## Convolution

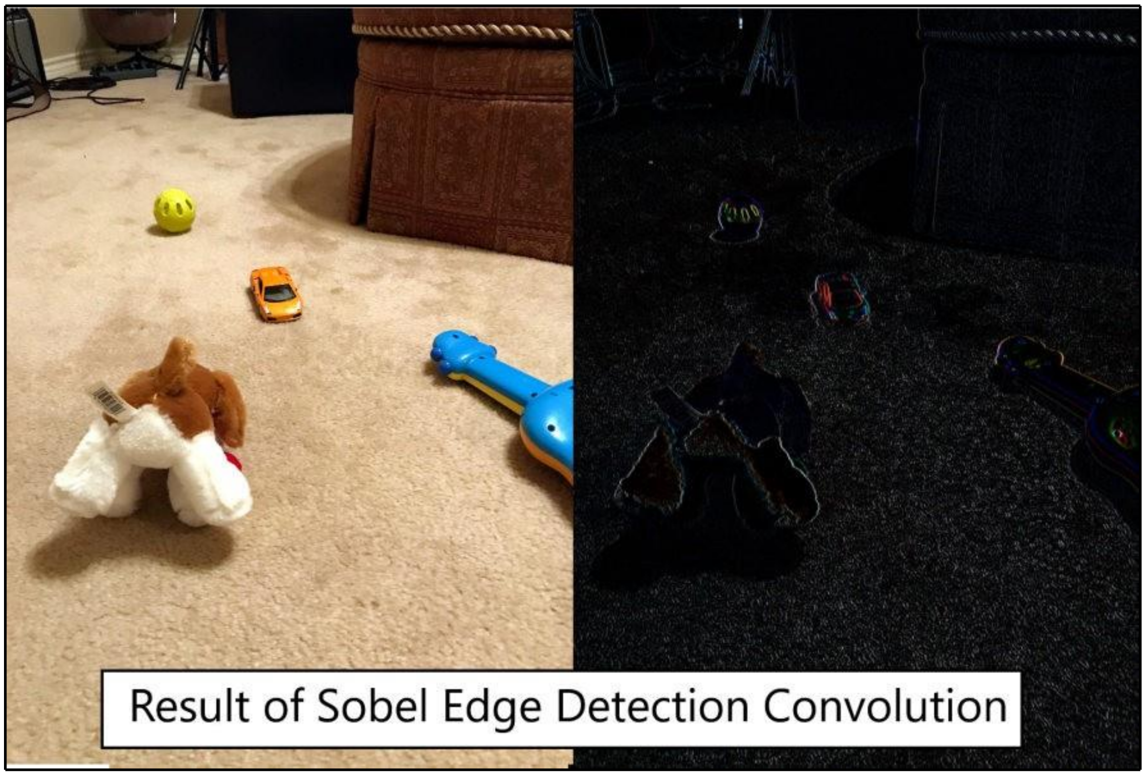
Every once in a while, you come across some mathematical construction that turns a complex task into just a bunch of adding, subtracting, multiplying, and dividing. Vectors in geometry work like that, and, in image processing, we have the convolution kernel. It transpires that most of the common image processing techniques – edge detection, corner detection, blurring, sharpening, enhancing, and so on, can be accomplished with a simple array construct.

It is pretty easy to understand that in an image, the neighbors of a pixel are just as important to what a pixel is as the pixel itself. If you were going to try and find all the edge pixels of a box, you would look for a pixel that has one type of color on one side, and another type on the other. We need a function to find edges by comparing pixels on one side of a pixel to the other.

The convolution kernel is a matrix function that applies weights to the pixel neighbors – or pixels around the one pixel we are analyzing. The function is usually written like this, as a 3 x 3 matrix:

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

This is the **sobel edge detector** in the $Y$ direction. This detects edges going up and down. Each block represents a pixel. The pixel being processed is in the center. The neighbors of the pixels on each side are the other blocks – top, bottom, left, and right. To compute the convolution, you apply the weight to the value of each pixel by multiplying the value (intensity) of that pixel, and then adding all of the results. If this image is in color – RGB – then we compute the convolution for each color separately and then combine the result. Here is an example of a convolution being applied to a pixel:



Add the final result to a sample image. Note that we only get the edge as the result – if the colors are the same on either side of the center pixel, they cancel each other out and we get zero, or black. If they are different, we get 255, or white, as the answer.

If we need a more complex result, we may also use a 5 x 5 convolution, which takes into account the two nearest pixels on each side, instead of just one.

The good news is that you don't have to choose the convolution to apply to the input images – the Keras AI frontend will set up all of the convolutions, and you only have to set the size as either 3 x 3 or 5 x 5. The neural network will determine which convolutions provide the most data and support the training output we want.

But wait, I hear you say. What if the pixel is on the edge of the image and we don't have neighbor pixels on one side? In that event, we have to add padding to the image – which is a border of extra pixels that permit us to consider the edge pixels as well.