[CV] 3. Gradient and Laplacian Filter, Difference of Gaussians (DOG)

Jun Follow
Nov 18, 2020 · 8 min read

> *Previously, we have taken a look at filters used to smooth or to remove noise in images. In this chapter, the filters, which are to extract edge information in images, will be explained.*

Before we get started, **note that filters can be seen as templates**. This means that applying a filter at some point can be consider as taking a dot product between 1D representation of image region and 1D vector representation of the filter. **Therefore, the filter response is maximized when the region, where the filter is applied, has the same looking as filter.** In other words, filters look like the effects(or image features) they are intended to find.

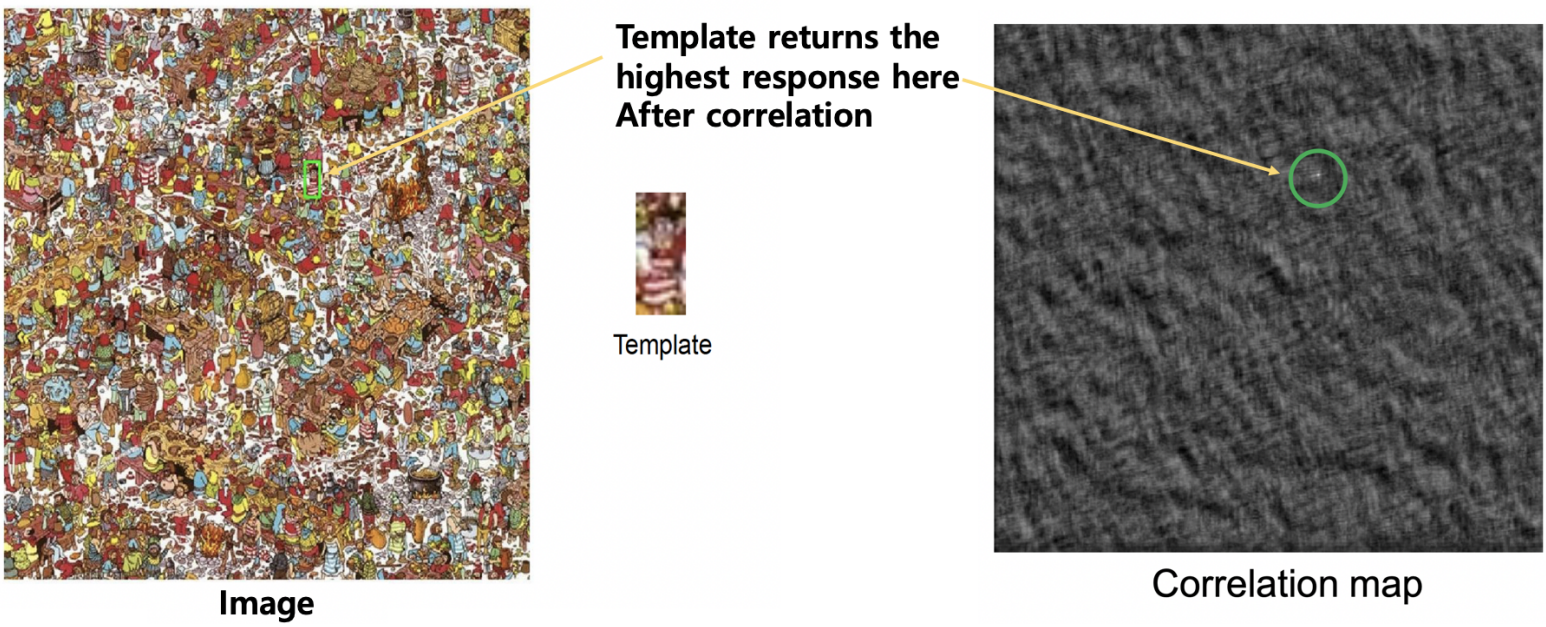


Figure 1. Example of the filter response given image and template, from [1], [2]

1. Gradient Filter

1.1 Image gradient

Gradients of each pixel in an image are useful to detect the edges, and therefore, Gradient filters are common choice to find edges. Then what makes gradient facilitating to detect edges and why is it useful?

That can be answered by the following example in Fig 2. The left panel is the given image, and the panel in the center is the corresponding pixel intensities for pixels on the red line of image. As we can observe from the two panels, the edges in image are where the pixel intensities largely change, for example, from white to black and from black to white.

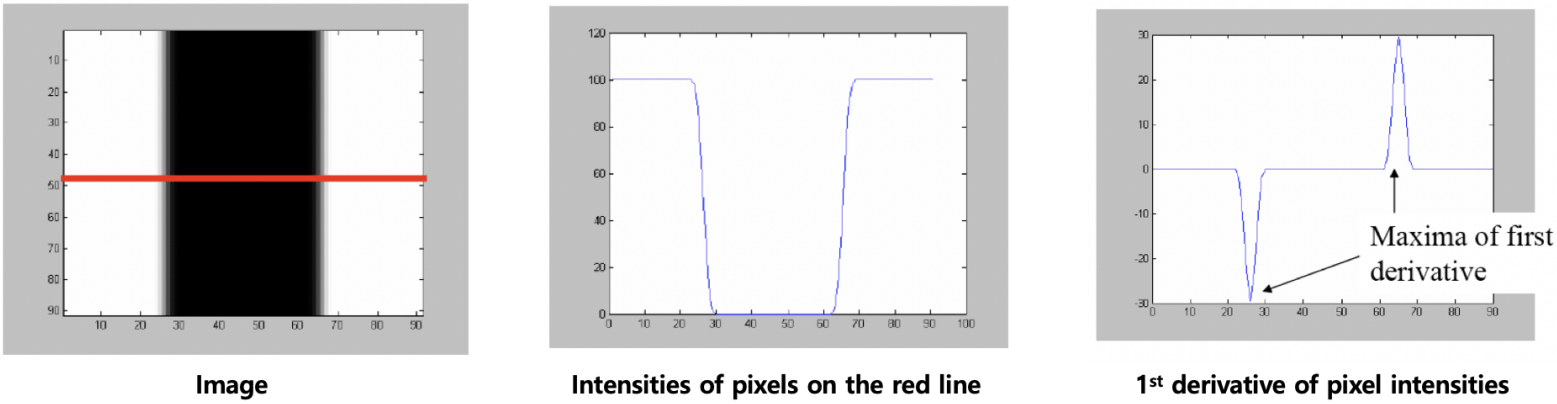


Figure 2. Graph of pixel intensity and 1st derivative given image , from [1]

Given the definition of edges, the idea of gradient comes into the consideration, because regions without any edge will return zero gradient, while the other regions output some positive or negative values. This is illustrated in the right panel of Fig 2. The 1st derivative map shows non-zero response only at pixels where the edges lay on. **Thus, we can conclude that, by finding pixels resulting the maxima of 1st derivative, edges in an image can be located.**

1.2 Design a filter to compute derivatives

In order to compute the derivative of an image, we can make use of the concept of convolution with filter (note that it is also can be computed by correlation). Let's recall how the partial derivative is calculated in **2D** function f that represents an image. In continuous setting, partial derivative of f with respect to x is defined as follows:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

Equation 1. Partial derivative of f with respect to x

However, in computer vision, we are dealing with images which are discrete data. Thus, we approximage it by using finite differences.

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$$

Equation 2. discretized Eq 1.

To implement Eq 2 as convolution, the associated filter will be

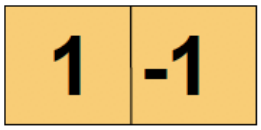


Figure 3. Convolutional filter to compute the image gradient, from [2]

Similarly, we can also design a filter to compute the partial derivative of f with respect to y. Fig 4 shows the filter response after convolution with respect to x (center) and y (right panel).

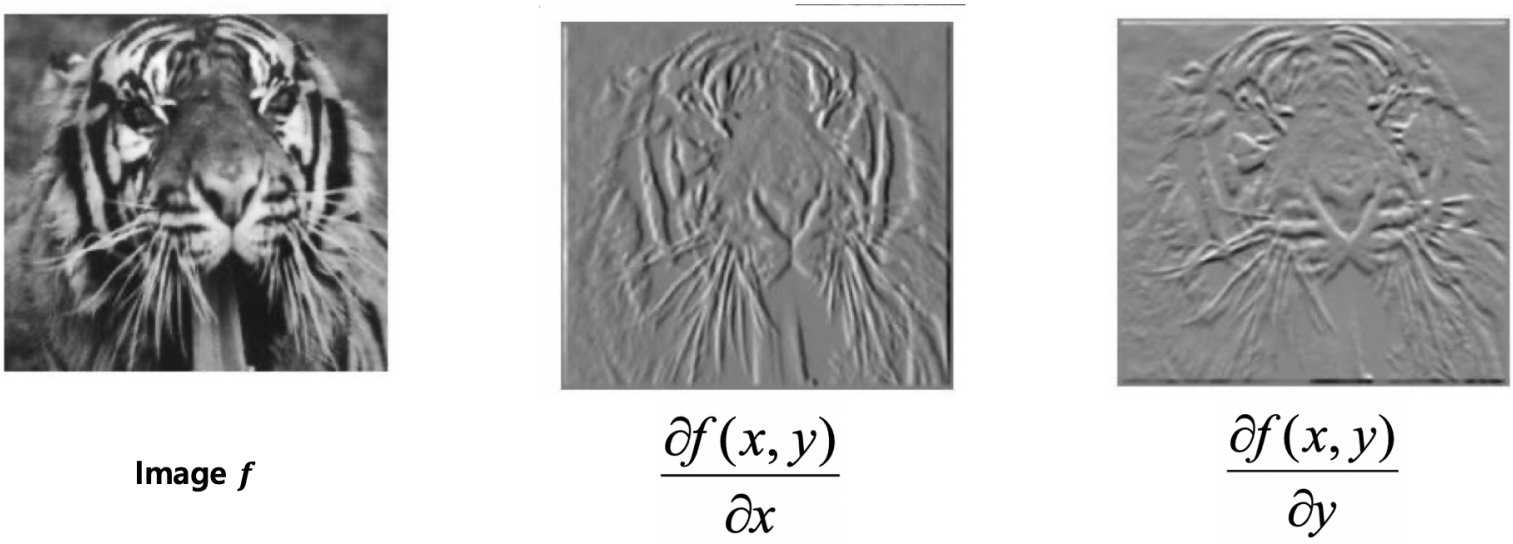


Figure 4. Filter response maps (center and right panel) after convolution with x-gradient and y-gradient filters, respectively, from [1], [2]

Before we sum up the filter design, we need to be aware that, in practice, we use 3×1 gradient filter for x derivative, instead of 2×1 gradient filter, and so does for y derivative (1×3 filter). Let me elaborate why and how to struct the 3×1 filter (for the derivatives with respect to x).

Despite the simplicity of filter: *[1 -1],* it has some issues.

- Firstly, once the image is convolved with this filter, it shifts the image by half a pixel.
- Secondly, which is also associated with first reason, when we apply *[1 -1]* to 2 pixels *(x, y)* and *(x+1, y)* it actually computes the gradient at position *(x+0.5, y)* not at position *(x, y)* nor *(x+1, y)*. In order to fix this, we insert 0 in-between the *[1 -1]* to make *[1 0 -1]*. By convolving the new filter *[1 0 -1]* with pixels *(x -1, y), (x, y)* and *(x+1, y)*, it returns the gradient with respect to ***x*** for the center pixel *(x, y)*.

In short,

- x-derivative filter: [1 -1] -> [1 0 -1]
- y-derivative filter: $[1\ -1]^{\mathbf{T}}$ -> $[1\ 0\ -1]^{\mathbf{T}}$

> *please refer to [8] for more detail explanation*

1.3 Noise removal

Images, in practice, contain noise inside as we've studied before. If we directly compute the derivatives of a noise-containing image, then it gives unexpected result where we cannot detect any information regarding edge locations. Fig 5 illustrates this.

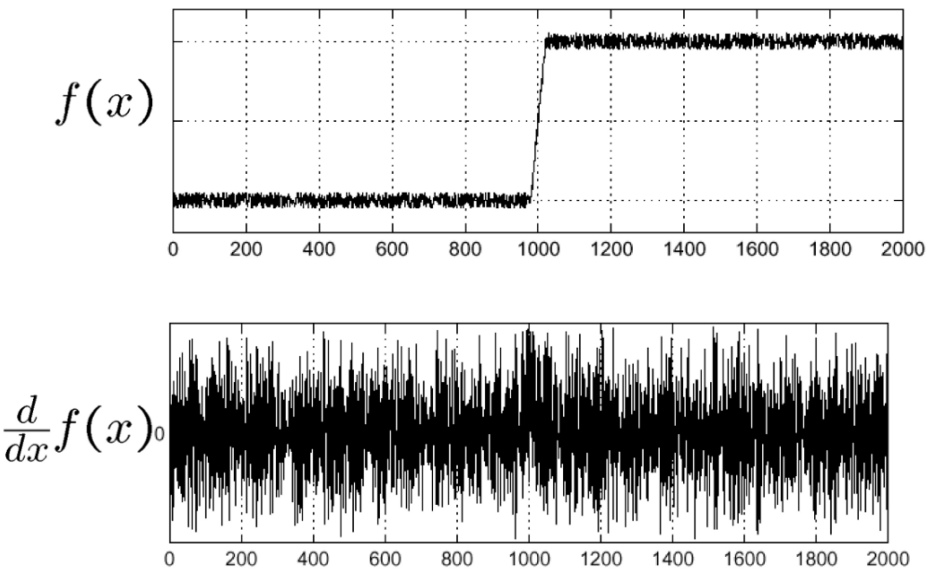


Figure 5. Computed derivatives of a noisy image, from [1], [3]

Consider a single row of an image (recall the red line in Fig 2) and plot the intensities ***f*** of pixels on the row as below. Because the noise make each pixel intensity fluctuate from others, when we compute their derivatives we will only get the zigzag patterns. **Therefore, in order to properly extract edge locations, the noise removal should be preceded before the computing image derivatives.**

And we already know how to exclude noise, the Gaussian Filtering (Smoothing).

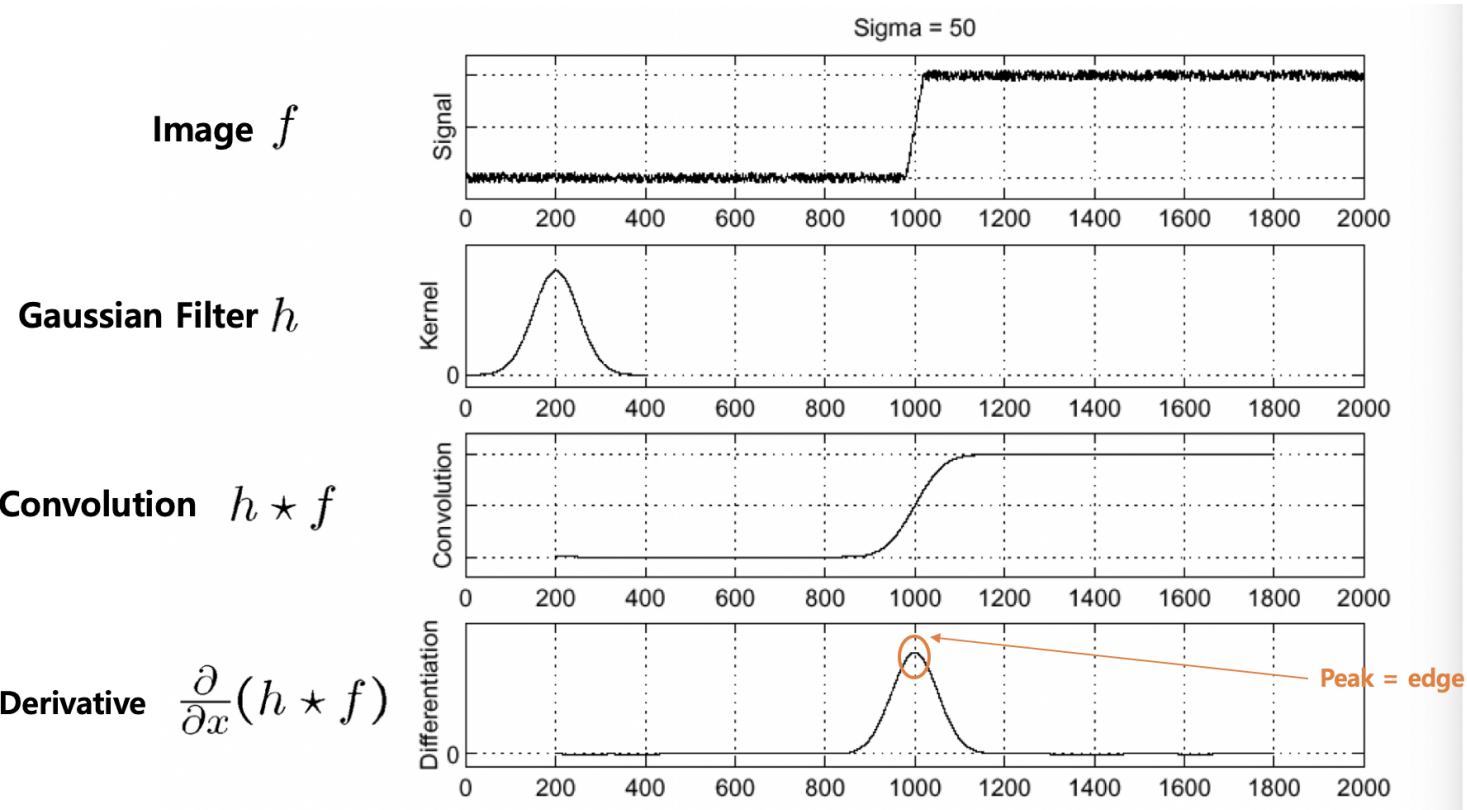


Figure 6. Proper approach to locate edges in a noisy image with Gaussian and Derivative Filters, from [1], [3]

First, convolve image with Gaussian filter with a certain sigma (standard deviation).

After this, we get an image from which noise is removed until certain extent.

Then by computing the derivative of the image and looking at its peak, we obtain the edge locations.

1.4 How to make this process efficiently? (Derivative of Gaussian)

Let f and h be an image and a filter, respectively. One characteristic of convolution is that the derivative of convolved image $(h \star f)$ is equivalent to convolving image f with the differentiated filter ∇h.

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Using this, we can simplify the steps in Fig 6. This time, we first compute the differentiated filter ∇h and convolve it with the image f.

This yields the same result as Fig 6.

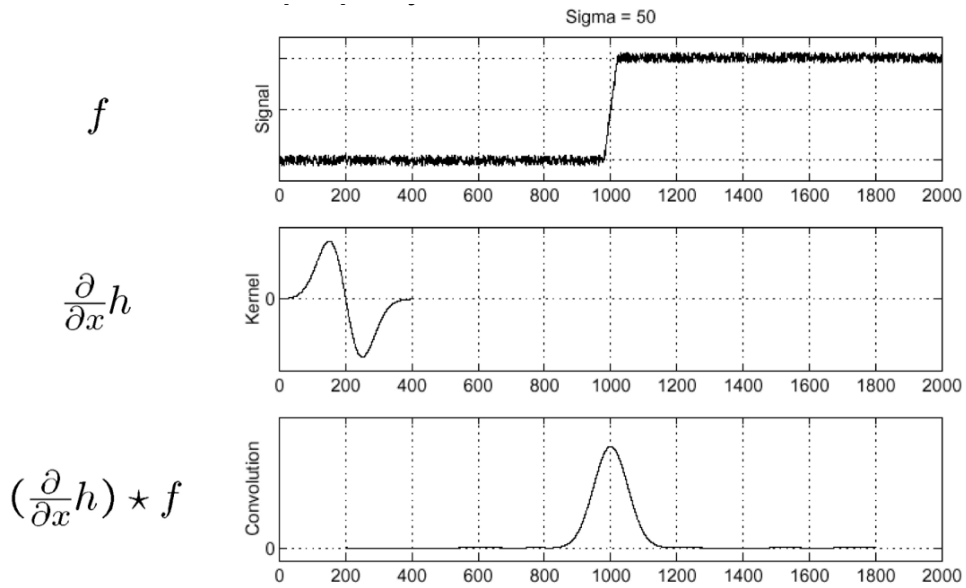
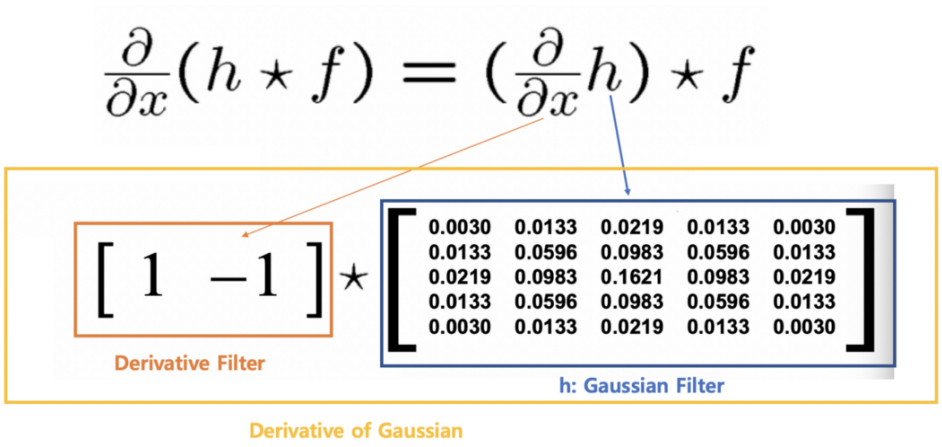


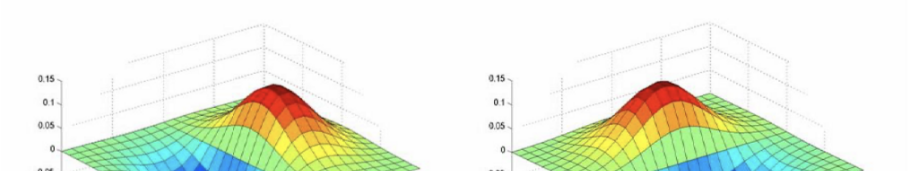
Figure 7. simplified approach to locate edges in image, from [1], [3]

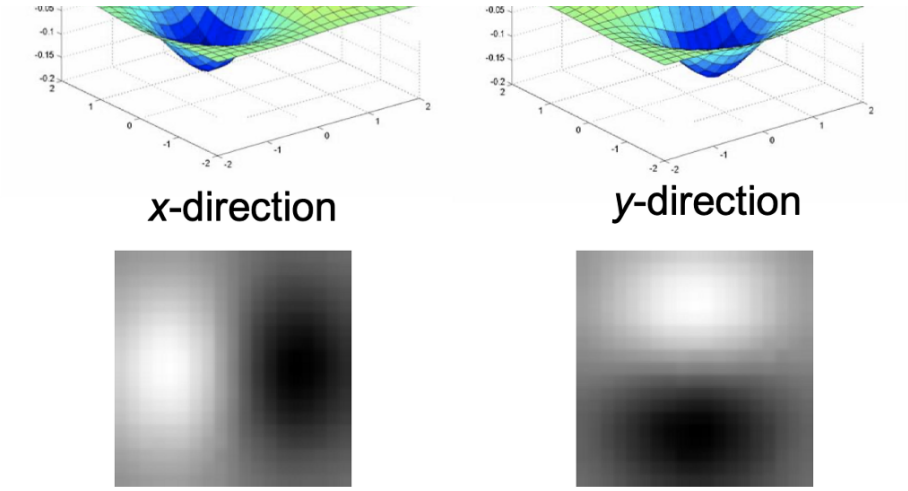
Lastly, recall that we can represent the derivative computation as a convolution with filter *[1 -1]* (or *[1 0 -1]* as mentioned above). Therefore, the term derivative of filter $\nabla \boldsymbol{h}$ becomes the convolution of *[1 -1]* and Gaussian Filter ***h,*** as illustrated below.



And the result of this convolution (Derivative filter $\star \boldsymbol{h}$) is called the **Derivative of Gaussian.**

In short, Gaussian Derivative filter contains the both Gaussian smoothing and Derivative computation in it. Therefore, as long as we convolve an image with **Derivative of Gaussian filter**, there is no need to perform Gaussian smoothing and derivative computation separately like Fig 6.





Example of Derivative of Gaussian Filter with respect to x and y direction

2. Laplacian Filter

A Laplacian filter is one of edge detectors used to compute the second spatial derivatives of an image. It measures the rate at which the first derivatives changes. In other words, Laplacian filter highlights the regions where the pixel intensities dramatically change. Due to this characteristic of Laplacian filter, it is often used to detect edges in an image. We will see how the filter find edges with visual illustration later.

The mathematical definition of Laplacian is as follows.

$$\begin{aligned} \textit{Laplacian } \nabla^2 f &= \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \\ &= f(x+1,y) + f(x-1,y) - 2f(x,y) + f(x,y+1) + f(x,y-1) - 2f(x,y) \\ &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \end{aligned}$$

Given the definition, discretized *3 x 3* Laplacian filter (because we are dealing with images and they are discrete) for an image ***f*** is defined as an array below:

0	−1	0
−1	4	−1
0	−1	0

Figure 8. 3 × 3 Laplacian filter

Note that, the defined kernel above, which is not identical to the mathematical definition of Laplacian due to the opossite signs, uses a

negative peak because it is more commonly used and straightforward. **However, it is still valid.**

2.1. Edge detection

Let's get back to the reason why we use the Laplacian filter. As mentioned before, Laplacian filter is one common method to detect edge, but how?

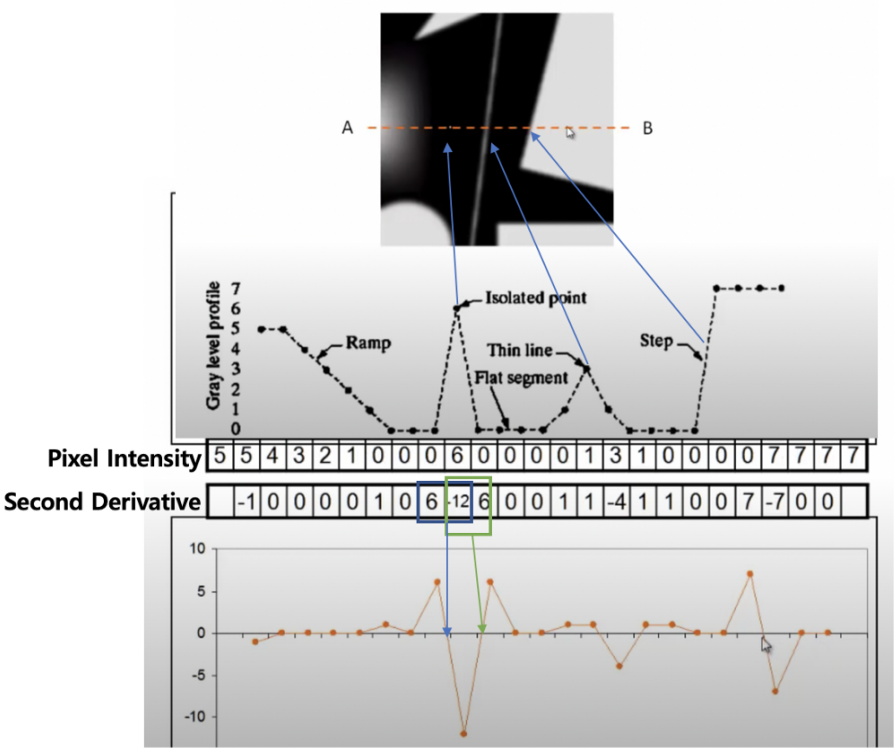


Figure 9. example of from here

First we compute the second derivative for each pixel. Our goal is to locate the edge locations(pixels), and previously, we've found them by looking at the Maxima of first derivatives. Recall how did we find the Maxima of first derivatives with second derivative. The answer is, edges are at between pixels whose second derivatives sharply transition from one sign to the other sign (zero-crossing). The two detected edges, among all edges, are marked with blue and green box in Fig 9.

2.2. Laplacian of Gaussian

Similar to 1.4 Derivative of Gaussian, the same idea to simplify the edge detection with Laplacian filter is applied. While all the other steps remain the same, the only difference from *Derivative of Gaussian Filter* is that Laplacian Filter replaces the Derivative Filter, meaning $\nabla \boldsymbol{h}$ in Fig 6 becomes $\nabla^2 \boldsymbol{h}$.

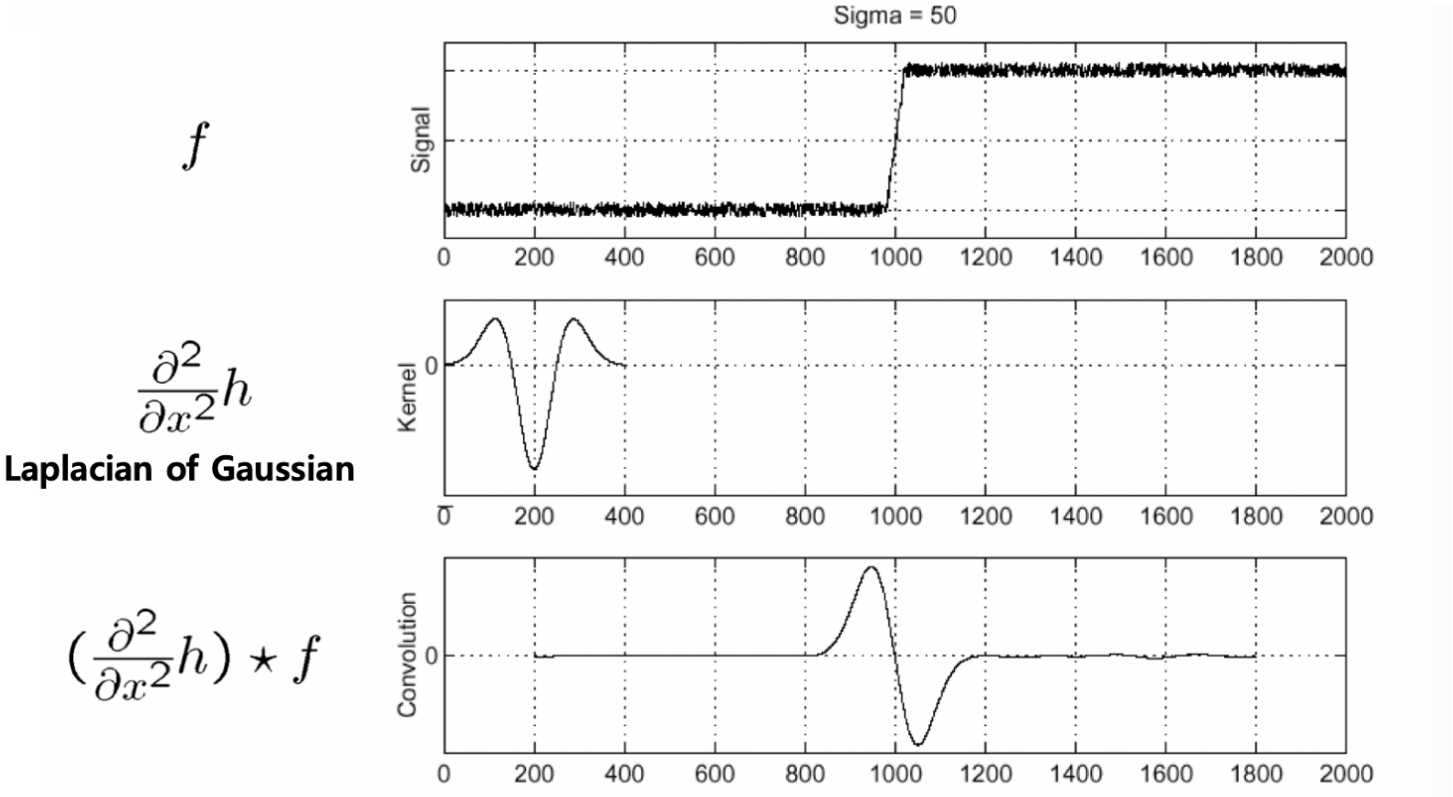


Figure 10. Proper approach to locate edges in a noisy image with Gaussian and Laplacian Filters, from [1], [3]

As Laplacian (∇^2) also can be represented by convolution with Laplacian filter, we convolve the Laplacian filter with Gaussian filter, and therefore, we obtain the **Laplacian of Gaussian (LoG)** filter. Similar to *Derivative of Gaussian*, the *LoG* contains Gaussian smoothing and second-derivative computation, there is no need to conduct them separately, as long as an image is convolved with *LoG*.

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

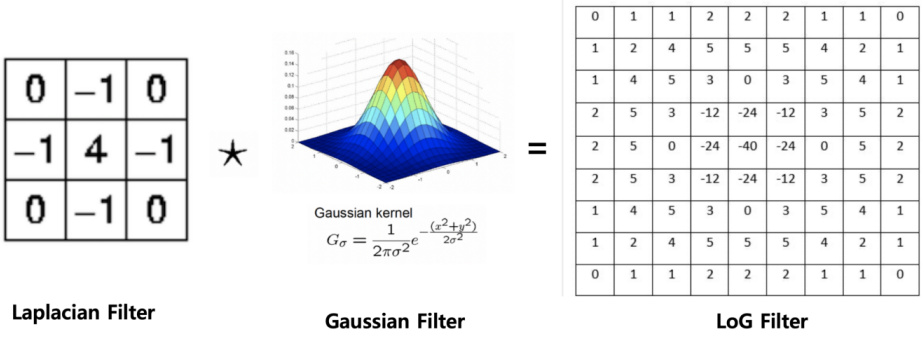


Figure 11. Illustration of building LoG Filter to facilitate understanding, from [1], [7]

> *Note that filter size and values in Fig 11 are arbitrary chosen, and therefore, the computed LoG is not accurate. Consider it as an example to just to see the*

> *general picture of how the LoG Filter is structured using convolution.*

3. Difference of Gaussians (DoG)

The difference of Gaussians and resulting image after convolution by it is illustrated below.

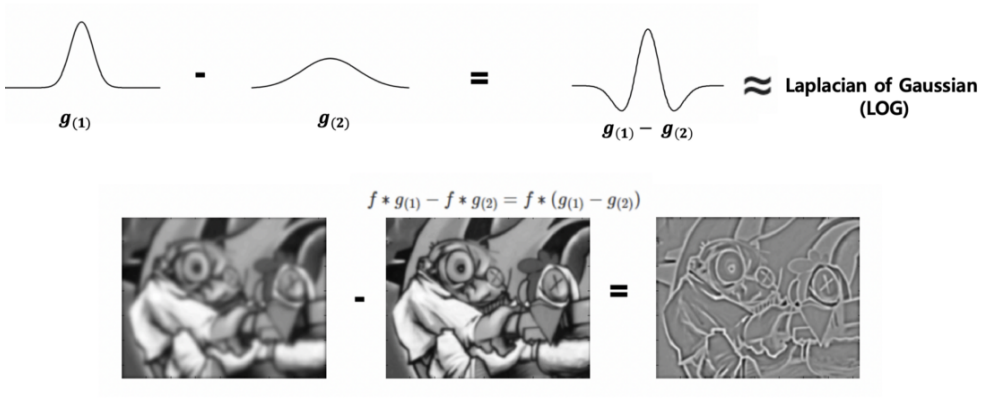


Figure 12. Laplacian ~ Difference of Gaussian, from [1]

The noticeable observation here is that Subtracting one Gaussian by another approximates the Laplacian of Gaussian. This indicates while approximating the LoG, there is no actual derivative computation needed. One additional reason why DoG is beneficial is that it is common in computer vision tasks that an image is filtered by Gaussian filters at many scales. While storing the Gaussian-filtering results of an image, we can easily make use of them to extract edges by computing the difference between subsequent filtering outputs, without actual computation of image derivatives.

4. Reference

[1] RWTH Aachen, computer vision group

[2] CS 376: Computer Vision of Prof. **Kristen Grauman**

[3] **S. Seitz**
[4] **Forsyth & Ponce**

[5] Prof. **Svetlana Lazebnik**

[6] Prof **M. Irani** and **R. Basri**

[7] Laplacian vs Laplacian of Gaussian

[8] Why [-1 0 1], not [-1 1]?

Any corrections, suggestions, and comments are welcome