 8: Blurring at pixel level

The range of distances across which the image appears on focus, due to blur circles being small enough, determines the Depth Of Field) of the imaging apparatus. Cameras often deploy an adjustable diaphragm (iris) to control the amount of light gathered through the effective aperture of the lens.

- Reduce aperture \rightarrow less light \rightarrow smaller blur circle
- More aperture \rightarrow more light \rightarrow larger blur circle.
- Close the diaphragm \rightarrow increase depth of field \rightarrow not enough light \rightarrow increase exposure time \rightarrow moving object \rightarrow motion blur \rightarrow still scenes.

2.1.8 Focusing mechanism (manually changing depth of field)

To focus on objects at diverse distances we need a mechanism that allows the lens (or the lens subsystem) to translate along the optical axis with respect to the fixed position of the image plane.

At one end position ($d_I = f_L$) the camera is focused at infinity (objects at infinity are on sharp focus). The focusing mechanism allows the lens to be translated farther away from the image plane up to a certain maximum value, which determines the minimum focusing distance.

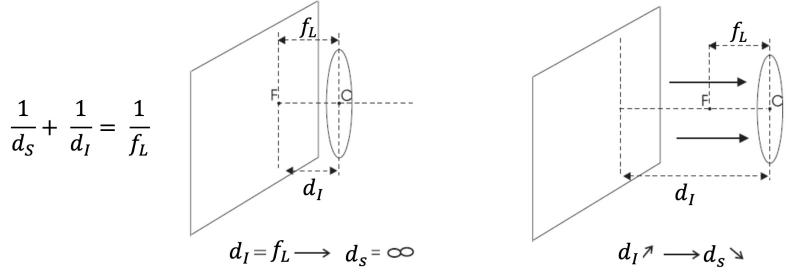


Figure 9: Focusing mechanism

2.1.9 Image digitization

How do we convert a continuous image to a discrete one which can be represented on a computer? The process can be divided in two steps: sampling and quantization.

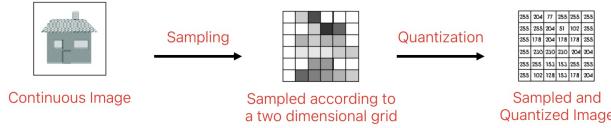


Figure 10: The two steps of the image digitization process

- **Sampling:** the planar continuous image is sampled along both the horizontal and vertical directions to pick up a matrix of $n \times m$ samples known as pixels.
- **Quantization:** the continuous range of values associated with pixels is quantized into $l = 2^m$ discrete levels known as gray-levels. m is the number of bits used to represent a pixel, with the memory occupancy (in bits) of a gray-scale image given by $B = N \times M \times m$. Coloured digital images are typically represented within computers using 3 bytes per pixels. Both more pixels and more bits per pixel result in a higher quality image.

The more bits we spend for its representation, the higher the quality of the digital image (becomes a closer approximation to the ideal continuous image). This applies both to sampling and quantization.

2.1.10 Camera sensors

The sensor is a matrix of photodetectors. During exposure time, each detector converts the incident light into a proportional electric charge. The companion circuitry reads-out the charge to generate the output signal, which can be either digital or analog. For digital cameras the sensor includes the necessary ADC circuitry.

Today, the two main sensor technologies are:

- Charge Coupled Devices (CCD), where the sensor and circuit are separated.
- CMOS (Complementary Metal Oxide Semiconductor), where everything is on the same circuit.

CCD/CMOS sensors can't sense colors, so we place an array of optical filters in front of the photodetectors, to render each pixel sensitive to a specific range of wavelengths.

2.1.11 SNR

The intensity measured at a pixel under perfectly static conditions varies due to the presence of random noise. The main noise sources are:

- **Photon Shot Noise:** the number of photons collected during exposure time is not constant.
- **Electronic Circuitry Noise:** generated by the electronics.
- **Quantization Noise:** related to the ADC conversion.
- **Dark Current Noise:** random charge observed due to thermal excitement.

SNR can be expressed both in decibels and bits.

2.1.12 Dynamic Range (DR)

If the sensed amount of light is too small, the "true" signal cannot be distinguished from noise. Given E_{\min} : the minimum detectable irradiation, and E_{\max} , the saturation irradiation. The

Dynamic Range (DR) of a sensor is defined as $DR = \frac{E_{\max}}{E_{\min}}$, and like the SNR, it is often specified in decibels or bits.

Like SNR, the higher the DR the better it is. A higher DR corresponds to the ability of the sensor to simultaneously capture in one image both the dark and bright structures of the scene.

2.2 Image Filtering

2.2.1 Noise and image filters

In computer vision we have to deal with noise. Noise is always different and more noticeable in uniform regions of the image. The simplest thing to reduce noise is to output the average of the pixel color over time, to get **almost** the ideal noiseless value.

$$O(p) = \frac{1}{T} \sum_{t=1}^T I_k(p) = \frac{1}{T} \sum_{t=1}^T (\tilde{I}(p) + n_t(p))$$

This technique works well on still images.

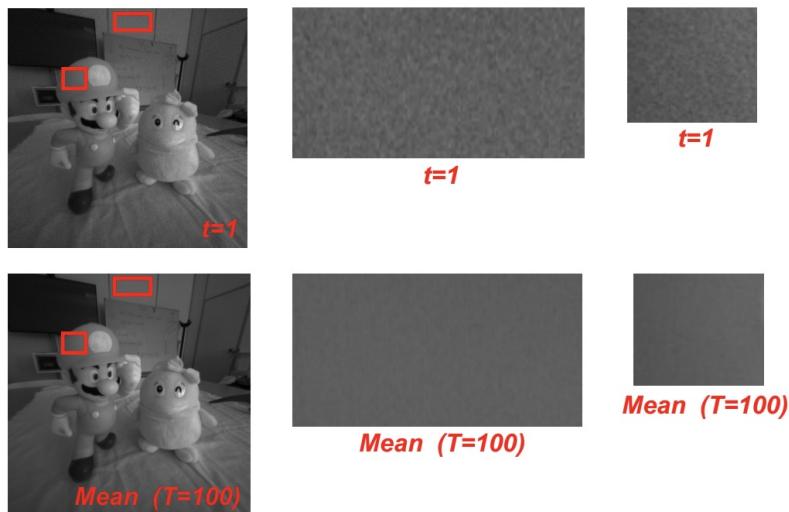


Figure 11: Simple denoising

If we are given a simple image, we may compute a mean across neighbouring pixels, like a spatial rather than temporal mean. The size of the square of the neighbouring pixels is a tradeoff.

This is a very basic denoising filter.

Image filters are image processing operators that compute the new intensity (colour) of a pixel, p , based on the intensities (colours) of those belonging to a neighbourhood (support) of p .

2.2.2 Convolution

An important sub-class of filters is given by **Linear** and **Translation-Equivariant (LTE)** operators.

The application of filters in image processing consists in a 2D convolution between the input image and the impulse response function of the LTE operator.

LTE operators are used as feature extractors in Convolutional Neural Networks (CNNs).

Given an input 2D signal $i(x, y)$, a 2D operator $Ti(x, y)$ is said to be linear if and only if

$$T\alpha i_1(x, y) + \beta i_2(x, y) = \alpha o_1(x, y) + \beta o_2(x, y)$$

with $o_1 = Ti_1$ and $o_2 = Ti_2$ and α, β are two constants. The operator is said to be **translation-equivariant** if and only if: $Ti(x - x_0, y - y_0) = o(x - x_0, y - y_0)$

If the operator is LTE, the output signal is given by the **convolution** between the input signal and the impulse response (point spread function) $h(x, y) = T\delta(x, y)$.

$$o(x, y) = i(x, y) * h(x, y) = Ti(x, y)$$

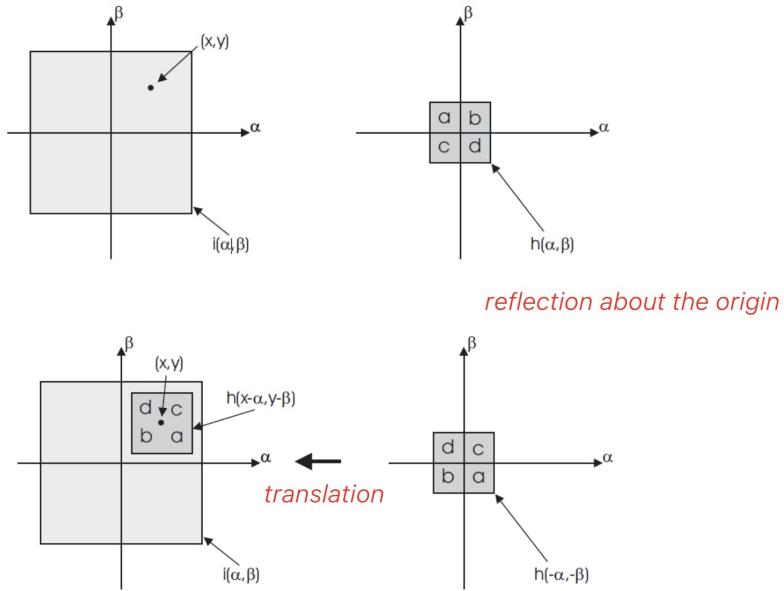


Figure 12: A graphical view of convolution

$$i(x, y) * h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} i(\alpha, \beta) h(x - \alpha, y - \beta) d\alpha d\beta$$

Convolutions have some useful properties:

- **Associative property:** $f * (g * h) = (f * g) * h$ (useful because we can decompose kernels and obtain faster operations).
- **Commutative property:** $f * g = g * f$.
- **Distributive property:** $f * (g + h) = f * g + f * h$.
- **Convolution commutes with differentiation:** $(f * g)' = f' * g = f * g'$.

The correlation of signal $i(x, y)$ with respect to signal $h(x, y)$ is defined as:

$$i(x, y) \circ h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} i(\alpha, \beta) h(x - \alpha, y - \beta) d\alpha d\beta$$

Correlation is not commutative, unlike the convolution.

2.2.3 Discrete convolution

Normal convolution is useful for signal theory, but we want to have a discrete convolution, where we use summations instead of integrals. The four convolution properties highlighted for the convolution hold for the discrete one too.

2.2.4 Practical implementation

CNNs learn flipped kernels. In image processing both the the input image and the kernel are stored into matrices of given finite sizes, with the image being much larger than the kernel. Conceptually, to obtain the output image we need to slide the kernel across the whole input image and compute the convolution at each pixel.

We have an issue on borders, since the kernel "goes out" of the matrix of the image. We have two main options to solve this issue:

- **CROP:** common in image processing.
- **PAD:** preferred in CNNs. We can do zero-padding, replicate the first pixel many times, reflect the first k pixels (half of the kernel)...

Without padding convolutions shrink the images.

2.2.5 Mean filter

Mean filtering is the simplest and fastest way to denoise an image. It consists in replacing each pixel intensity by the average intensity overa chosen neighbourhood. According to signal processing

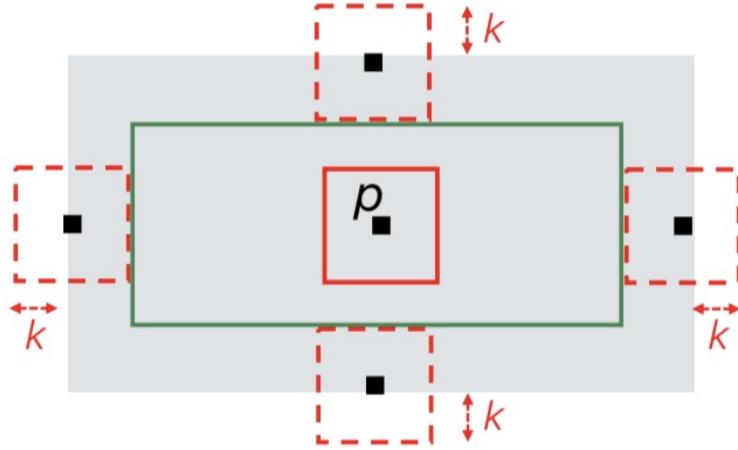


Figure 13: Issues with convolutions

theory, the Mean Filter carries out a low-pass filtering operation, which in image processing is also referred to as image smoothing. Smoothing is often aimed at image denoising, but sometimes is used to cancel out small-size unwanted details that might hinder the image analysis task. Linear filtering reduces noise but blurs the image, so we lose sharpness.

2.2.6 Gaussian filter

Let's try to use a more sophisticated but still linear filter. The gaussian filter is an LTE operator whose impulse response is a 2D Gaussian function

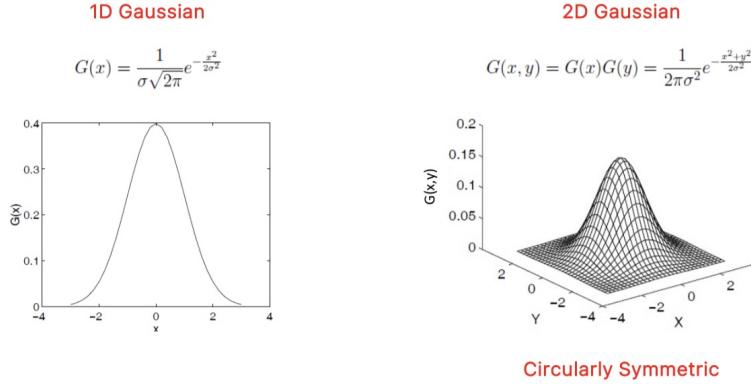


Figure 14: 1D and 2D Gaussian plot

The larger the size of the gaussian kernel, the more accurate will the discrete approximation be, but the computational cost grows with the filter size. We should then use larger sizes for filters with higher σ , smaller sizes whenever σ is smaller. A typical rule is to choose the size of the filter by capturing the interval $[-3\sigma, +3\sigma]$ since it captures 99% of the energy of the gaussian impulse. The Gaussian filter is the only one which has been mathematically proven of not altering image information but just simplification. To speedup the filtering we can apply the 2D gaussian by doing 2 1D gaussian filterings.

We also observe that the higher the σ , the higher is the smoothing caused by the filter. We can use this to remove details from the image.

2.2.7 Median filter

There is noise that Gaussian filters can't handle well, that is the salt and pepper noise. It's usually caused by image corruption (or broken pixels in the sensor). Linear filtering is ineffective and just blurs the image.

We can use a **non-linear** filter, where each pixel intensity is replaced by the median over a given neighbourhood. Median filtering counteracts impulse noise effectively, since outliers tend to

fall at either the top or the bottom end of the sorted intensities. Median filtering tends to keep sharper edges than linear filters such as the Mean or the Gaussian.



Figure 15: Example of the power of the median filter

The median filter can effectively denoise the image without introducing significant blur, yet, Gaussian-like noise, such as sensor noise, cannot be dealt with by the median, as this would require computing new noiseless intensities.

2.2.8 Bilateral filter

The bilateral filter preserves edges while filtering out noise. It's an advanced non-linear filter to accomplish denoising of Gaussian-like noise without blurring the image. It's also called edge preserving smoothing.

$$O(p) = \sum_{q \in S} H(p, q) \cdot I_q \quad H(p, q) = \frac{1}{W(p)} G_{\sigma_s}(d_s(p, q)) G_{\sigma_r}(d_r(p, q))$$

- **spatial distance:** $d_s(p, q) = \|p - q\|_2 = \sqrt{(u_p - u_q)^2 + (v_p - v_q)^2}$.
- **range (intensity) distance:** $d_r(I_p, I_q) = |I_p - I_q|$.
- **normalization factor:** $W(p) = \sum_{q \in S} G_{\sigma_s}(d_s(p, q)) G_{\sigma_r}(d_r(p, q))$

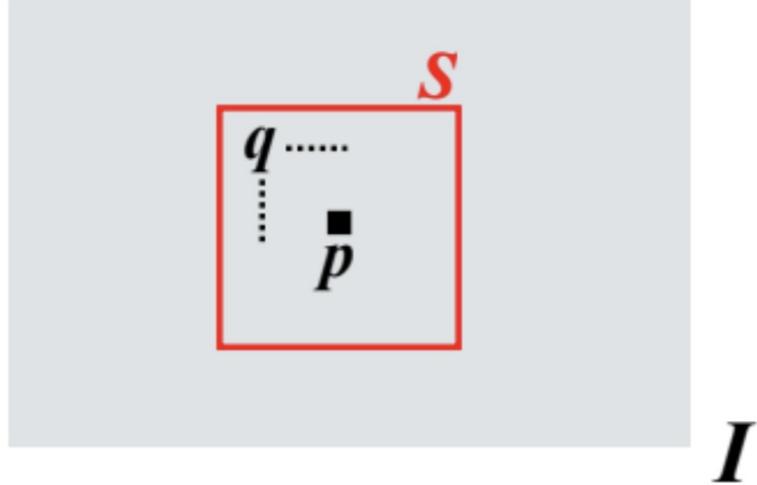


Figure 16: Application of the bilateral filter

Given the supporting neighbourhood, neighbouring pixels take a larger weight as they are both closer and more similar to the central pixel. At a pixel nearby an edge, the neighbourhood falling on the other side of the edge looks quite different and thus cannot contribute significantly to the output value due to their weights being small. **The kernel has to be recomputed for each pixel.**

2.2.9 Non-local means filter

It's another non-linear edge preserving smoothing filter. The key idea is that the **similarity among patches spread over the image** can be deployed to achieve denoising. It's even more expensive than the bilateral filter, since it has to look at more of the image.

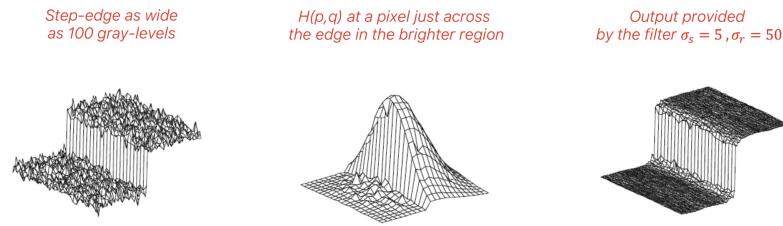


Figure 17: Bilateral filter differences

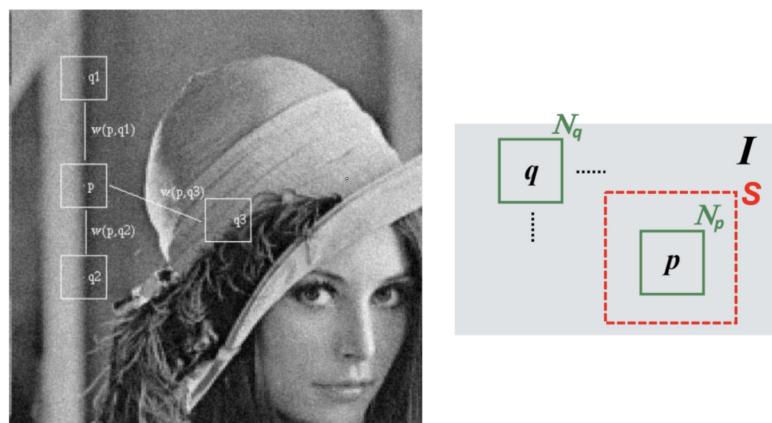


Figure 18: Non-local means filter



Figure 19: Difference between Gaussian filter and non-local means filter

2.3 Edge detection

Edge points are local features of the image that capture important information related to its semantic content. Edges are pixels that lie exactly in between image regions of different intensities.

1D step-edge We can detect an edge with a sharp change of a 1D signal (1D step-edge). The simplest edge-detection operator relies on thresholding the absolute value of the derivative of the signal.

2D step-edge A 2D step-edge is characterized not only by its strength but also by its direction (diagonal edge). The operator which allows us to sense the edge in any direction is the gradient: $\nabla I(x, y) = \frac{\partial I(x, y)}{\partial x} i + \frac{\partial I(x, y)}{\partial y} j$. Gradient's direction gives the direction along which the function exhibits its maximum variation. We can use differences to approximate the gradient (since we are in the discrete world and we talk about pixels, not functions). We can estimate the magnitude of the gradient by using different approximations. We choose $|\nabla I|_{\max} = \max(|I_x|, |I_y|)$, since it's fast and invariant with respect to the edge direction.

Edges and noise In real images an edge will not look as smooth as we have seen due to noise. Taking derivatives of noisy signals is an ill posed problem (the solution is not robust with respect to the input variation) since derivatives amplify noise. Noise robustness is usually achieved by smoothing the signal before computing the derivatives required to highlight edges. Smoothing has the side effect of blurring true edges, making it more difficult to detect and localize them, since pixel of different colors will look similar after filtering because of the blending.

Prewitt and Sobel Prewitt and Sobel address the challenge of detecting edges in noisy environments by combining smoothing and differentiation into a single step. The Prewitt and Sobel operators perform spatial filtering by using 3×3 convolution kernels. Each kernel smooths the image in one direction (reducing noise) and computes the derivative in the perpendicular region (highlighting edges). Both operators use two kernels: one detects vertical edges, and one horizontal ones. After convolving the image with both kernels, the gradient magnitude and direction are computed.

$$\text{Magnitude} = \sqrt{G_x^2 + G_y^2} \quad \text{Direction} = \arctan\left(\frac{G_y}{G_x}\right)$$

Prewitt is better for precise edge localization in cleaner images. Sobel is preferable in noisy images but may slightly blur edges.

2.3.1 Non-Maxima Suppression (NMS)

Detecting edges by gradient thresholding is inherently inaccurate, since it's difficult to choose the right threshold a priori. A better approach to detect edges may consist in finding the local maxima of the absolute value of the derivative of the signal.

When dealing with 2D images, we should look for:

- Maxima of the absolute value of the derivative (gradient magnitude).
- Along the gradient direction (orthogonal to the edge direction).

We don't know in advance the correct direction to carry out Non-Maxima Suppression (NMS). The direction has to be estimated locally based on the gradient's direction. The magnitude of the gradient has to be estimated at points which do not belong to the discrete pixel grid. Such values can be estimated by linear interpolation of those computed at the closest point belonging to the grid.

A final thresholding step on the magnitude of the gradient at the points selected by the NMS process typically helps pruning out unwanted edges due to either noise or less important details.

2.3.2 Canny's edge detector

Canny proposed to set forth quantitative criteria to measure the performance of an edge detector and then to find the optimal filter with respect to such criteria. The three criteria he proposed were:

- **Good detection:** correctly extract edges in noisy images.
- **Good localization:** distance between found and true edge should be minimum.
- **One response to one edge:** detect one single edge pixel at each true edge.

A straightforward Canny edge detector can be achieved by:

- Gaussian smoothing.
- Gradient computation.
- NMS along the gradient direction.

2D convolution by a Gaussian can be slow, so we can leverage on separability of the Gaussian function to speedup the calculation: $G(x, y) = G(x)G(y)$.

Non Maxima-Suppression (NMS) is often followed by thresholding of gradient magnitude to help distinguish between true "semantic" edges and unwanted ones. Edge **streaking** may occur when magnitude varies along object contours.

Canny proposed a "hysteresis" thresholding approach relying on a higher T_h and a lower T_l threshold. A pixel is taken as an edge if either the gradient magnitude is higher than T_h or higher than T_l and the pixel is a neighbor of an already detected edge.

The hysteresis thresholding is usually carried out by tracking edge pixels along contours. First the edge candidates are provided by NMS, then all strong edges are picked, and then for each strong edge we track the weak edges along contours.

Zero-crossing Instead of maximum values we can look for zero-crossing of the second derivative of the signal to locate edges (instead of the peaks of the first derivative). This requires significant computational effort since we are calculating second derivatives.

2.3.3 Second derivative along the gradient & Laplacian

The second derivative along the gradient's direction can be obtained as $n^T H n$.

- **Unit vector along the gradient's direction** $n = \frac{\nabla I(x, y)}{\|\nabla I(x, y)\|}$
- **Hessian matrix** $H = \begin{bmatrix} \frac{\partial^2 I(x, y)}{\partial x^2} & \frac{\partial^2 I(x, y)}{\partial x \partial y} \\ \frac{\partial^2 I(x, y)}{\partial y \partial x} & \frac{\partial^2 I(x, y)}{\partial y^2} \end{bmatrix}$

Computing the second derivative along the gradient turns out very expensive.

Discrete Laplacian We can use the forward and backward differences to approximate first and second order derivatives. It can be shown that the zero-crossing of the Laplacian typically lay close to those of the second derivative along the gradient. Yet, the former differential operator is much faster to compute.

2.3.4 Laplacian of Gaussian (LoG)

A robust edge detector should include a smoothing step to filter out noise. Edge detection by using the Laplacian of Gaussian (LoG) can be summarized as follows:

- Gaussian smoothing: $\tilde{I}(x, y) = I(x, y) * G(x, y)$.
- Second order differentiation by the Laplacian.
- Extraction of the zero-crossing of $\nabla^2 \tilde{I}(x, y)$

Practical implementations of the LoG may deploy the properties of convolutions to speed-up the computation.

We can find sign changes from minus to plus (and viceversa) between two consecutive pixels. Once a sign change is found, the actual edge may be localized:

- At the pixel where the LoG is positive (darker side of the edge)
- At the pixel where the LoG is negative (brighter side of the edge)
- At the pixel where the **absolute** value of the LoG is smaller (it's the best choice, since the edge turns out closer to the true zero-crossing).

Using a higher σ yields less details, so usually we use it for noisy images.

2.4 Feature detection

We can create a larger image by aligning two images of the same scene. Aligning two images may be done with a homography, which requires at least 4 feature correspondences. Feature detection is also useful for object detection, since we can look for features in the source image and try to find them again in the target image.

Several computer vision tasks deal with finding "Corresponding Points" between two (or more) images of a scene. Correspondences are image points which are the projection of the same 3D point in different views of the scene. Establishing correspondences may be difficult, as the points may look different in different views.

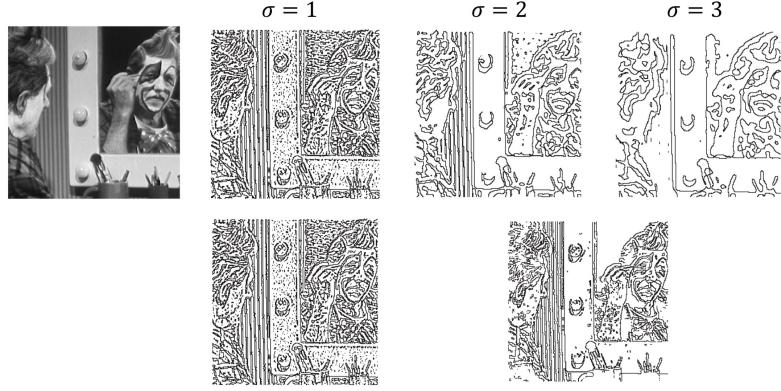


Figure 20: Examples of LoG application

2.4.1 Local invariant features paradigm

The task of establishing correspondences is split into 3 successive steps:

- **Detection** of salient points.
- **Description**: computation of a suitable descriptor based on pixels in the keypoint neighbourhood.
- **Matching** descriptors between images.

Descriptors should be **invariant** to as many transformations as possible. Ideally we want scale/rotation/illumination invariance.

2.4.2 Properties of good detectors/descriptors

- Detector
 - Repeatability: it should find the same keypoints in different views of the scene despite the transformations.
 - Saliency: it should find keypoints surrounded by informative patterns.
- Descriptor
 - Distinctiveness vs. robustness trade-off: the algorithm should capture the salient information around a keypoint.
 - Compactness: the description should be as concise as possible.

Speed is desirable for both, and in particular for detectors, which need to be run on the whole image (while descriptors are computed at keypoints only).

Edge pixels can be hardly told apart as they look very similar along the direction perpendicular to the gradient. Edges are locally ambiguous, since there are many other points that look just the same.

2.4.3 Moravec Interest Point Detector

We can define the **cornerness** at a point p , which is given by the minimum squared difference between the patch centered at p and those centered at its 8 neighbours: $C(p) = \min_{q \in n_8(p)} \|N(p) - N(q)\|^2$.

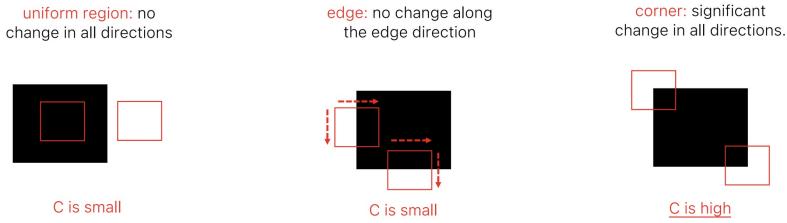


Figure 21: Cornerness of an image segment

After computing the cornerness we threshold and do Non-Maxima Suppression.

2.4.4 Harris Corner Detector

Harris & Stephens proposed to rely on a continuous formulation of the Moravec's "error" function. Assume to shift the image with a generic infinitesimal shift $(\Delta x, \Delta y)$: $w(x, y)$ is a window set to 1 around the pixel under evaluation and 0 in all the image. In the end we consider only the pixel around the (x, y) position.

Due to the shift being infinitesimal, we can deploy Taylor's expansion of the intensity function at $(x, y) : f(x + \Delta x) = f(x) + f'(x)\Delta x$

M encodes the local image structure around the considered pixel. We hypothesize that M is a diagonal matrix $M = \begin{bmatrix} \sum_{x,y} w(x, y)I_x(x, y)^2 & \sum_{x,y} w(x, y)I_x(x, y)I_y(x, y) \\ \sum_{x,y} w(x, y)I_y(x, y)I_x(x, y) & \sum_{x,y} w(x, y)I_y(x, y)^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$

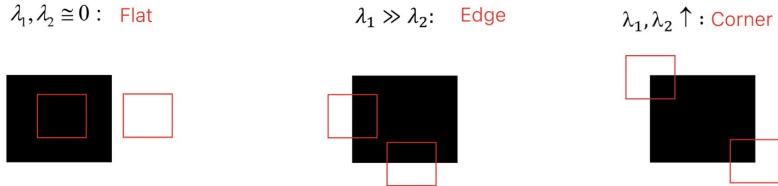


Figure 22: Harris corner detector and lambda values

The previous considerations have general validity as M is real and symmetric, and thus can always be diagonalized by a rotation of the image coordinate system.

$$M = R \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R^T$$

The columns of R are the orthogonal unit eigenvectors of M . λ_i are the corresponding eigenvalues. R^T is the rotation matrix that aligns the image axes to the eigenvectors of M .

Computing eigenvalues at each pixel is costly, so we can compute a more efficient "cornerness" function that gives a result C that is positive on a corner, negative on an edge and about 0 on a flat surface.

The Harris corner detection algorithm can thus be summarized as follows:

1. Compute C at each pixel.
2. Select all pixels where C is higher than a chosen positive threshold T .
3. Within the previous set, detect as corners only those pixels that are local maxima of C (NMS).

Eigenvalues of M are invariant to a rotation of the image axes, and thus so is Harris cornerness function. It also invariant to additive intensity changes. However, it's not invariant to scale changes, as corner structures may degrade or transform into edges under significant scaling, nor to multiplicative intensity variations which scale gradients and alter the corner response magnitude unless thresholds are adaptively adjusted.

2.4.5 Scale-Space

Scale invariance is the main issue addressed by second generation local invariant features. The key idea is to apply a fixed-size detection tools on increasingly down-sampled and smoothed versions of the input image.



Figure 23: We increase the blur as we reduce the image size

As we move along scales, small details should continuously disappear and no structure should be introduced.

A Scale-Space is a one-parameter family of images created from the original one so that the structures at smaller scales are successively suppressed by smoothing operations.

A Scale-Space must be realized by Gaussian Smoothing $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$. A Scale-Space is created by repeatedly smoothing the original image with larger and larger Gaussian kernels.

Feature Detection & Scale Selection The Gaussian Scale-Space is only a tool to represent the input image at different scales. It neither includes any criterion to detect features nor to select their characteristic scale. As features do exist across a range of scales how do we establish at which scale a feature turns out maximally interesting and should therefore be described?

The fundamental research work on multi-scale feature detection and automatic scale selection was proposed in "Feature detection with automatic scale selection" by T. Lindeberg in 1998.

As we filter more by using a higher sigma, derivatives tends to become weaker. To compensate, Lindeberg proposed to multiply/normalize derivatives by sigma (doing scale-normalization).

Scale-Normalized LoG The scale-normalized Laplacian Of Gaussian (LoG) is: $F(x, y, \sigma) = \sigma^2 \nabla^2 L(x, y, \sigma) = \sigma^2 (\nabla^2 G(x, y, \sigma) * I(x, y))$, where σ^2 is the normalization factor.

The scale-normalized LoG detects blobs (regions with intensity variations) by combining Gaussian Smoothing (which reduces noise and suppresses fine details) and Laplacian Operator (which highlights regions of rapid intensity change).

Scale normalization is introduced since the raw LoG response diminishes as the scale (σ) increases, biasing detection toward smaller blobs.

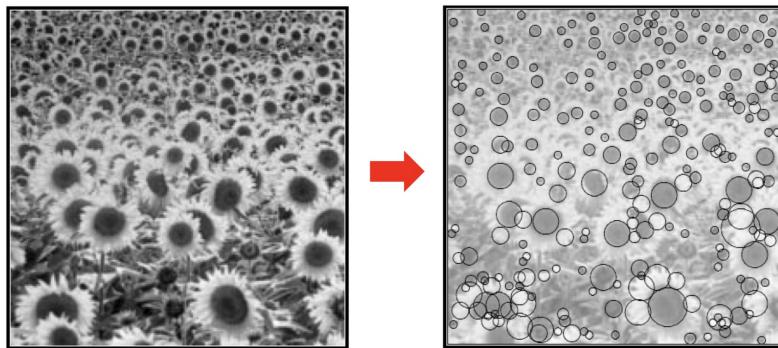


Figure 24: LoG filter locates blobs

Difference of Gaussian (DoG) Lowe proposed to detect keypoints by seeking for the extrema of the DoG (Difference of Gaussian) function across the (x, y, σ) domain.

$$DoG(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

This approach provides a computationally efficient approximation of Lindeberg's scale-normalized LoG.

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G(x, y, \sigma) * I(x, y)$$

Both detectors are rotation invariant and find blob-like features.

In the DoG we compute several Gaussian smoothed functions within an octave. For each pair we take the differences, then we seek for extrema in the DoG.

Extrema detection A point (x, y, σ) is detected as a keypoint if and only if its DoG is higher (or lower) than that of the 26 neighbour (8 at the same scale and 18 = 9 + 9 at the two nearby scales) in the (x, y, σ) space.

According to the original article the best number of scales within an octave is $s = 3$, initial σ for each octave should be $\sigma = 1.6$ and the input image is enlarged by a factor of 2 in both dimensions.

After detecting points as local extrema, the next step is to prune weak and unstable responses to ensure robustness. Keypoints with low contrast, corresponding to weak DoG responses, are often sensitive to noise and less repeatable across different images.

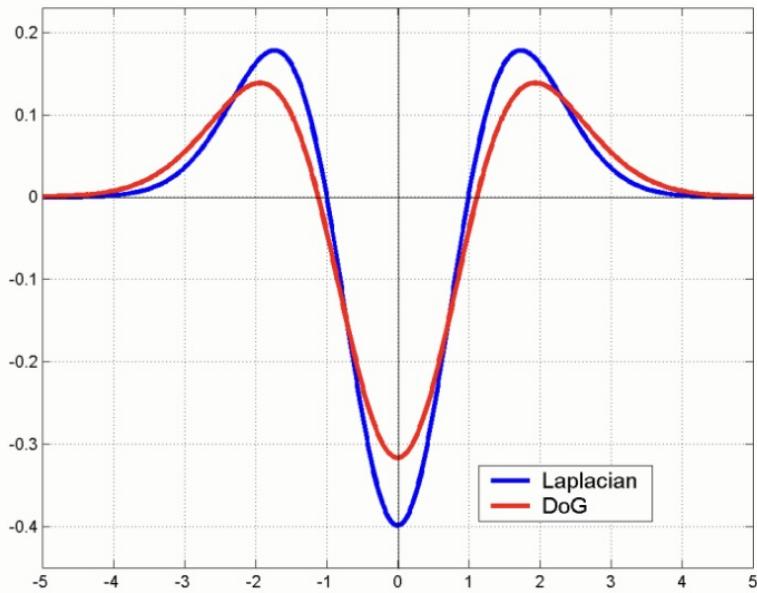


Figure 25: Lowe proves that this is a scaled version of Lindeberg

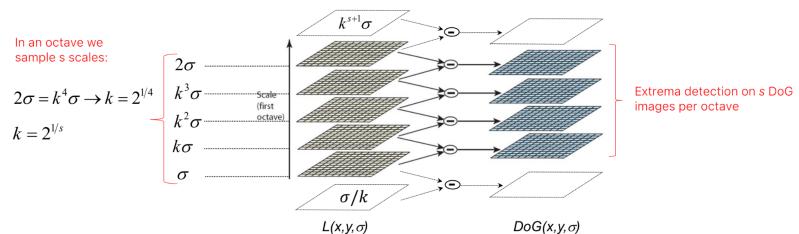


Figure 26: Difference of Gaussian

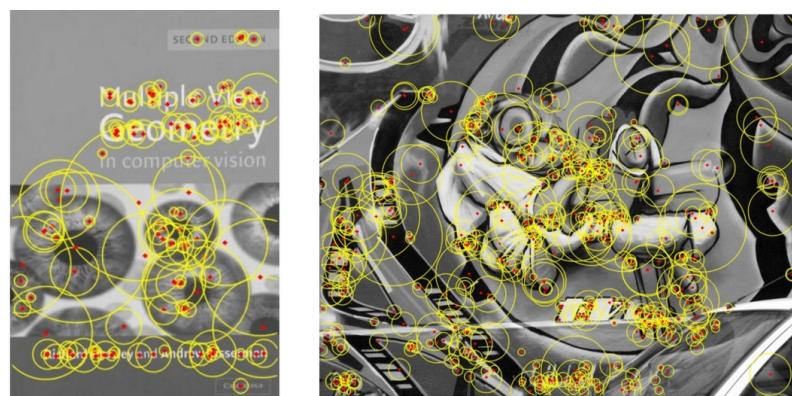


Figure 27: The size of the circle is proportional to the scale of the feature (σ)

2.4.6 Scale and Rotation Invariance Description

DoG are rotation invariant because of circular symmetry. Once each keypoint has been extracted, a surrounding patch is considered to compute its descriptor (scale and rotation invariant).

We need to identify a prominent direction inherent to the patch (the canonical orientation) and, based on such direction, define a local reference frame. A reasonable choice consists in identifying the direction along which most of the gradient is found.

To compute the canonical orientation we need to define a scale and rotation invariant description. Once each keypoint has been extracted, a surrounding patch is considered to compute its descriptor (scale and rotation invariant).

- Scale invariance: the patch is taken from the stack of images that correspond to the characteristic scale.
- Rotation invariance: a canonical patch orientation is computed, so that the descriptor can then be computed on a canonically-oriented patch (orientation with respect to a new reference system, not the one of the image).

Lowe proposed to compute the canonical orientation of DoG keypoints as follows: given the keypoint, the magnitude and orientation of the gradient are computed at each pixel of the associated Gaussian-smoothed image, L :

- $m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$
- $\theta(x, y) = \tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$

We can then compute an orientation histogram by accumulating the contributions of the pixels belonging to a neighborhood of the keypoint location (a bin size of 10 deg). The characteristic orientation of the keypoint is given by the highest peak of the orientation histogram. Other peaks higher than 80% of the main one would be kept as well. A keypoint may have multiple canonical orientations and, in turn, multiple descriptors sharing the same location/scale with diverse orientations (this happen rarely, about on 15% of the keypoints).

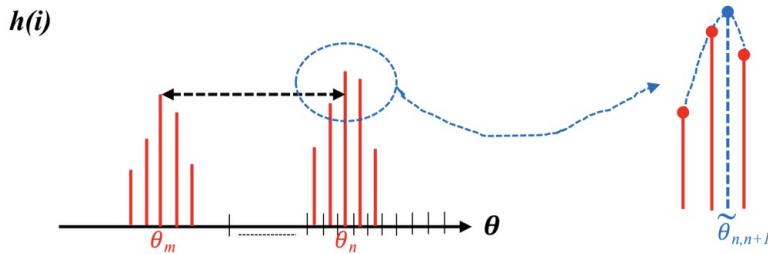


Figure 28: This is an ambiguous keypoint, we have two prominent decisions

2.4.7 SIFT Descriptor

The SIFT (Scale Invariant Feature Transform) descriptor is computed as follows:

- 16 × 16 oriented pixel grid around each keypoint is considered.
- This is further divided into 4 × 4 regions (each of size 4 × 4 pixels).
- A gradient orientation histogram is created for each region.
- Each histogram has 8 bins.
- Each pixel in the region contributes to its designated bin according to gradient magnitude and gaussian weighting function centered at the keypoint (with σ equal to half the grid size).

Matching process The descriptor is normalized to unit length to gain invariance with respect to affine intensity changes. Descriptors like SIFT are compared across diverse views of a scene to find corresponding keypoints. This is a classical Nearest Neighbour (NN) search problem. Given a set S of points, p_i , in a metric space M and a query point $q \in M$, find the p , closest to q . We wish to match the local features computed from an image under analysis (target image T) to those already computed from a reference image (R) or a set of reference images.

For each feature in T we look for the most similar one in R :

- The features in T represent the query points, q .
- The features in R provide set S .
- When matching SIFT descriptors the distance typically used is the Euclidean distance.

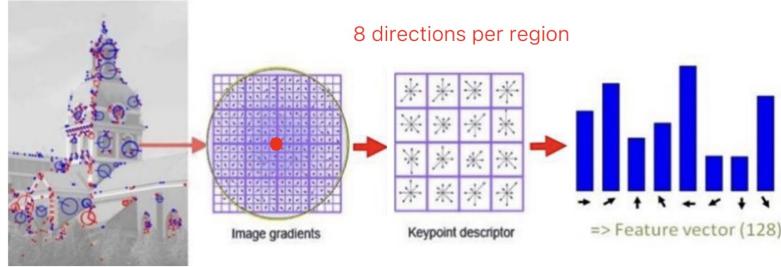


Figure 29: The descriptor size is given by the number of regions times the number of histogram bins per region

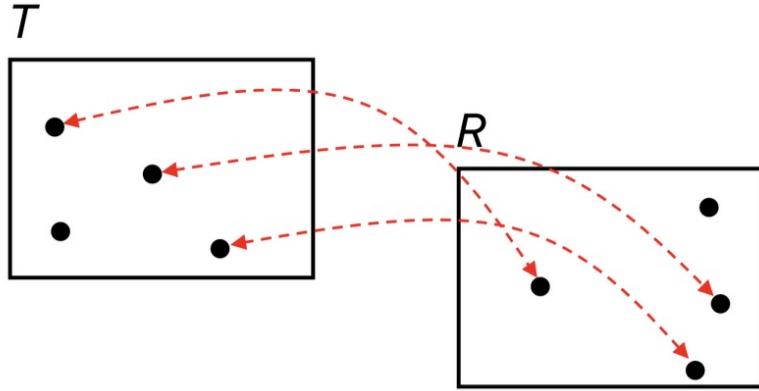


Figure 30: Image matching process with SIFT

The found NN does not necessarily provide a valid correspondence as some features in T may not have a corresponding feature in R . We enforce a criteria to accept/reject a match found by the NN search process. The simplest thing to do is to use a threshold. Lowe showed that $T = 0.8$ may allow for rejecting 90% of the wrong matches while missing only 5% of those correct.

Exhaustively searching for the NN of the query feature, q , has linear complexity in the size of S . This is slow, so efficient indexing techniques are exploited to speed-up the NN-search process. The main indexing technique exploited for feature matching is known as k-d tree.

3 Advanced Topics in Deep Learning for Computer Vision