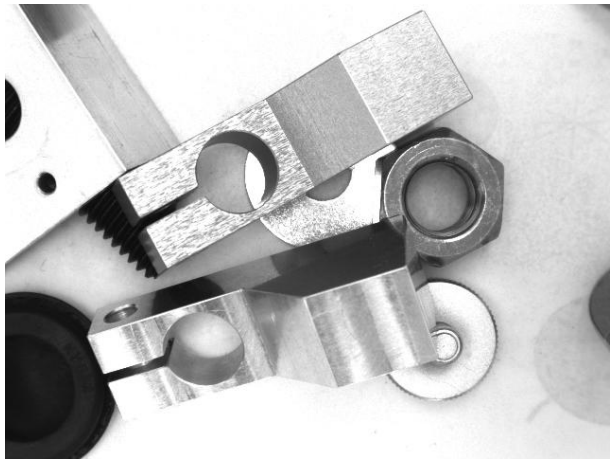
University of Bologna

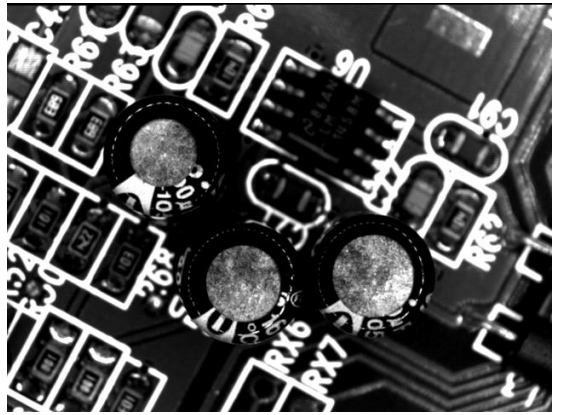
Local Features (Part 1)

Luigi Di Stefano (luigi.distefano@unibo.it)

But segmentaion may fail….

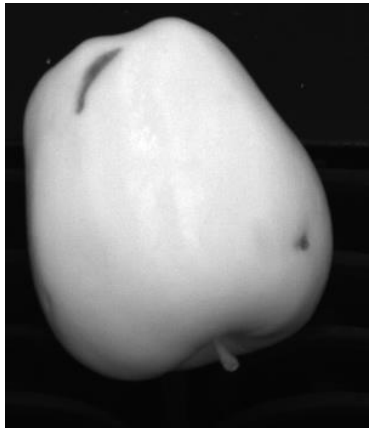
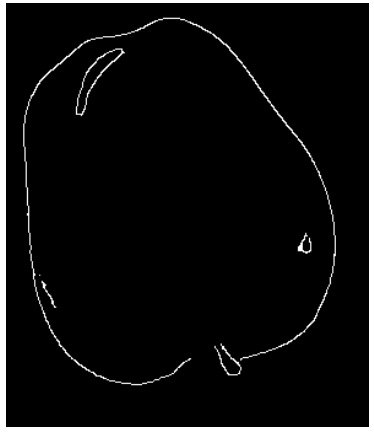
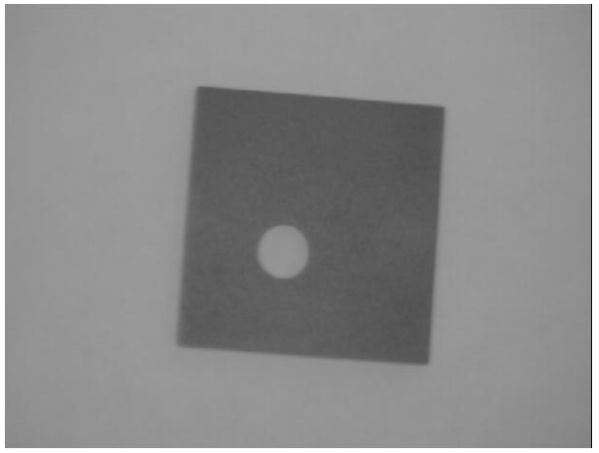
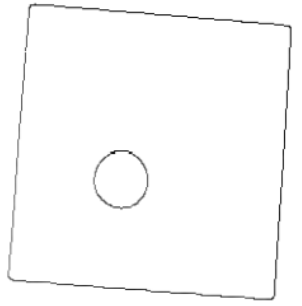


Major nuisances: brightness/colour variations across objects, uneven lighting, specular surfaces, occlusions, clutter…..

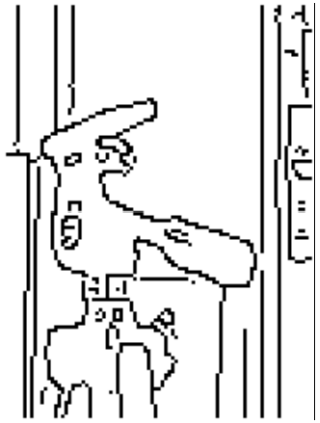


...rely on *local* features directly extracted from the input gray-scale/colour image !

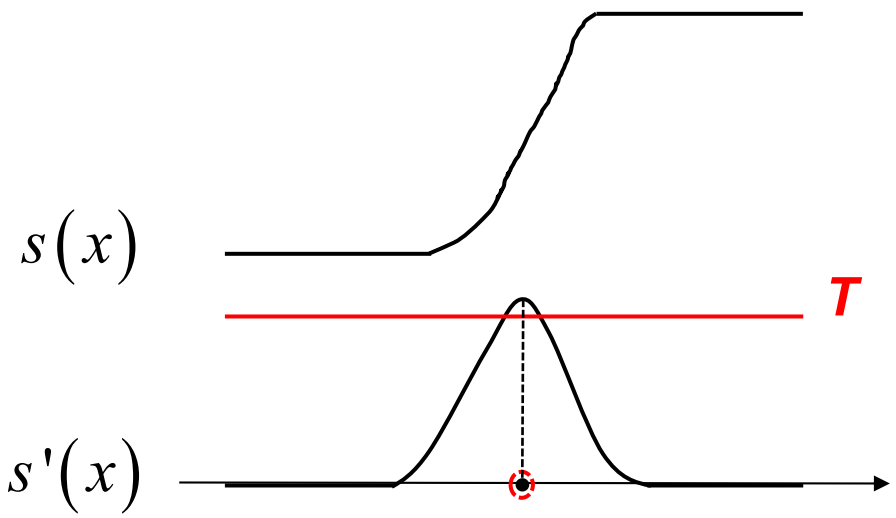
Edges

- ***Edge*** **or** ***contour points*** **are local features of the image that capture important information related to its semantic content.**
- **Edges are pixels that can be thought of as lying exactly in between image regions of different intensity, or, in other words, to separate different uniformly colored regions.**
- **Edge features are key to measurement tools in industrial vision.**

1D Signal Edge: Strong Variation (Derivative)



In the transition region between the two constant levels the absolute value of the derivative of the signal is high (and reaches a maximum at the inflection point).

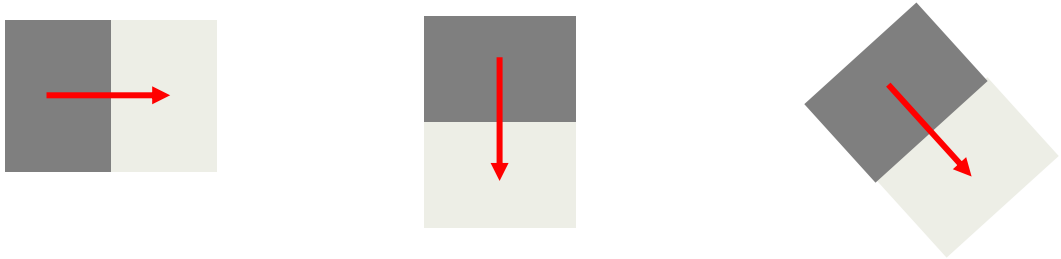
Detect peaks of the (absolute) derivative of the signal.

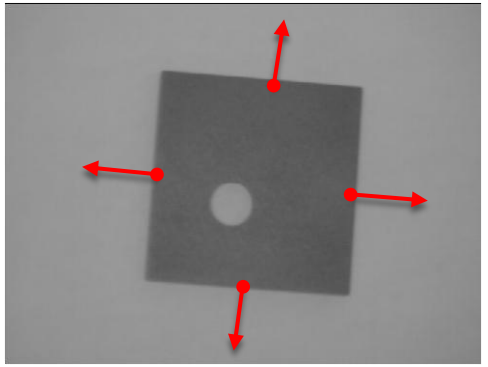
To detect only strong enough edges, keep the peaks where the (absolute) derivative is above a certain threshold T).

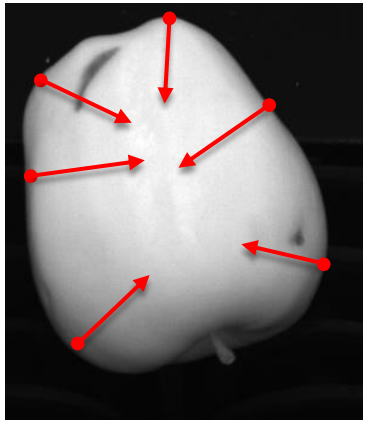
What about detecting edges in images (2D signals) then ?

Image Edges as Strong Gradient Maxima

To detect image edges we should find those pixels where there exist a strong variation (i.e. absolute derivative) *along a certain direction*.







The *Gradient* at pixel *(x,y)*: $\nabla I(x,y) = \frac{\partial I(x,y)}{\partial x}\mathbf{i} + \frac{\partial I(x,y)}{\partial y}\mathbf{j}$

is a vector directed along the maximal image variation whose *magnitude* is the *absolute derivative* along that direction. Indeed, a generic *directional* derivative can be computed by the *dot product* between the gradient and the unit vector along the direction.

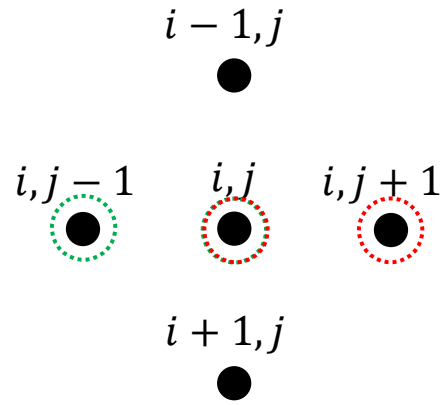
→ ***Detect (strong) maxima of gradient magnitude***

Discrete Approximation of the Gradient

- **We can use either *backward* or *forward* differences:**

$$\frac{\partial I(x,y)}{\partial x} \cong I_x(i,j) = I(i,j) - I(i,j-1), \qquad \frac{\partial I(x,y)}{\partial y} \cong I_y(i,j) = I(i,j) - I(i-1,j)$$

$$\frac{\partial I(x,y)}{\partial x} \cong I_x(i,j) = I(i,j+1) - I(i,j), \qquad \frac{\partial I(x,y)}{\partial y} \cong I_y(i,j) = I(i+1,j) - I(i,j)$$



which may be thought of as *correlation* by:

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$(0,0)$

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$(0,0)$

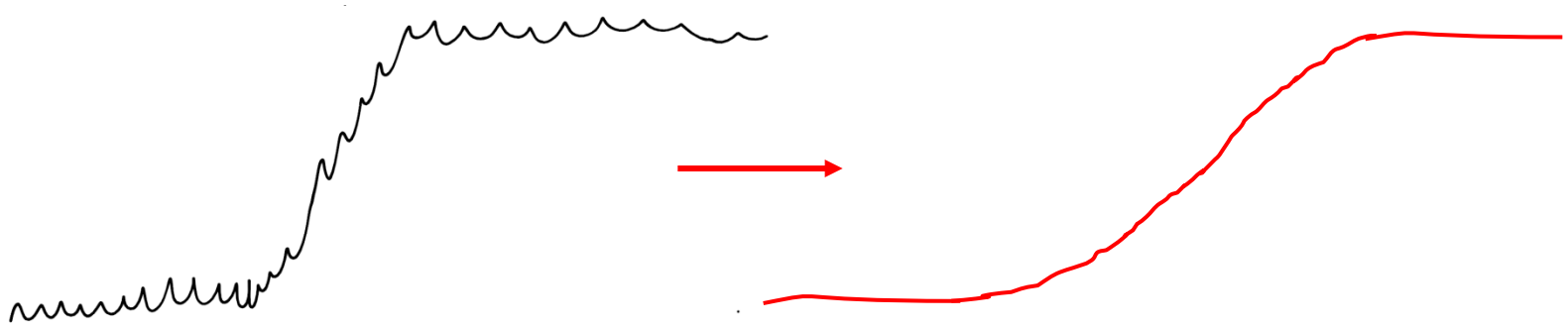
- **Or also central differences:**

$$I_x(i,j) = I(i,j+1) - I(i,j-1), \qquad I_y(i,j) = I(i+1,j) - I(i-1,j)$$

with the corresponding correlation kernels given by: $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ $(0,0)$ $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

Dealing with Noise

- **Due to noise, in real images an edge will likely look as depicted in the picture below**



which makes it hard to detect the "true" edge despite the many spurious variations due to noise. Indeed, it is well-known that taking derivatives of noisy signals is an *ill-posed* problem: due to derivatives amplifying noise, a small change in the input yields a very different output.

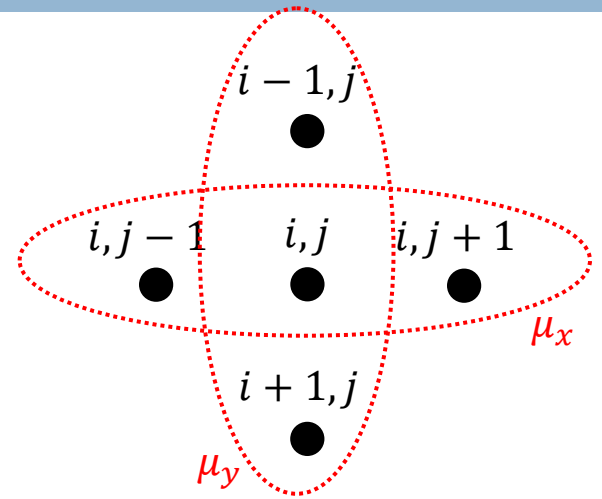
- **To work with real images, an edge detector should therefore be robust to noise, so as to highlight the meaningful edges only and filter out effectively the spurious variations caused by noise.**

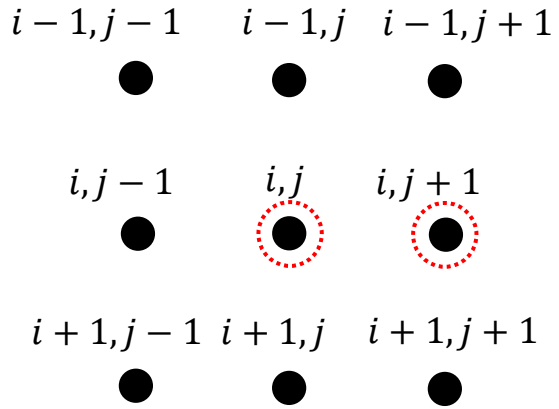
- **This is usually achieved by smoothing the signal before computing the derivatives required to highlight edges. Unfortunately, smoothing yields also the side-effect of blurring true edges, making it more difficult to detect and localize them accurately.**

Smooth Derivatives (1)

Smoothing and differentiation can be carried out jointly within a single step. This is achieved by computing *differences of means* (rather than smoothing the image and then computing differences). To avoid blurring edges, the two operations are carried out along orthogonal directions.

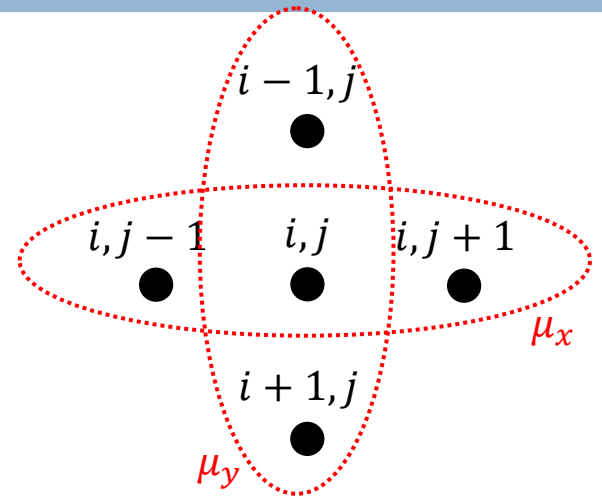
$$\mu_x(i,j) = \frac{1}{3}[I(i,j-1) + I(i,j) + I(i,j+1)] \qquad \mu_y(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)]$$





Smooth Derivatives (2)

Smoothing and differentiation can be carried out jointly within a single step. This is achieved by computing *differences of means* (rather than smoothing the image by a mean filter and then computing differences). To avoid blurring edges, the two operations are carried out along orthogonal directions.

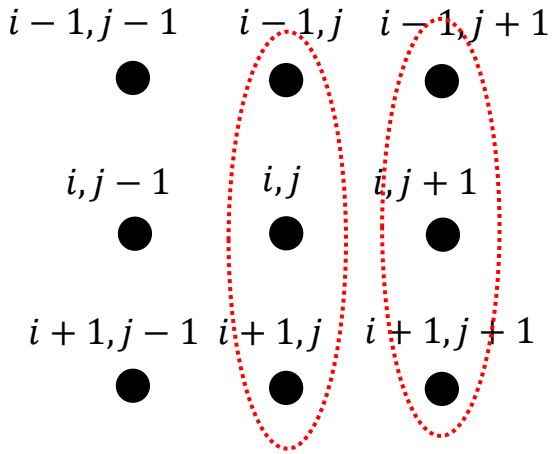


$$\mu_x(i,j) = \frac{1}{3}[I(i,j-1) + I(i,j) + I(i,j+1)] \qquad \mu_y(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)]$$

$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j)$$

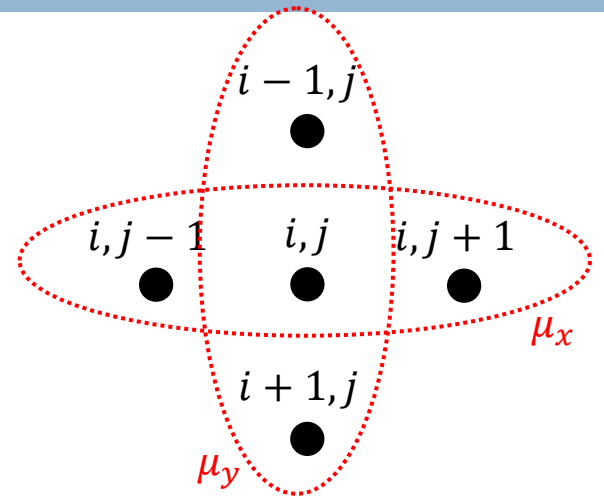
$$= \frac{1}{3}[I(i-1,j+1) + I(i,j+1) + I(i+1,j+1) - I(i-1,j) - I(i,j) - I(i+1,j)]$$

$$\Rightarrow \frac{1}{3}\begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$



Smooth Derivatives (3)

Smoothing and differentiation can be carried out jointly within a single step. This is achieved by computing *differences of means* (rather than smoothing the image by a mean filter and then computing differences). To avoid blurring edges, the two operations are carried out along orthogonal directions.

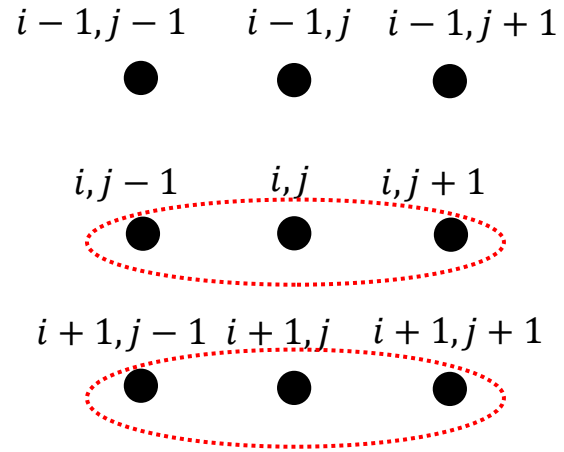


$$\mu_x(i,j) = \frac{1}{3}[I(i,j-1) + I(i,j) + I(i,j+1)] \qquad \mu_y(i,j) = \frac{1}{3}[I(i-1,j) + I(i,j) + I(i+1,j)]$$

$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j)$$

$$= \frac{1}{3}[I(i-1,j+1) + I(i,j+1) + I(i+1,j+1) - I(i-1,j) - I(i,j) - I(i+1,j)]$$

$$\Rightarrow \frac{1}{3}\begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$



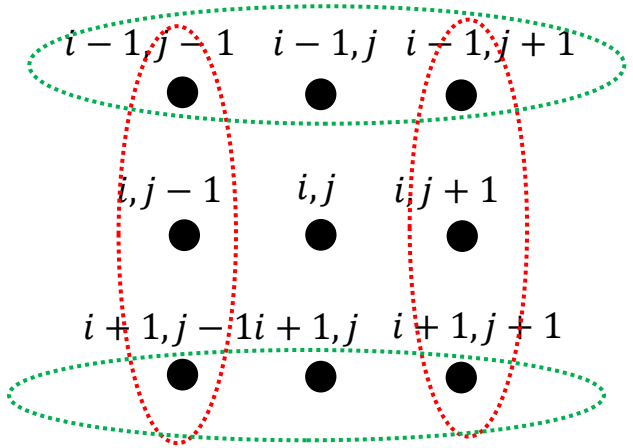
$$\tilde{I}_y(i,j) = \mu_x(i+1,j) - \mu_x(i,j)$$

$$= \frac{1}{3}[I(i+1,j-1) + I(i+1,j) + I(i+1,j+1) - I(i,j-1) - I(i,j) - I(i,j+1)]$$

$$\Rightarrow \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

Smooth Derivatives (4)

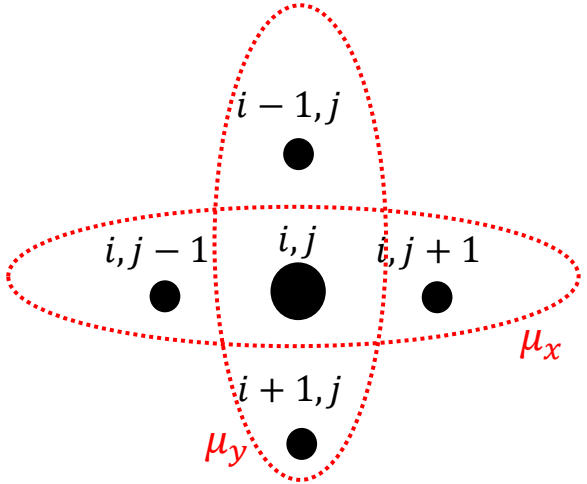
- **In the *Prewitt operator,* derivatives are approximated by central differences (better for *diagonal edges*):**



$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j-1) \quad \Rightarrow \quad \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\tilde{I}_y(i,j) = \mu_x(i+1,j) - \mu_x(i-1,j) \quad \Rightarrow \quad \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- **In the *Sobel operator*, the central pixel is weighted more to further improve *isotropy*:**



$$\mu_x(i,j) = \frac{1}{4}[I(i,j-1) + 2I(i,j) + I(i,j+1)]$$

$$\tilde{I}_x(i,j) = \mu_y(i,j+1) - \mu_y(i,j-1)$$

$$\Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

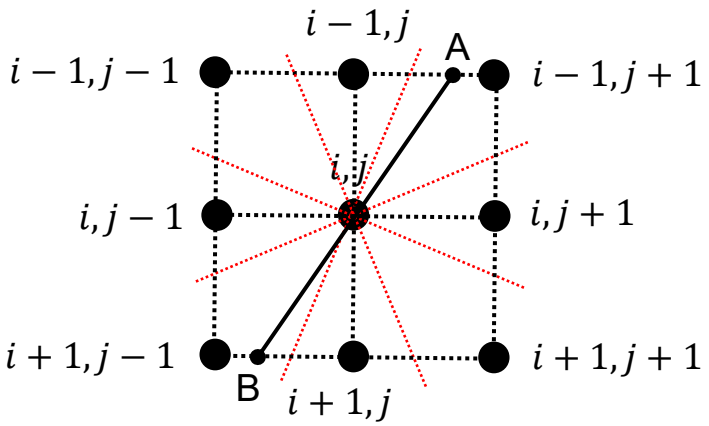
$$\mu_y(i,j) = \frac{1}{4}[I(i-1,j) + 2I(i,j) + I(i+1,j)]$$

$$\tilde{I}_y(i,j) = \mu_x(i+1,j) - \mu_x(i-1,j)$$

$$\Rightarrow \quad \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Non-Maxima Suppression (NMS)

- **Discrete formulation**



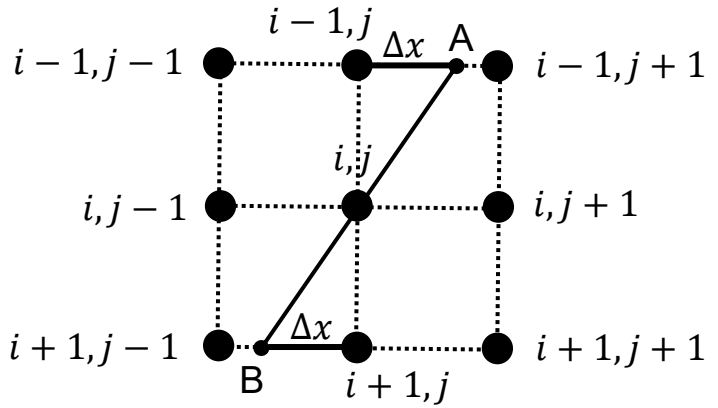
$$G = \|\nabla I(i,j)\|$$

$$G_A \cong \|\nabla I(i-1,j+1)\|$$

$$G_B \cong \|\nabla I(i+1,j-1)\|$$

$$NMS(i,j) = \begin{cases} 1: \ (G > G_A) \wedge (G > G_B) \\ 0: otherwise \end{cases}$$

- **Continuous formulation based on gradient interpolation**



$$G_1 = \|\nabla I(i-1,j)\|$$

$$G_2 = \|\nabla I(i-1,j+1)\|$$

$$G_3 = \|\nabla I(i+1,j-1)\|$$

$$G_4 = \|\nabla I(i+1,j)\|$$

$$G = \|\nabla I(i,j)\|$$

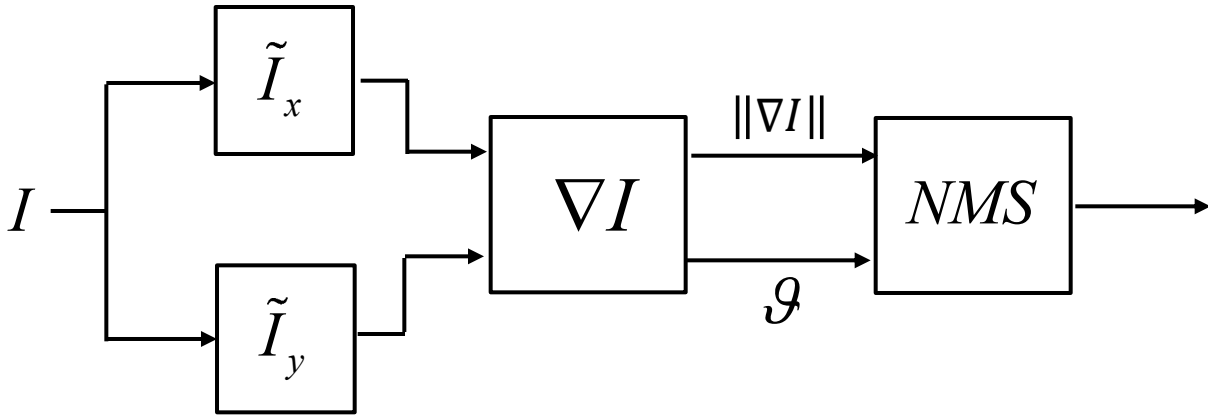
$$G_A \cong G_1 + (G_2 - G_1)\Delta x$$

$$G_B \cong G_4 + (G_3 - G_4)\Delta x$$

$$NMS(i,j) = \begin{cases} 1: \ (G > G_A) \wedge (G > G_B) \\ 0: otherwise \end{cases}$$

Edge Detection by Smooth Derivatives and NMS

The overall flow-chart of an edge detector based on smooth derivatives and NMS is sketched below:



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.

Canny's Edge Detector (1)

- Canny proposed to set forth quantitative criteria to measure the performance of an edge detector and then to find the optimal operator with respect to such criteria. Accordingly, he proposed the following three criteria:

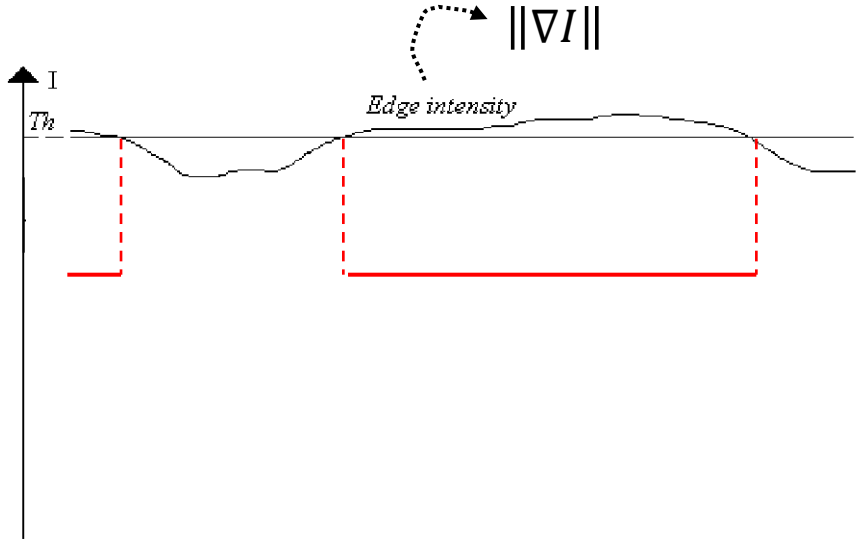
  1. *Good Detection*: the filter should correctly extract edges in noisy images.
  2. *Good Localization*: the distance between the found edge and the "true" edge should be minimum.
  3. *One Response to One Edge*: the filter should detect one single edge pixel at each "true" edge.

- Addressing the 1D case and modeling an edge as a noisy step, he shows that the optimal edge detection operation consists in finding local extrema of the convolution of the signal by a first order Gaussian derivative (i.e. $G'(x)$) . This theoretical result can also be regarded as a proof of the optimality of the Gaussian as smoothing filter to detect noisy edges.

- As usual, to end up with a practical 2D edge detector to be applied to images, we should look for local extrema of the directional derivative along the gradient. As such, a straightforward implementation of the Canny edge detector can be achieved by Gaussian smoothing followed by gradient computation and NMS along the gradient direction.

Canny's Edge Detector (2)

- As already discussed, NMS is often followed by thresholding of gradient magnitude to help distinguish between true "semantic" edges and unwanted ones. However, ***edge streaking*** may occur when gradient magnitude varies along object contours.

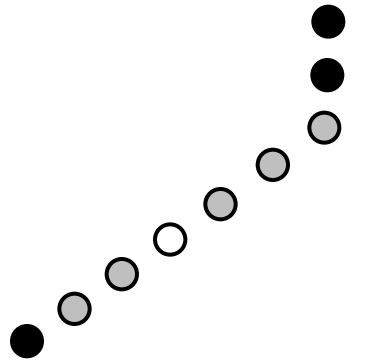


- To address the above issue, Canny proposed a **hysteresis thresholding** approach relying on a higher (T_h) and a lower (T_l) threshold. A pixel selected by NMS is taken as an edge if either the gradient magnitude is higher than T_h OR it is higher than T_l AND the pixel is a neighbour (according to 8-way connectivity) of an already detected edge.

Canny's Edge Detector (4)

- **Hysteresis thresholding is usually carried out by tracking edge pixels along contours, which also brings in the "side-effect" of Canny's providing as output chains of connected edge pixels rather than edge maps.**

Edge candidates provided by NMS



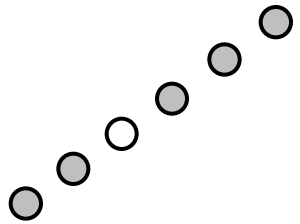
○ : $\|\nabla I\| > T_h$

○ : $\|\nabla I\| > T_l$

● : $\|\nabla I\| < T_l$

Step-1: pick up all strong edges

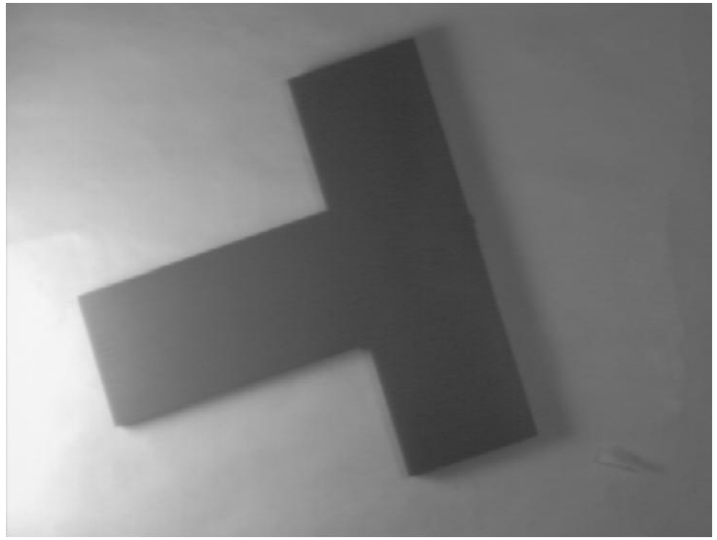
Step-2: for each strong edge track weak edges along contours



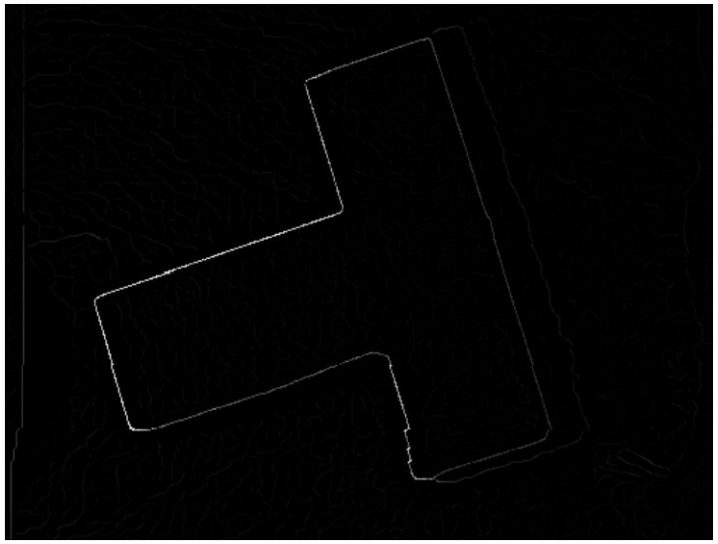
OpenCV: cv2.Canny

- Smooth derivatives by Sobel, no Gaussian Smoothing -> denoise the image by cv2.Gaussianblur before invoking cv2.Canny

Exemplar Results by Canny's

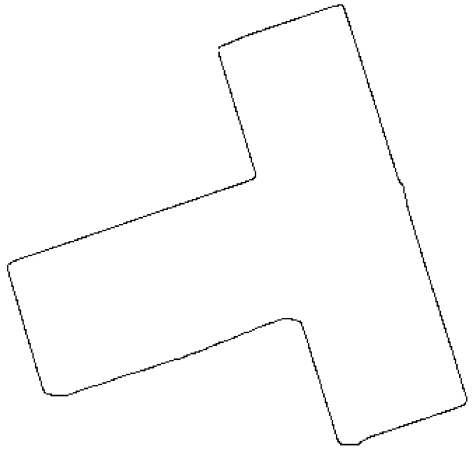


Input Image

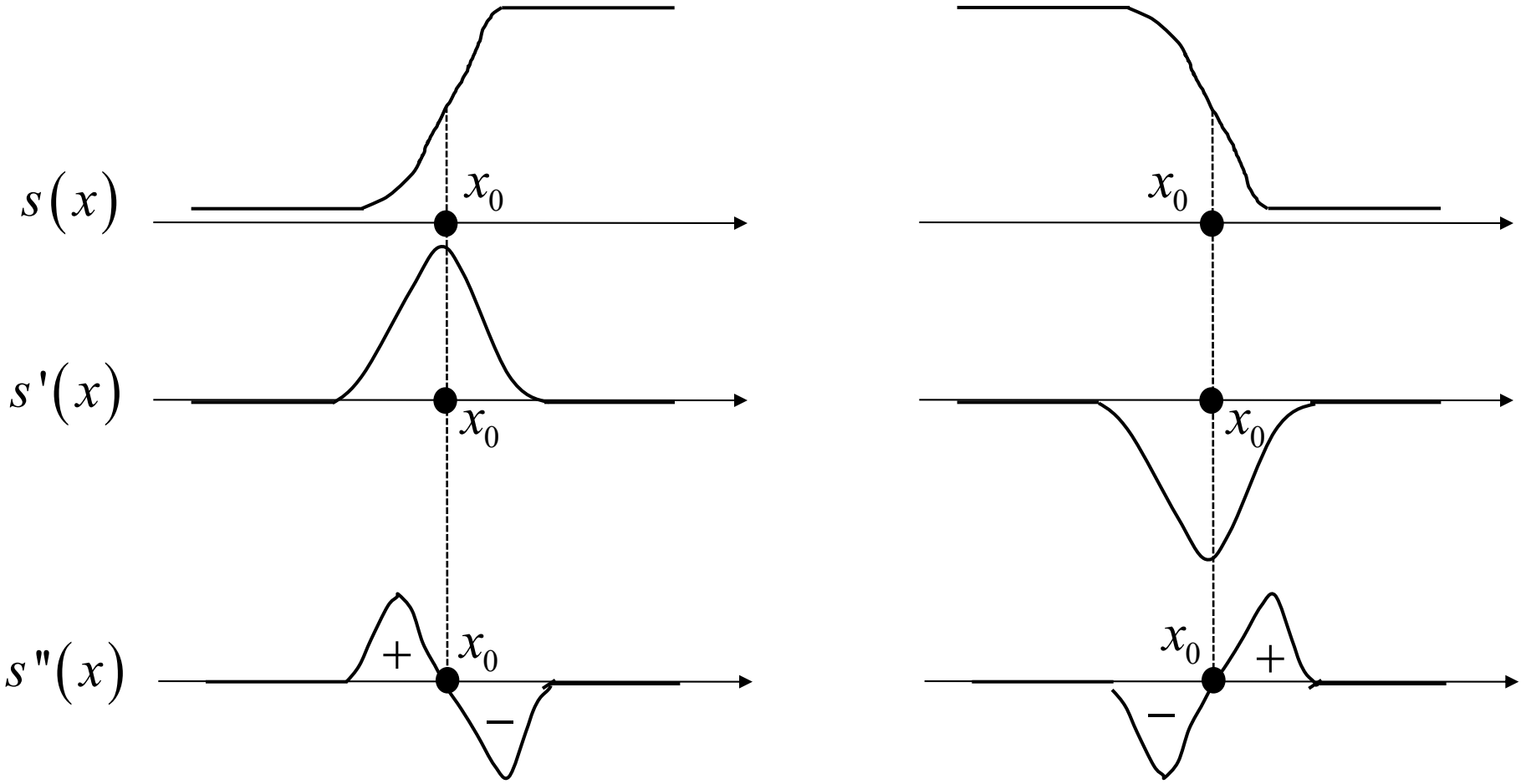


NMS output (with gradient magnitude represented by gray-scales)

Final Output



Zero-crossing of the second derivative



Edges may also be localizd by looking for zero-crossing of the second derivative of the signal.

Second derivative along the gradient & Laplacian

- **The second derivative along the gradient's direction can be obtained as** $\mathbf{n}^T\mathbf{H}\mathbf{n}$

with $\mathbf{n} = \dfrac{\nabla I(x,y)}{\|\nabla I(x,y)\|}$ **and** $\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 I(x,y)}{\partial x^2} & \dfrac{\partial^2 I(x,y)}{\partial x \partial y} \\ \dfrac{\partial^2 I(x,y)}{\partial y \partial x} & \dfrac{\partial^2 I(x,y)}{\partial y^2} \end{bmatrix}$

(unit vector along the gradient's direction)

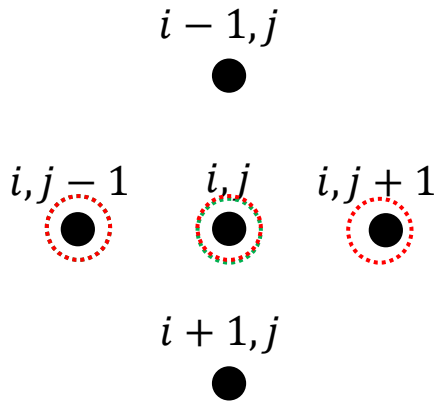
(Hessian matrix)

- **Computing the second derivative along the gradient turns out very expensive. In their seminal work on edge detection, instead, Marr & Hildreth proposed to rely on the Laplacian as second order differential operator:**

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} = I_{xx} + I_{yy}$$

Discrete Laplacian

- One can use forward and backward differences to approximate first and second order derivatives, respectively:



$$I(i,j+1) - I(i,j) \qquad I(i,j) - I(i,j-1)$$

$$I_{xx} \cong$$

$$I_{yy} \cong I_y(i,j) - I_y(i-1,j) = I(i-1,j) - 2I(i,j) + I(i+1,j)$$

- 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, i.e. just a convolution by a 3x3 kernel, than the latter.

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Laplacian of Gaussian (LOG)

- As already discussed, a robust edge detector should include a smoothing step to filter out noise (especially in case second rather than first order derivatives are deployed). In their edge detector, Marr&Hildreth proposed to use a Gaussian filter as smoothing operator. Hence, their edge detector is referred to as LOG (Laplacian of Gaussian).

- Edge detection by the LOG can the be summarized conceptually as follows:

1. Gaussian smoothing: $\tilde{I}(x,y) = I(x,y) * G(x,y)$
2. Second order differentiation by the Laplacian: $\nabla^2 \tilde{I}(x,y)$
3. Extraction of the zero-crossing of $\nabla^2 \tilde{I}(x,y)$ (see Appendix 1)

Practical implementations of the LOG may deploy the properties of convolutions to speed-up the computation (see Appendix 2).

Appendix 1 - Finding the zero-crossings of the LOG

- **Zero-crossing are usually sought for by scanning the image by both rows and columns to identify changes of the sign of the LOG.**

- **Once a sign change is found, the actual edge may be localized:**

  1. **At the pixel where the LOG is positive (darker side of the edge).**
  2. **At the pixel where the LOG is negative (brighter side of the edge).**
  3. **At the pixel where the absolute value of the LOG is smaller (the best choice, as the edge turns out closer to the "true" zero-crossing).**

- **To help discarding spurious edges, a final thresholding step may be enforced (usually based on the slope of the LOG at the found zero-crossing).**

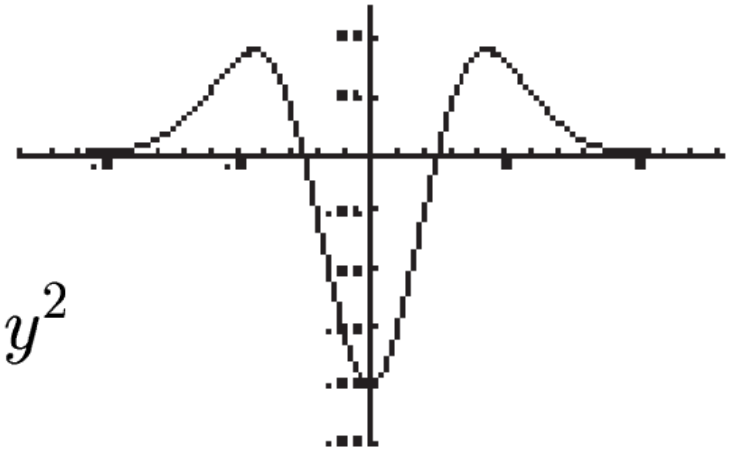
Appendix 2 – Computation of the LOG by a single 2D convolution

$$\nabla^2 \tilde{I}(x,y) = \nabla^2 \left(I(x,y) * G(x,y) \right) = I(x,y) * \nabla^2 G(x,y)$$

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$

$$= \frac{1}{2\pi\sigma^4} \left[\frac{r^2}{\sigma^2} - 2 \right] e^{-\frac{r^2}{2\sigma^2}}, \quad r^2 = x^2 + y^2$$

Mexican Hat filter



Appendix 2 – Computation of the LOG by four 1D convolutions

- **2D convolution by the Mexican Hat can be expensive in terms of computation, especially when the size of the filter is large. As it is the case of the Gaussian, the size of the filter, *d*, must increase with σ. According to several studies:**

$$3\omega \le d \le 4\omega, \quad \omega = 2\sqrt{2}\cdot\sigma$$

$$\begin{aligned}
\sigma = 0.5 \quad d = 4.24 \quad &\Rightarrow \quad 5x5 \\
\sigma = 1 \quad d = 8.48 \quad &\Rightarrow \quad 9x9 \\
\sigma = 2 \quad d = 16.97 \quad &\Rightarrow \quad 17x17 \\
\sigma = 3 \quad d = 25.45 \quad &\Rightarrow \quad 26x26 \\
\sigma = 4 \quad d = 33.94 \quad &\Rightarrow \quad 34x34 \\
\sigma = 5 \quad d = 42.42 \quad &\Rightarrow \quad 43x43
\end{aligned}$$

- **However, thanks to separabillty of the Gaussian, computing the LOG boils down to four 1D convolutions, which is substantially faster (*4d* rather that d^2 MACs per pixel):**

$$I(x,y) * \nabla^2 G(x,y) = I(x,y) * \left(G''(x)G(y) + G''(y)G(x)\right)$$

$$= I(x,y) * \left(G''(x)G(y)\right) + I(x,y) * \left(G''(y)G(x)\right)$$

$$= \left(I(x,y) * G''(x)\right) * G(y) + \left(I(x,y) * G''(y)\right) * G(x)$$

Main References

1) A. Rosenfeld, A. Kak, “Digital Picture Processing – Vol. 2, Academic Press, 1982.

2) R. Fisher, S. Perkins, A. Walker, E. Wolfart, “Hypermedia Image Processing Reference”, Wiley, 1996 (http://homepages.inf.ed.ac.uk/rbf/HIPR2/)

3) R. Schalkoff, “Digital Image Processing And Computer Vision, Wiley, 1989.

4) D. Marr, E. Hildreth, “Theory of Edge Detection”, Proc. Royal Society of London, 1980.

5) J. Canny, “A computational approach to edge detection” , IEEE Trans. On PAMI, 1986.