Tutorial 4 - Convolution

Gidon Rosalki

2025-11-12

1 Convolution 1D and 2D

1.1 1D Convolution

A 1D Convolution has the following formula:

$$h(x) = (f * g)(x) = \sum_{i=0}^{N-1} f(i)\, g(x-i)$$

What we essentially do, is flip the direction of the second vector, and beginning with an overlap of 1, multiply element wise all the elements of the vectors, summing them together, and the result is the relevant element of the new vector. The easiest way to understand is to see a positioned example, and there is a great one in the provided presentation.

1.1.1 Properties

Convolution has various properties:

$$\begin{gathered}\text{Commutative: } f * g = g * f \\ \text{Associative: } f * (g * h) = (f * g) * h \\ \text{Distributive: } f * (g + h) = f * g + f * h\end{gathered}$$

In addition to this, convolution is a linear operator, meaning it can be represented as simply a matrix multiplication:

$$f * g = Gf$$

Consider

$$\begin{aligned} f &= (1\ 0\ 1) \\ g &= (0\ 2\ 1) \end{aligned}$$

Then

$$\begin{aligned} f * g &= Gf \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 2 \\ 1 \\ 2 \\ 1 \end{bmatrix} \\ &\implies (0\ 2\ 1\ 2\ 1) \end{aligned}$$

1.2 2D Convolution

We define a 2D convolution as follows:

$$f(x,y) * g(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} f(x-i, y-j)\, g(i,j)$$

Here f is the image, and g is the kernel / filter.

We can use these kernels for doing all manner of effects on images. Remember, kernels sum to 1. If we take $\frac{1}{9}$ times a matrix of all 1s, we will get a smoothing of the image. A more popular method for smoothing is the kernel

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Since it results in a nicer smoothing to our eyes. Of course, we're using 3×3 examples here, but kernels are normally much larger, since a 3×3 kernel does very little to a large image.

If we have a shifted kernel:

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

Then we will shift the image to the left. Do not forget that for a 2D kernel, you need to flip twice, since otherwise your convolution will shift in the wrong direction.

2 Image Derivatives

2.1 First derivative approximation

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

This is approximately the same as setting $\varepsilon = 1$, where 1 indicates 1 pixel (which is our smallest possible division). We can perform a derivative against the rows (ie, y direction) with a convolution with

$$\begin{pmatrix}1\\ -1\end{pmatrix}$$

and in against the columns (ie, the x direction) with a convolution with

$$\begin{pmatrix}1 & -1\end{pmatrix}$$

2.1.1 Gradient

The gradient is the vector of partial derivatives. It points in the direction of the most rapid change in intensity.

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

The gradient has magnitude:

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

and direction (orientation angle):

$$\alpha = atan2\left(\frac{\left(\frac{\partial f}{\partial x}\right)}{\left(\frac{\partial f}{\partial y}\right)}\right)$$

We also have the directional derivative, which gives us the change in angle. This is not massively used, but it is good to know that it exists.

$$\cos(\alpha)\frac{\partial f}{\partial x} + \sin(\alpha)\frac{\partial f}{\partial y}$$

Overall, what do these different direction derivatives do? A derivative with respect to the columns will show vertical edges in a picture, and with respect to the rows will show horizontal edges. Logically from here, if we take the magnitude of the gradients, we get the lines in both directions.

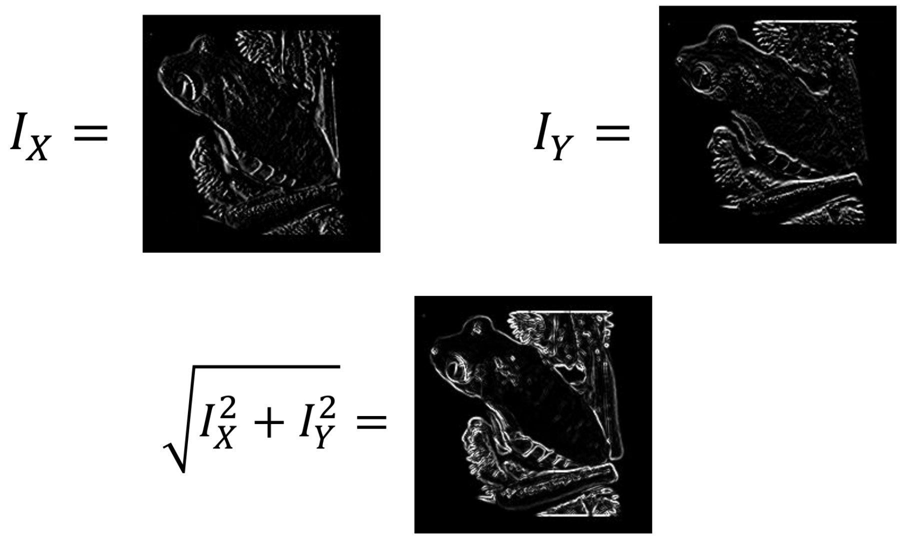

Figure 1: Edge directions example

2.2 Second derivative approximation

$$\frac{\partial^2}{\partial x^2} f(i,j) \approx f(i-1,j) + f(i+1,j) - 2f(i,j)$$

We can implement this by convolving with

$$(1 \ -2 \ 1)$$

We can check this by seeing that

$$[1 \ -1] * [-1 \ 1] = [1 \ -2 \ 1]$$

Remember, that in Fourier transforms, we move from the time domain, to the frequency domain. The DFT:

$$F(\omega) = \sum_{x=0}^{N-1} f(x) e^{-\frac{2\pi i x \omega}{N}}$$

and IDFT:

$$f(\omega) = \frac{1}{N} \sum_{\omega=0}^{N-1} F(\omega) e^{\frac{2\pi i x \omega}{N}}$$

The image derivative is the inverse FT of the weighted frequency domain. High frequencies affect the image derivative more than low frequencies. This causes problems, since noise has more high frequency than normal image.

$$\frac{\partial f(x,y)}{\partial x} = \frac{2\pi i}{N} \cdot \Phi^{-1}\left(u \cdot \Phi\left(f(x,y)\right)\right)$$

The problem with noise is that it can affect the second derivative so much, that the second derivative just looks like random noise. We would like to reduce the effects of noise on the derivative, but symmetric smoothing may eliminate the derivative response altogether. To resolve this, we use Sobel kernels, where we smooth the image in one direction, and compute the derivative in the orthogonal direction. The Sobel kernels are as follows:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\frac{\partial f}{\partial y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Note that if we convolve a smoothing kernel with a derivative kernel:

$$f * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * [1 \ 0 \ -1] = f * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

3 Fourier Transform

Theorem 1 (Convolution theorem)**.**

$$\Phi\left(f * g\right) = F \cdot G\Phi\left(f \cdot g\right) \qquad\qquad = F * G$$

So, to convolve by Fourier:

$$f * g = \Phi^{-1}\left(F \cdot G\right)$$

This has the complexity of $O\left(n \log\left(n\right)\right)$, thanks to the FFT, where n is the number of pixels in the image. So we see, different FT phenomena can be explained by convolution, and vice versa. We use this Fourier interpretation in order to design convolution filters.

4 Sharpening

We have used various methods to enhance images. From the first tutorial, we used histogram equalisation. We also learnt in the lecture about smoothing, and median filtering (the salt and pepper thing). We will also discuss sharpening.

4.1 Laplacian subtraction

The laplacian:

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

Or in matrix form:

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

We can then subtract this from the image, and sharpen the image, reducing its smoothing. It should however be noted, that while this does enhance the details, it also enhances the noise, which is less than ideal.

5 Edge detection

We can use the gradient for edge detection. If we take the gradient of an image, then there are a number of pixels between the maximum, and 0. As a result we have a lot of lit up pixels that are not the edge. What we can do is set some sort of boundary, and convert the gradient into a binary image, where pixels with values above the binary are set to 1, and others are set to 0.

Edges generally involve a diagonal line, crossing from one level, to another. As a result, the first derivative will be a square wave over this transition, and the second derivative will be two hard peaks at the change points of the square wave.

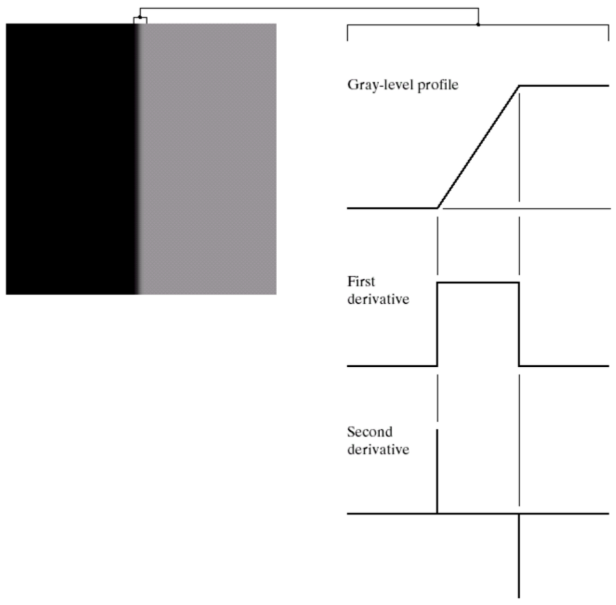

Figure 2: Derivative response at edges