Convolution

Carlo Tomasi

To introduce the concept of convolution, suppose that we want to determine where in the image there are vertical edges. Since an edge is an abrupt change of image intensity, we might start by computing the derivatives of an image in the horizontal direction. Derivatives with a large magnitude, either positive or negative, are elements of vertical edges. The partial derivative of a continuous function $F(x,y)$ with respect to the "horizontal" variable x is defined as the local slope of the plot of the function along the x direction or, formally, by the following limit:

$$\frac{\partial F(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{F(x+\Delta x, y) - F(x,y)}{\Delta x} \, .$$

An image from a digitizer is a function of a discrete variable, so we cannot take Δx arbitrarily small: the smallest we can go is one pixel. If our unit of measure is the pixel, we have

$$\Delta x = 1$$

and a rather crude approximation to the derivative at an integer position $j = x$, $i = y$ is therefore[1]

$$\left. \frac{\partial F(x,y)}{\partial x} \right|_{x=j, y=i} \approx f(i, j+1) - f(i,j) \, .$$

Here we assume for simplicity that the origins an axis orientations of the x, y reference system and the i, j system coincide. When we do edge detection, we will see that we can do much better than this as an approximation to the derivative, but this example is good enough for introducing convolution.

Here is a piece of code that computes this approximation along row i in the image:

```
for (j = jstart; j <= jend; j++) h[i][j]= f[i][j+1] - f[i][j];
```

Notice, in passing, that the last value of j for which this computation is defined is the next-to-last pixel in the row, so `jend` must be defined appropriately. This operation amounts to taking a little two-cell mask g with the values $g[0] = 1$ and $g[1] = -1$ in its two entries, placing the mask in turn at every position j along row i, multiplying what is under the mask by the mask entries, and adding the result. In C, we have

[1]Notice that to conform with usual notation the order of variables i, j in the discrete array is switched with respect to that of the corresponding variables x, y in the continuous function: x, y are right and up, respectively, while i, j are down and right. Other conventions are possible, of course.

```
for (j = jstart; j <= jend; j++) h[i][j]= g[0]*f[i][j+1] + g[1]*f[i][j];
```

This adds a little generality, because we can change the values of g without changing the code. Since we are generalizing, we might as well allow for several entries in g. For instance, we might in the future switch to a centered approximation to the derivative,

$$\left.\frac{\partial F(x,y)}{\partial x}\right|_{x=j,y=i} \approx f(i,j+1) - f(i,j-1) \; .$$

So now we can define for instance $g[-1] = 1$, $g[0] = 0$, and $g[1] = -1$ and write a general-purpose loop in view of possible future changes in our choice of g:

```
for (j = jstart; j <= jend; j++)
{
   h[i][j] = 0;
   for (b = bstart; b <= bend; b++)
      h[i][j] += g[b]*f[i][j-b];
}
```

This is now much more general: it lets us choose which horizontal neighbors to combine and with what weights. But clearly we will soon want to also combine pixels above i, j, not only on its sides, and for the whole picture, not just one row. This is easily done:

```
for (i = istart; i <= iend; i++)
for (j = jstart; j <= jend; j++)
{
   h[i][j] = 0;
   for (a = astart; a <= aend; a++)
   for (b = bstart; b <= bend; b++)
      h[i][j] += g[a][b]*f[i-a][j-b];
}
```

where now `g[a][b]` is a two-dimensional array. The part within the braces is a very important operation in signal processing. The two innermost `for` loops just keep adding values to `h[i][j]`,so we can express that piece of code by the following mathematical expression:

$$h(i,j) = \sum_{a=a_{start}}^{a_{end}} \sum_{b=b_{start}}^{b_{end}} g(a,b) f(i-a, j-b) \; . \tag{1}$$

This is called a *convolution*. Convolving a signal with a given mask g is also called *filtering* that signal with that mask. When referred to image filtering, the mask is also called the *point-spread function* of the filter. In fact, if we let

$$f(i,j) = \delta(i,j) = \begin{cases} 1 & \text{if } i = j = 0 \\ 0 & \text{otherwise} \end{cases} \quad , \tag{2}$$

then the image f is a single point (the 1) in a sea of zeros. When the convolution (1) is computed, we obtain

$$h(i, j) = g(i, j) \ .$$

In words, the single point at the origin is spread into a blob equal to the mask (interpreted as an image).

The choice of subscripts for the entries of g, in both the code and the mathematical expression, seems arbitrary at first. In fact, instead of defining $g[-1] = 1$, $g[0] = 0$, $g[1] = -1$, we could have written, perhaps more naturally, $g[-1] = -1$, $g[0] = 0$, $g[1] = 1$, and in the expressions `f[i-a][j-b]` and $f(i-a, j-b)$ the minus signs would be replaced by plus signs. In terms of programming, there is no difference between these two options (and others as well). Mathematically, on the other hand, the minus sign is much preferable. The first reason is that $g(i, j)$ can be interpreted, as done above, as a point spread function. With the other choice of signs the convolution of $f = \delta$ with g would yield a doubly-mirrored image $g(-i, -j)$ of the mask g.

Another reason for this choice of signs is that the convolution now looks like the familiar multiplication for polynomials. In fact, consider two polynomials

$$\begin{aligned} f(z) &= f_0 + f_1 z + \ldots + f_m z^m \\ g(z) &= g_0 + g_1 z + \ldots + g_n z^n \ . \end{aligned}$$

Then, the sequence of coefficients of the product

$$h(z) = h_0 + h_1 z + \ldots + h_{m+n} z^{m+n}$$

of these polynomials is the (one-variable) convolution of the sequences of their coefficients:

$$h_i = \sum_{a=a_{start}}^{a_{end}} g(a) f(i-a) \ . \tag{3}$$

In fact, notice that $g(a)$ multiplies z^a and $f(i-a)$ multiplies z^{i-a}, so the power corresponding to $g(a)f(i-a)$ is z^i for all values of a, and h_i as defined by equation (3) is the sum of all the products with a term z^i, as required by the definition of product between two polynomials. Verify this with an example. Thus, putting a minus sign in the definition (1) of the convolution makes the latter coincide with the product of two polynomials, thereby making the convolution an even deeper are more pervasive concept in mathematics.

The interpretation of the convolution mask $g(i, j)$ as a point-spread function suggests another useful way to look at the operation of filtering. The δ function defined in (2) is a single spike of unit height at the origin. A generic image $f(i, j)$, on the other hand, can be seen as a whole collection of spikes, one per pixel, whose height equals the image value. In formulas,

$$f(i, j) = \sum_a \sum_b f(a, b) \delta(i-a, j-b) \ ,$$

where the summations range over the entire image. This expression is the convolution of f and δ. Notice that this is the same as

$$f(i,j) = \sum_a \sum_b f(i-a, j-b)\delta(a,b)$$

after the change of variables $i \to i-a$, $j \to j-b$ at least if the summation ranges are assumed to be $(-\infty, +\infty)^2$. But if the output to $\delta(i,j)$ is the point-spread function $g(i,j)$, then the output to $\sum_a \sum_b f(a,b)\delta(i-a, j-b)$ is a linear combination of point-spread functions, amplified each by one of the pixels in the image. This describes, for instance, what happens in a pinhole camera with a pinhole of nonzero radius. In fact, one point in the world spreads into a small disk on the image plane (the point-spread function, literally). Each point in the world draws a little disk onto the image, and the brightness of each disk is proportional to the brightness of the point in the world. This results in a blurred image. In conclusion, the image formed by a pinhole camera is the convolution of the ideal (sharp) image with a pillow-case function.

The difference between the convolution defined in (1) and what happens in the pinhole camera is that the points in the world are not neatly arranged onto a rectangular grid, as are pixels in an image, but form a continuous. Fortunately, all the concepts relative to convolution can be extended to continuous functions as well. In analogy with equation (1), we define the convolution between two continuous functions $f(x,y)$ and $g(x,y)$ as the following double integral:

$$h(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} g(a,b) f(x-a, y-b)\, da\, db \; .$$

The blurred image produced by the pinhole camera is then the convolution of the ideally sharp image $f(x,y)$ with the pillow-case function

$$g(x,y) = \begin{cases} 1 & \text{if } \sqrt{x^2+y^2} \leq r \\ 0 & \text{otherwise} \end{cases} \quad ,$$

where r is the radius of the pinhole.

Convolution and filtering are key concepts in image processing, and we will encounter them throughout this class.

[2] Otherwise, they should be modified according to the change of variables.