2D Convolution Computation

Reference for the theoretical background: Chapter 6, Robot Vision, pp. 104 – 111, by BKP Horn, MIT Press

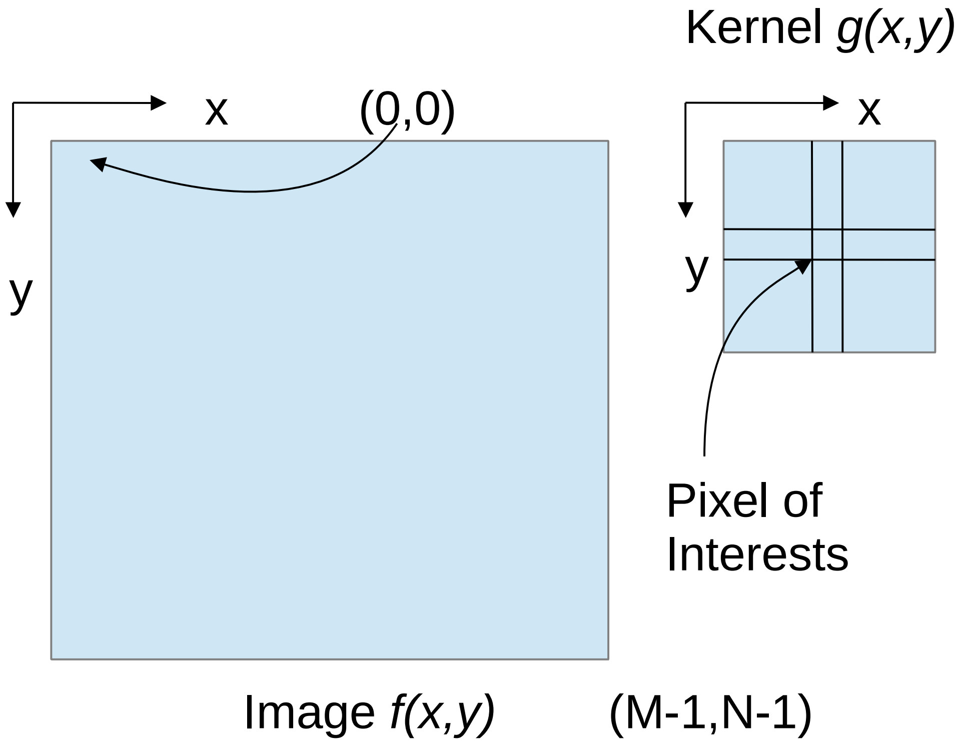
Definition:

$$f(x) * g(x) \;=\; \int_{-\infty}^{\infty} f(\tau)\cdot g(x-\tau)\,d\tau$$

$$c(n_1,n_2) = \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} \underbrace{a(k_1,k_2)}_{\text{Image}}\underbrace{b(n_1-k_1,n_2-k_2)}_{\text{Kernel}}$$

Summation lower and upper bound in the case of M-by-N image f(x,y), should be adjusted to k1 = 0 to M-1, k2 = 0 to N-1

Reference for the OpenCV implementation: Learning OpenCV, Chapter 6, pp. 144 – 164.



Note: (1) 3 primitive computations: shift, multiplication, and addition;
(2) use discrete 2D convolution formula to compute 5x5 sample image with 3x3 kernels

Harry Li, Ph.D.

2D Convolution with Matlab/Octave

C = conv2(A,B) computes the two-dimensional convolution of matrices A and B.

The size of C is determined as follows: if [ma,na] = size(A), [mb,nb] = size(B), Then [mc,nc] = size(C), Where mc = max([ma+mb-1,ma,mb]) and nc = max([na+nb-1,na,nb]).

Octave
on Linux

```
>> A = [ 0 0 100 100 100
          0 0 100 100 100
          0 0 100 100 100
          0 0 100 100 100
          0 0 100 100 100 ]


 >> B = [ 1 0 -1
           1 0 -1
           1 0 -1 ]
```

```
C = conv2(A,B)
C =

    0    0   100   100     0  -100  -100
    0    0   200   200     0  -200  -200
    0    0   300   300     0  -300  -300
    0    0   300   300     0  -300  -300
    0    0   300   300     0  -300  -300
    0    0   200   200     0  -200  -200
    0    0   100   100     0  -100  -100
```

Information about Octave is also available on the WWW at http://www.octave.org and via the help@octave.org mailing list.

Matlab/Octave Gaussian Convolution

Let's consider Gaussian kernel computation first, use the following function

h = fspecial('gaussian', hsize, sigma)

the 'fspecial' function belongs to the image package from Octave Forge, if you have installed but not loaded, run 'pkg load image' from the Octave prompt.

```
>> pkg load image

>> sigma = 1.0

>> hsize = 5

>> h = fspecial('gaussian', hsize, sigma)
h =

   0.0029690   0.0133062   0.0219382   0.0133062   0.0029690
   0.0133062   0.0596343   0.0983203   0.0596343   0.0133062
   0.0219382   0.0983203   0.1621028   0.0983203   0.0219382
   0.0133062   0.0596343   0.0983203   0.0596343   0.0133062
   0.0029690   0.0133062   0.0219382   0.0133062   0.0029690
```

Matlab/Octave LoG Computation

Let's consider Gaussian kernel computation first, use the following function

h = fspecial('gaussian', hsize, sigma)

the 'fspecial' function belongs to the image package from Octave Forge, if you have installed but not loaded, run 'pkg load image' from the Octave prompt.

```
>> pkg load image

>> sigma = 1.0

>> hsize = 5

>> h = fspecial('log', hsize, sigma)
h =

   0.002835   0.006353   0.006983   0.006353   0.002835
   0.006353   0.000000  -0.015648   0.000000   0.006353
   0.006983  -0.015648  -0.051599  -0.015648   0.006983
   0.006353   0.000000  -0.015648   0.000000   0.006353
   0.002835   0.006353   0.006983   0.006353   0.002835
```

Guideline for 2D Convolution by OpenCV

first need to define a Mat object that holds the mask

```
Mat kern = (Mat_<char>(3,3) <<  0, -1,  0,
                               -1,  5, -1,
                                0, -1,  0);
```

Then call the filter2D function specifying the input, the output image and the kernel to use

:

```
filter2D(I, K, I.depth(), kern);
```

```
filter2D(src, dst, ddepth , kernel, anchor, delta, BORDER_DEFAULT );
imshow( window_name, dst );
```

The 5th optional argument specifies the center of the kernel, and the 6th one for determining what to do in the regions where the operation is undefined (borders). Using this function has the advantage that it’s shorter, usually faster than the hand-coded method. Check to see if this method takes 13 milliseconds (depends on the image and kernel size) while hand coded approach may take around 31 milliseconds.

http://docs.opencv.org/2.4/doc/tutorials/core/mat-mask-operations/mat-mask-operations.html

Sample 2D Convolution OpenCV (1)

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>
using namespace cv;
/** @function main */
int main ( int argc, char** argv )
{
  /// Declare variables
  Mat src, dst;
  Mat kernel;
  Point anchor;
  double delta;
  int ddepth;
  int kernel_size;
  char* window_name = "filter2D Demo";
  int c;
  /// Load an image
  src = imread( argv[1] );
  if( !src.data )
  { return -1; }
  /// Create window
  namedWindow( window_name, CV_WINDOW_AUTOSIZE );
```

http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/filter_2d/filter_2d.html

Sample 2D Convolution OpenCV (2)

```
/// Initialize arguments for the filter
  anchor = Point( -1, -1 );
  delta = 0;
  ddepth = -1;
  /// Loop - filter image w/different kernel sizes each 0.5 seconds
  int ind = 0;
  while( true )
    {
      c = waitKey(500);
      /// Press 'ESC' to exit the program
      if( (char)c == 27 )
        { break; }
      /// Update kernel size for a normalized box filter
      kernel_size = 3 + 2*( ind%5 );
      kernel = Mat::ones( kernel_size, kernel_size, CV_32F )/
(float)(kernel_size*kernel_size);
      filter2D(src, dst, ddepth , kernel, anchor, delta,
BORDER_DEFAULT );
      imshow( window_name, dst );
      ind++;
    }
  return 0;
}
```

Kernel Definition

2D Convolution

http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/filter_2d/filter_2d.html