

Programming Techniques

Tutorials and Mini Projects of C, C++, PHP, OpenGL, and other languages with C/C++ codes of Data Structure, Numerical Met and Computer Graphics

[Home](#)
[C Projects](#)
[C++ Projects](#)
[Computer Graphics](#)
[Data Structure](#)
[GLUT Tutorial](#)
[Numerical Methods](#)
[Image Processing](#)

Saturday, February 2, 2013

Calculating a convolution of an Image with C++: Image Processing

In convolution, the calculation performed at a pixel is a weighted sum of grey levels from a neighbourhood surrounding a pixel. Grey levels taken from the neighbourhood are weighted by coefficients that come from a matrix or convolution kernel. The kernel's dimensions define the size of the neighbourhood in which calculation take place. The most common dimension is 3x3. I am using this size of matrix in this article. During convolution, we take each kernel coefficient in turn and multiply it by a value from the neighbourhood of the image lying under the kernel. We apply the kernel to the image in such a way that the value at the top-left corner of the kernel is multiplied by the value at bottom-right corner of the neighbourhood. This can be expressed by following mathematical expression for kernel of size $m \times n$.

$$g(x, y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j, k) f(x - j, y - k)$$

Where h denotes the convolution kernel, f is the input pixel, g is the output pixel. Here $n_2 = n/2$ and $m_2 = m/2$. For 3x3 dimension of kernel, the above expression reduces to

$$g(x, y) = \sum_{k=-1}^1 \sum_{j=-1}^1 h(j, k) f(x - j, y - k)$$

Lets see an example. Suppose we have convolution matrix

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

and part of the image pixel is

	72	53	60	
	76	56	65	
	88	78	82	

Now applying above expression, the value of a central pixel becomes

$$g(x, y) = -1 \times 82 + 0 \times 78 + 1 \times 88 + -1 \times 65 + 0 \times 56 + 1 \times 76 + -1 \times 60 + 0 \times 53 + 1 \times 72 = 29$$

Algorithm for single pixel convolution can be written as (for 3x3)

Create a Kernel of size 3x3 and fill the kernel with coefficients
sum = 0

G+1 59

Topics

- [Advanced C Tutorial](#) (18)
- [Android Application Development](#)
- [Artificial Intelligence](#) (4)
- [Assembly Tutorials](#) (3)
- [Business Intelligence](#) (1)
- [C Projects](#) (7)
- [C Tutorial](#) (40)
- [Computer Graphics](#) (13)
- [cplusplus projects](#) (6)
- [Cplusplus Tutorial](#) (34)
- [Data Analysis](#) (1)
- [Data Structure and Algorithm](#) (26)
- [Database](#) (1)
- [GLUT Tutorial](#) (16)
- [Graphics Libraries](#) (4)
- [Image Processing](#) (12)
- [Javascript Tutorial](#) (3)
- [Linux](#) (3)
- [Machine Learning](#) (3)
- [Numerical Methods](#) (20)
- [PHP Tutorial](#) (13)
- [PROLOG](#) (8)
- [R](#) (1)

Find us on Facebook



Programming Techn
3,400 likes

Like Page

C++ Projects

Computer Gr

Be the first of your friends to like this



Popular Posts

[Top Collections\(Lists\) of Mini Pro and C++ free download source code exe.](#)

Here are the collections of Mini Pr and c++ with full source code and executable file. All the codes are cc using GCC Com...



[Breadth First Search Algorithm and Source](#)

Basic Theory Breadth searches are performed exploring all nodes at a depth before proceeding to the next. This means...

```

for k = -1 to 1 do
    for j = -1 to 1 do
        sum = sum + h(j + 1, k + 1) * f(x - j, y - k)
    end for
end for
g(x, y) = sum

```

The above discussion is mainly focused on single pixel operation. But the image doesn't have only single pixel. To do the convolution operation on whole image, we must perform convolution on all pixels as follows

```

for y = 0 to image_height do
    for x = 0 to image_width do
        perform single pixel convolution
    end for
end for

```

But this algorithm has one limitation that it is impossible to calculate the convolution at borders. When we try to calculate convolution of pixel lying in image border, part of the kernel lies outside the image. To handle this limitation there are various approaches, some are given below

No processing at border

This is simple technique in which pixels at border are simply neglected. This can be done using following algorithm

```

for all pixel coordinates x and y, do
    g(x, y) = 0
end for
for y = 1 to image_height - 1
    for x = 1 to image_width - 1
        perform single pixel convolution
    end for
end for

```

Reflected Indexing

In this method, the pixel lying outside the image i.e. $(x - j, y - k)$ are reflected back into the image by using following algorithm

```

if x < 0 then
    x = -x - 1
else if x >= image_width then
    x = 2 * image_width - x - 1
end if

```

Circular Indexing

In this method, coordinates that exceed the bounds of the image wrap around to the opposite side using following algorithm

```

if x < 0 then
    x = x + image_width
else if x >= image_width then
    x = x - image_width
end if

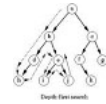
```

C++ implementation : Source Code

```

1 | #include<iostream>
2 | #include<opencv2/imgproc/imgproc.hpp>
3 | #include<opencv2/highgui/highgui.hpp>
4 |

```



Depth First Search in Algorithm and Source
Basic Theory Depth - searches are performed diving downward into quickly as possible. It does this by generatin...



Mini project Employ system using C
The employee record is very simple and for beginner mini project based on the menu-driven program elementary datab...

Difference between Arrays and Structures in C

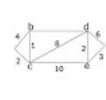
Both the arrays and structures are as structured data types as they provide a mechanism that enable us to access and manipulate...



Sum of two matrices dimensional array in C++
Matrix is the perfect of two dimensional array has row and column. represents one dimension and column represents second dimension...

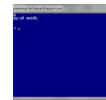


Mini Project Student System in C
Mini project student system is another program based on programming language C. It also uses files as data. This project is sim...

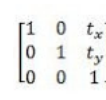


Implementation of Dijkstra's Shortest Path Algorithm in C++

Dijkstra's Shortest Path Algorithm is popular algorithm for finding shortest path between different nodes. The algorithm (Pseudocode) is a...



Mini project "Calend Application" Using C
Free download. Mini project "Calend Application" is also a project made using C. It uses many properties to make it colorful, for



Transformation (Translation, Rotation and Scaling) of 2D objects
1. Translation A transformation applied to an object is repositioning it along a straight-line from one coordinate location to another.

Blog Archive

- 2016 (2)
- 2015 (1)
- 2014 (2)
- ▼ 2013 (27)
 - December (4)
 - November (2)
 - July (2)
 - June (1)
 - May (3)
 - March (4)
 - ▼ February (5)

Loading large data into Oracle using SQL*Loader...

C/C++ program to determine the area of a week

Gaussian Filter generation using C++

```

5  using namespace std;
6  using namespace cv;
7
8  int reflect(int M, int x)
9  {
10     if(x < 0)
11     {
12         return -x - 1;
13     }
14     if(x >= M)
15     {
16         return 2*M - x - 1;
17     }
18     return x;
19 }
20
21 int circular(int M, int x)
22 {
23     if (x<0)
24         return x+M;
25     if(x >= M)
26         return x-M;
27     return x;
28 }
29
30
31 void noBorderProcessing(Mat src, Mat dst, float Kernel[][3])
32 {
33
34     float sum;
35     for(int y = 1; y < src.rows - 1; y++){
36         for(int x = 1; x < src.cols - 1; x++){
37             sum = 0.0;
38             for(int k = -1; k <= 1; k++){
39                 for(int j = -1; j <= 1; j++){
40                     sum = sum + Kernel[j+1][k+1]*src.at<uchar>(y - j, x -
41                     )
42                 }
43             }
44             dst.at<uchar>(y,x) = sum;
45         }
46     }
47
48     void refletedIndexing(Mat src, Mat dst, float Kernel[][3])
49     {
50         float sum, x1, y1;
51         for(int y = 0; y < src.rows; y++){
52             for(int x = 0; x < src.cols; x++){
53                 sum = 0.0;
54                 for(int k = -1; k <= 1; k++){
55                     for(int j = -1; j <= 1; j++){
56                         x1 = reflect(src.cols, x - j);
57                         y1 = reflect(src.rows, y - k);
58                         sum = sum + Kernel[j+1][k+1]*src.at<uchar>(y1,x1);
59                     }
60                 }
61                 dst.at<uchar>(y,x) = sum;
62             }
63         }
64     }
65
66     void circularIndexing(Mat src, Mat dst, float Kernel[][3])
67     {
68         float sum, x1, y1;
69         for(int y = 0; y < src.rows; y++){
70             for(int x = 0; x < src.cols; x++){
71                 sum = 0.0;
72                 for(int k = -1; k <= 1; k++){
73                     for(int j = -1; j <= 1; j++){
74                         x1 = circular(src.cols, x - j);
75                         y1 = circular(src.rows, y - k);
76                         sum = sum + Kernel[j+1][k+1]*src.at<uchar>(y1,x1);
77                     }
78                 }
79                 dst.at<uchar>(y,x) = sum;
80             }
81         }
82     }
83
84     int main()
85     {
86
87         Mat src, dst;
88
89
90         /// Load an image
91         src = imread("salt.jpg", CV_LOAD_IMAGE_GRAYSCALE);
92
93         if( !src.data )
94         { return -1; }
95
96
97         float Kernel[3][3] = {
98             {1/9.0, 1/9.0, 1/9.0},
99             {1/9.0, 1/9.0, 1/9.0},
100            {1/9.0, 1/9.0, 1/9.0}
101        };
102

```

Median Filter using C++ and Op
Image Processi...

Calculating a convolution of an
with C++: I...

► January (6)

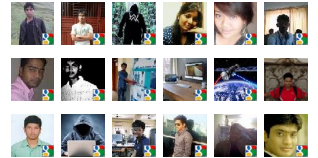
► 2012 (74)

► 2011 (128)

Search This Blog

Followers

Followers (132) [Next](#)



[Follow](#)

Flag Counter

Visitors			See more▶
	427,570		20,709
	179,504		20,197
	54,572		18,360
	23,574		17,070
	22,318		14,882
	21,550		14,136
			11,978
			11,937
			11,886
			11,572
			10,917
			10,008

FLAG counter

[Free counters](#)

```
103     dst = src.clone();
104     for(int y = 0; y < src.rows; y++)
105         for(int x = 0; x < src.cols; x++)
106             dst.at<uchar>(y,x) = 0.0;
107
108     circularIndexing(src, dst, Kernel);
109
110
111     namedWindow("final");
112     imshow("final", dst);
113
114     namedWindow("initial");
115     imshow("initial", src);
116
117     waitKey();
118
119
120
121     return 0;
122 }
```

About Author



Bibek Subedi is a computer engineering graduate and founder of Programing Techniques. He loves researching in the field of Machine learning, data mining and Algorithms. He is a part time blogger, a bathroom singer (:D) and an employee of a software company. You can follow him in [Twitter](#) and find him in [Facebook](#) or [mail him](#)

2 comments:

Anonymous [August 8, 2015 at 12:41 PM](#)

I am not able to understand why are you using "dst.at(y,x) = 0.0;" when you have already created a cloned image. Help. Thanks.

[Reply](#)

Anonymous [July 29, 2016 at 1:28 PM](#)

65456

[Reply](#)

Enter your comment...

Comment as: Unknown (Google)

[Sign out](#)

[Publish](#)

[Preview](#)

☐ Notify me

Links to this post

[Create a Link](#)

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

02134379 [View My Stats](#)

DMCA PROTECTED

Bibek Subedi. Powered by [Blogger](#).

[←Back](#)

Convolution

Convolution is the most important and fundamental concept in signal processing and analysis. By using convolution, we can construct the output of system for any arbitrary input signal, if we know the impulse response of system.

How is it possible that knowing only impulse response of system can determine the output for any given input signal? We will find out the meaning of convolution.

Related Topics: [Window Filters](#)

Download: [conv1d.zip](#), [conv2d.zip](#)

- [Definition](#)
- [Impulse Function Decomposition](#)
- [Impulse Response](#)
- [Back to the Definition](#)
- [Convolution in 1D](#)
- [C++ Implementation for Convolution 1D](#)
- [Convolution in 2D](#)
- [Separable Convolution 2D](#)
- [C++ Algorithm for Convolution 2D](#)

Definition

First, let's see the mathematical definition of convolution in discrete time domain. Later we will walk through what this equation tells us.

(We will discuss in discrete time domain only.)

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

where $x[n]$ is input signal, $h[n]$ is impulse response, and $y[n]$ is output. $*$ denotes convolution. Notice that we multiply the terms of $x[k]$ by the terms of a time-shifted $h[n]$ and add them up.

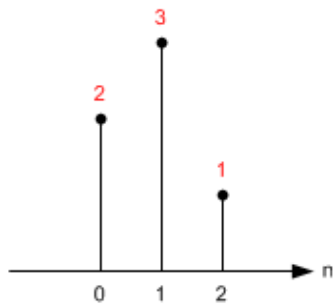
The keystone of understanding convolution is laid behind impulse response and impulse decomposition.

Impulse Function Decomposition

In order to understand the meaning of convolution, we are going to start from the concept of signal decomposition. The input signal is decomposed into simple additive components, and the system response of the input signal results in by adding the output of these components passed through the system.

In general, a signal can be decomposed as a weighted sum of basis signals. For example, in Fourier Series, any periodic signal (even rectangular pulse signal) can be represented by a sum of sine and cosine functions. But here, we use impulse (delta) functions for the basis signals, instead of sine and cosine.

Examine the following example how a signal is decomposed into a set of impulse (delta) functions. Since the impulse function, $\delta[n]$ is 1 at $n=0$, and zeros at $n \neq 0$. $x[0]$ can be written to $2 \cdot \delta[n]$. And, $x[1]$ will be $3 \cdot \delta[n-1]$, because $\delta[n-1]$ is 1 at $n=1$ and zeros at others. In same way, we can write $x[2]$ by shifting $\delta[n]$ by 2, $x[2] = 1 \cdot \delta[n-2]$. Therefore, the signal, $x[n]$ can be represented by adding 3 shifted and scaled impulse functions.



$$x[0] = x[0] \cdot \delta[n] = 2 \cdot \delta[n-0]$$

$$x[1] = x[1] \cdot \delta[n-1] = 3 \cdot \delta[n-1]$$

$$x[2] = x[2] \cdot \delta[n-2] = 1 \cdot \delta[n-2]$$

$$x[n] = x[0] \cdot \delta[n-0] + x[1] \cdot \delta[n-1] + x[2] \cdot \delta[n-2]$$

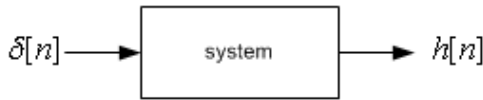
In general, a signal can be written as sum of scaled and shifted delta functions;

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot \delta[n-k]$$

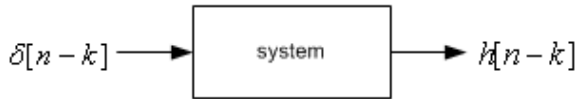
Impulse Response

Impulse response is the output of a system resulting from an impulse function as input.

And it is denoted as $h[n]$.

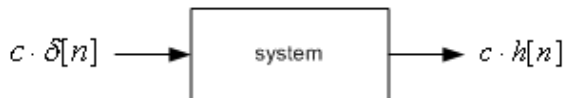


If the system is [time-invariant](#), the response of a time-shifted impulse function is also shifted as same amount of time.



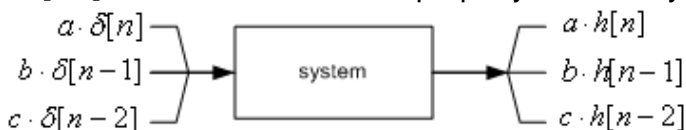
For example, the impulse response of $\delta[n-1]$ is $h[n-1]$. If we know the impulse response $h[n]$, then we can immediately get the impulse response $h[n-1]$ by shifting $h[n]$ by +1. Consequently, $h[n-2]$ results from shifting $h[n]$ by +2.

If the system is [linear](#) (especially scalar rule), a scaled in input signal causes an identical scaling in the output signal.



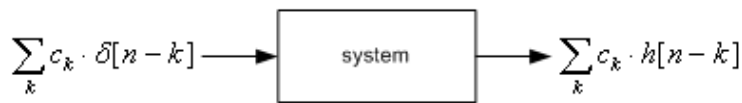
For instance, the impulse response of $3 \cdot \delta[n]$ is just multiplying by 3 to $h[n]$.

If the input has 3 components, for example, $a \cdot \delta[n] + b \cdot \delta[n-1] + c \cdot \delta[n-2]$, then the output is simply $a \cdot h[n] + b \cdot h[n-1] + c \cdot h[n-2]$. This is called additive property of [linear](#) system, thus, it is valid only on the [linear](#) system.



Back to the Definition

By combining the properties of impulse response and impulse decomposition, we can finally construct the equation of convolution. In [linear](#) and [time-invariant system](#), the response resulting from several inputs can be computed as the sum of the responses each input acting alone.



For example, if input signal is $x[n] = 2 \cdot \delta[n] + 3 \cdot \delta[n-1] + 1 \cdot \delta[n-2]$, then the output is simply $y[n] = 2 \cdot h[n] + 3 \cdot h[n-1] + 1 \cdot h[n-2]$.

Therefore, we now clearly see that if the input signal is $x[n] = \sum_k x[k] \cdot \delta[n-k]$, then the output will be $y[n] = \sum_k x[k] \cdot h[n-k]$. Note one condition; convolution works on the [linear](#) and [time invariant system](#).

To summarize, a signal is decomposed into a set of impulses and the output signal can be computed by adding the scaled and shifted impulse responses.

Furthermore, there is an important fact under convolution; the only thing we need to know about the system's characteristics is the impulse response of the system, $h[n]$. If we know a system's impulse response, then we can easily find out how the system reacts for any input signal.

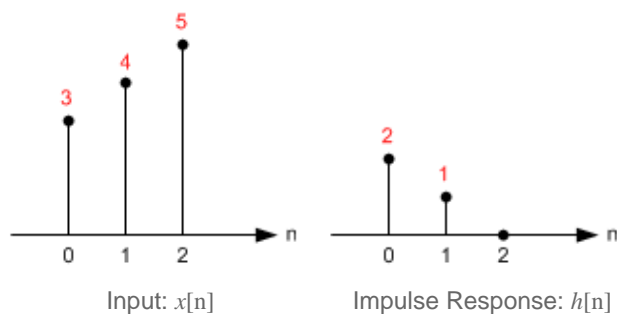
Convolution in 1D

Let's start with an example of convolution of 1 dimensional signal, then find out how to implement into computer programming algorithm.

$$x[n] = \{ 3, 4, 5 \}$$

$$h[n] = \{ 2, 1 \}$$

$x[n]$ has only non-zero values at $n=0,1,2$, and impulse response, $h[n]$ is not zero at $n=0,1$. Others which are not listed are all zeros.



One thing to note before we move on: Try to figure out yourself how this system behaves, by only looking at the impulse response of the system. When the impulse signal is entered the system, the output of the system looks like amplifier and echoing. At the time is 0, the intensity was increased (amplified) by double and gradually decreased while the time is passed.

From the equation of convolution, the output signal $y[n]$ will be $y[n] = \sum_k x[k] \cdot h[n-k]$.

Let's compute manually each value of $y[0]$, $y[1]$, $y[2]$, $y[3]$, ...

$$y[0] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[0-k] = x[0] \cdot h[0] = 3 \cdot 2 = 6$$

$$\begin{aligned} y[1] &= \sum_{k=-\infty}^{\infty} x[k] \cdot h[1-k] = x[0] \cdot h[1-0] + x[1] \cdot h[1-1] + \dots \\ &= x[0] \cdot h[1] + x[1] \cdot h[0] = 3 \cdot 1 + 4 \cdot 2 = 11 \end{aligned}$$

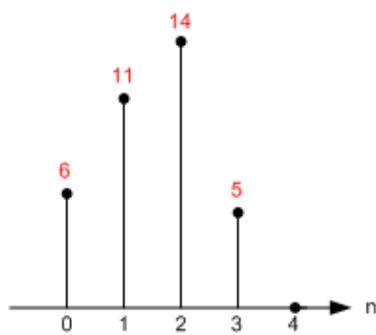
$$y[2] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[2-k] = x[0] \cdot h[2-0] + x[1] \cdot h[2-1] + x[2] \cdot h[2-2] + \dots$$

$$= x[0] \cdot h[2] + x[1] \cdot h[1] + x[2] \cdot h[0] = 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 2 = 14$$

$$y[3] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[3-k] = x[0] \cdot h[3-0] + x[1] \cdot h[3-1] + x[2] \cdot h[3-2] + x[3] \cdot h[3-3] + \dots$$

$$= x[0] \cdot h[3] + x[1] \cdot h[2] + x[2] \cdot h[1] + x[3] \cdot h[0] = 0 + 0 + 5 \cdot 1 + 0 = 5$$

$$y[4] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[4-k] = x[0] \cdot h[4-0] + x[1] \cdot h[4-1] + x[2] \cdot h[4-2] + \dots = 0$$



Output: $y[n]$

Notice that $y[0]$ has only one component, $x[0]h[0]$, and others are omitted, because others are all zeros at $k \neq 0$. In other words, $x[1]h[-1] = x[2]h[-2] = 0$. The above equations also omit the terms if they are obviously zeros.

If n is larger than and equal to 4, $y[n]$ will be zeros. So we stop here to compute $y[n]$ and adding these 4 signals ($y[0], y[1], y[2], y[3]$) produces the output signal $y[n] = \{6, 11, 14, 5\}$.

Let's look more closely the each output. In order to see a pattern clearly, the order of addition is swapped. The last term comes first and the first term goes to the last. And, all zero terms are ignored.

$$y[0] = x[0] \cdot h[0]$$

$$y[1] = x[1] \cdot h[0] + x[0] \cdot h[1]$$

$$y[2] = x[2] \cdot h[0] + x[1] \cdot h[1]$$

$$y[3] = x[3] \cdot h[0] + x[2] \cdot h[1]$$

Can you see the pattern? For example, $y[2]$ is calculated from 2 input samples; $x[2]$ and $x[1]$, and 2 impulse response samples; $h[0]$ and $h[1]$. The input sample starts from 2, which is same as the sample point of output, and decreased. The impulse response sample starts from 0 and increased.

Based on the pattern that we found, we can write an equation for any sample of the output;

$$y[i] = x[i] \cdot h[0] + x[i-1] \cdot h[1] + x[i-2] \cdot h[2] + \dots + x[i-(k-1)] \cdot h[k-1]$$

where i is any sample point and k is the number of samples in impulse response.

For instance, if an impulse response has 4 samples, the sample of output signal at 9 is;

$$y[9] = x[9] \cdot h[0] + x[8] \cdot h[1] + x[7] \cdot h[2] + x[6] \cdot h[3]$$

Notice the first sample, $y[0]$ has only one term. Based on the pattern that we found, $y[0]$ is calculated as:

$$y[0] = x[0] \cdot h[0] + x[-1] \cdot h[1]. \text{ Because } x[-1] \text{ is not defined, we simply pad it to zero.}$$

C++ Implementation for Convolution 1D

Implementing the convolution algorithm is quite simple. The code snippet is following;


```
for ( i = 0; i < sampleCount; i++ )
{
    y[i] = 0;                // set to zero before sum
    for ( j = 0; j < kernelCount; j++ )
    {
        y[i] += x[i - j] * h[j];    // convolve: multiply and accumulate
    }
}
```

However, you should concern several things in the implementation.

Watch out the range of input signal. You may be out of bound of input signal, for example, $x[-1]$, $x[-2]$, and so on. You can pad zeros for those undefined samples, or simply skip the convolution at the boundary. The results at the both the beginning and end edges cannot be accurate anyway.

Second, you need rounding the output values, if the output data type is integer and impulse response is floating point number. And, the output value may be exceeded the maximum or minimum value.

If you have unsigned 8-bit integer data type for output signal, the range of output should be between 0 and 255. You must check the value is greater than the minimum and less than the maximum value.

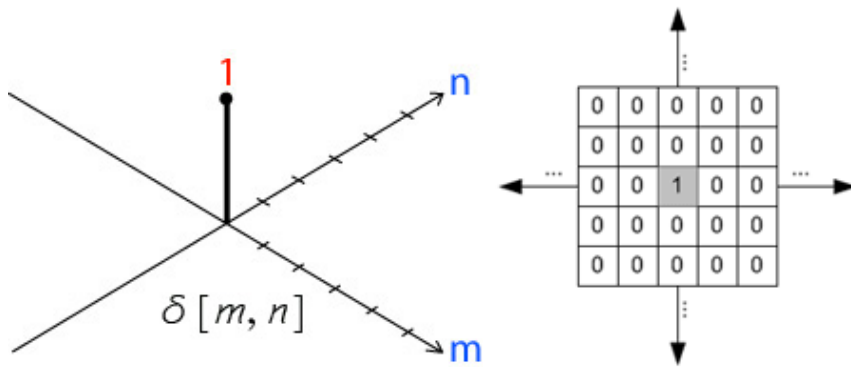
Download the 1D convolution routine and test program.

[conv1d.zip](#)

Convolution in 2D

2D convolution is just extension of previous 1D convolution by convolving both horizontal and vertical directions in 2 dimensional spatial domain. Convolution is frequently used for image processing, such as smoothing, sharpening, and edge detection of images.

The impulse (delta) function is also in 2D space, so $\delta[m, n]$ has 1 where m and n is zero and zeros at $m, n \neq 0$. The impulse response in 2D is usually called "kernel" or "filter" in image processing.



The second image is 2D matrix representation of impulse function. The shaded center point is the origin where $m=n=0$.

Once again, a signal can be decomposed into a sum of scaled and shifted impulse (delta) functions;

$$x[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] \cdot \delta[m-i, n-j]$$

For example, $x[0, 0]$ is $x[0, 0] \cdot \delta[m, n]$, $x[1, 2]$ is $x[1, 2] \cdot \delta[m-1, n-2]$, and so on. Note that the matrices are referenced here as [column, row], not [row, column]. M is horizontal (column) direction and N is vertical (row) direction.

And, the output of [linear](#) and [time invariant system](#) can be written by convolution of input signal $x[m, n]$, and

impulse response, $h[m, n]$;

$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] \cdot h[m-i, n-j]$$

Notice that the kernel (impulse response) in 2D is center originated in most cases, which means the center point of a kernel is $h[0, 0]$. For example, if the kernel size is 5, then the array index of 5 elements will be -2, -1, 0, 1, and 2. The origin is located at the middle of kernel.

$m \backslash n$	-1	0	1
-1	a	b	c
0	d	e	f
1	g	h	i

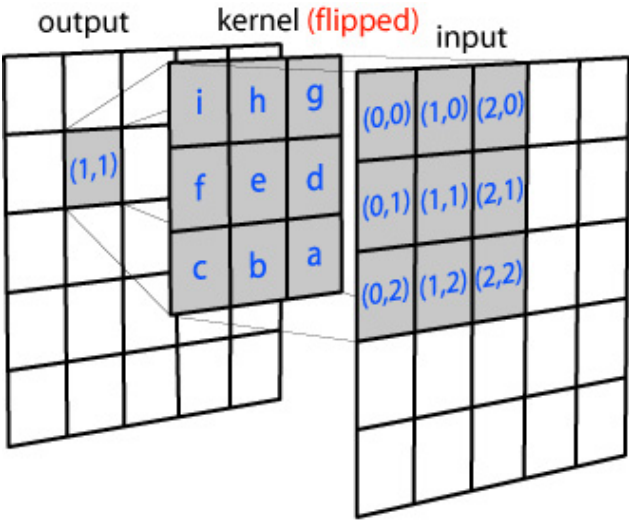
Examine an example to clarify how to convolve in 2D space.
Let's say that the size of impulse response (kernel) is 3x3, and it's values are a, b, c, d,...

Notice the origin (0,0) is located in the center of kernel.

Let's pick a simplest sample and compute convolution, for instance, the output at (1, 1) will be;

$$\begin{aligned} y[1, 1] &= \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] \cdot h[1-i, 1-j] \\ &= x[0, 0] \cdot h[1, 1] + x[1, 0] \cdot h[0, 1] + x[2, 0] \cdot h[-1, 1] \\ &\quad + x[0, 1] \cdot h[1, 0] + x[1, 1] \cdot h[0, 0] + x[2, 1] \cdot h[-1, 0] \\ &\quad + x[0, 2] \cdot h[1, -1] + x[1, 2] \cdot h[0, -1] + x[2, 2] \cdot h[-1, -1] \end{aligned}$$

It results in sum of 9 elements of scaled and shifted impulse responses. The following image shows the graphical representation of 2D convolution.



Notice that the kernel matrix is flipped both horizontal and vertical direction before multiplying the overlapped input data, because $x[0,0]$ is multiplied by the last sample of impulse response, $h[1,1]$. And $x[2,2]$ is multiplied by the first sample, $h[-1,-1]$.

2D Convolution

Exercise a little more about 2D convolution with another example. Suppose we have 3x3 kernel and 3x3 input matrix.

1	2	3
4	5	6
7	8	9

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

-13	-20	-17
-18	-24	-18
13	20	17

Input

Kernel

Output

The complete solution for this example is here; [Example of 2D Convolution](#)

By the way, the kernel in this example is called **Sobel** filter, which is used to detect the horizontal edge lines in an image. See more details in the [window filters](#).

Separable Convolution 2D

In convolution 2D with $M \times N$ kernel, it requires $M \times N$ multiplications for each sample. For example, if the kernel size is 3×3 , then, 9 multiplications and accumulations are necessary for each sample. Thus, convolution 2D is very expensive to perform multiply and accumulate operation.

However, if the kernel is separable, then the computation can be reduced to $M + N$ multiplications.

A matrix is separable if it can be decomposed into $(M \times 1)$ and $(1 \times N)$ matrices.

For example;

$$\begin{bmatrix} A \cdot a & A \cdot b & A \cdot c \\ B \cdot a & B \cdot b & B \cdot c \\ C \cdot a & C \cdot b & C \cdot c \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot \begin{bmatrix} a & b & c \end{bmatrix}$$

And, convolution with this separable kernel is equivalent to;

$$x[m,n] * \begin{bmatrix} A \cdot a & A \cdot b & A \cdot c \\ B \cdot a & B \cdot b & B \cdot c \\ C \cdot a & C \cdot b & C \cdot c \end{bmatrix} = x[m,n] * \left(\begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot \begin{bmatrix} a & b & c \end{bmatrix} \right) = \left(x[m,n] * \begin{bmatrix} A \\ B \\ C \end{bmatrix} \right) * \begin{bmatrix} a & b & c \end{bmatrix}$$

[\(Proof of Separable Convolution 2D\)](#)

As a result, in order to reduce the computation, we perform 1D convolution twice instead of 2D convolution; convolve with the input and $M \times 1$ kernel in vertical direction, then convolve again horizontal direction with the result from the previous convolution and $1 \times N$ kernel. The first vertical 1D convolution requires M times of multiplications and the horizontal convolution needs N times of multiplications, altogether, $M+N$ products.

However, the separable 2D convolution requires additional storage (buffer) to keep intermediate computations. That is, if you do vertical 1D convolution first, you must preserve the results in a temporary buffer in order to use them for horizontal convolution subsequently.

Notice that convolution is associative; the result is same, even if the order of convolution is changed. So, you may convolve horizontal direction first then vertical direction later.

Gaussian smoothing filter is a well-known separable matrix. For example, 3×3 Gaussian filter is;

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

3x3 Gaussian Kernel

C++ Algorithm for Convolution 2D

We need 4 nested loops for 2D convolution instead of 2 loops in 1D convolution.

```
// find center position of kernel (half of kernel size)
kCenterX = kCols / 2;
kCenterY = kRows / 2;

for(i=0; i < rows; ++i)           // rows
{
    for(j=0; j < cols; ++j)       // columns
    {
        for(m=0; m < kRows; ++m) // kernel rows
        {
            mm = kRows - 1 - m;    // row index of flipped kernel

            for(n=0; n < kCols; ++n) // kernel columns
            {
                nn = kCols - 1 - n; // column index of flipped kernel

                // index of input signal, used for checking boundary
                ii = i + (m - kCenterY);
                jj = j + (n - kCenterX);

                // ignore input samples which are out of bound
                if( ii >= 0 && ii < rows && jj >= 0 && jj < cols )
                    out[ii][jj] += in[ii][jj] * kernel[mm][nn];
            }
        }
    }
}
```

The above snippet code is simple and easiest way to understand how convolution works in 2D. But it may be the slowest implementation.

Take a look at a real example; convolution with 256x256 image and 5x5 Gaussian filter.



The source image is uncompressed raw, 8-bit (unsigned char) grayscale image. And again, Gaussian kernel is separable;

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{256} \cdot \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \cdot [1 \quad 4 \quad 6 \quad 4 \quad 1]$$

5x5 Gaussian Kernel

On my system (AMD 64 3200+ 2GHz), normal convolution took about 10.3 ms and separable convolution took only 3.2 ms. You can see how much separable convolution is faster compared to normal convolution.

Download 2D convolution application and source code here: [conv2d.zip](#)


The program uses OpenGL to render images on the screen.



[←Back](#)

Hide Comments

[comments powered by Disqus](#)



WIKIPEDIA
The Free Encyclopedia

Main page

Contents

Featured content

Current events

Random article

Donate to Wikipedia

Wikipedia store

Interaction

Help

About Wikipedia

Community portal

Recent changes

Contact page

Tools

What links here

Upload file

Special pages

Page information


Languages

File

Talk

Read

View on Commons

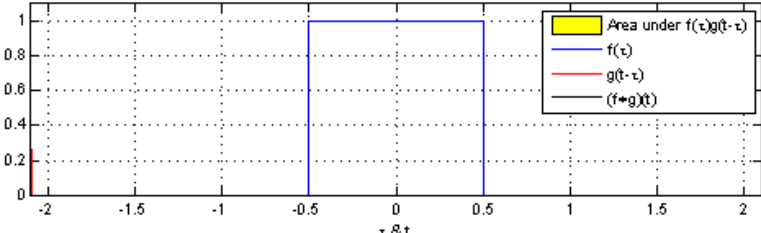


File:Convolution of box signal with itself2.gif




From Wikipedia, the free encyclopedia

FileFile historyFile usageGlobal file usageMetadata



No higher resolution available.

Convolution_of_box_signal_with_itself2.gif (468 × 147 pixels, file size: 83 KB, MIME type: image/gif , looped, 301 frames, 15 s)

 This is a file from the [Wikimedia Commons](#) . Information from its [description page there](#) is shown below. Commons is a freely licensed media file repository. [You can help.](#)

Description

An animation displaying the process and result of convolving a box signal with itself. For each position in the resulting hat function the input functions and the multiplied input functions are shown, and the area which is integrated is marked. This work mimics [\[1\]](#) but includes the source code and shows the result of the convolution in the same graph as the input functions.

Date


25 July 2010, 23:41 (UTC)

Source

- Convolution_of_box_signal_with_itself.gif

Author

- Convolution_of_box_signal_with_itself.gif: Brian Amberg
- derivative work: Tinos (talk)

 This graphic was created with [MATLAB](#).

MATLAB Code [\[edit\]](#)

```
% Create folding of two rectangular impulses

clear

X=-2.6:0.001:3;
F1=abs(X)<=0.5;
F2=abs(X)<=0.5;
clf

mkdir('tmp');
[tmp zero_offset] = min(abs(X));

SyncFrames=[1 round(18.67*(1:numel(X)))];
integral=nan(size(X));
frame=1;
for offset_i=1:numel(X);
    offset=X(offset_i);
    shift=offset_i-zero_offset;
    F2_shifted = circshift(F2, [0 shift]);
```

```
product = F2_shifted.*F1;
integral(offset_i) = sum(product)/numel(X)*(X(end)-X(1));

if offset_i==SyncFrames(frame)
    frame=frame+1;
    area(X, product, 'facecolor', 'yellow');
    hold on
    plot(X, F1, 'b', X, F2_shifted, 'r', X, integral, 'k', [offset offset], [0
2], 'k:');
    hold off
    axis image
    axis([-2.1 2.1 0 1.1])
    xlabel('\tau & t');
    grid on
    legend('Area under f(\tau)g(t-\tau)', 'f(\tau)', 'g(t-\tau)', '(f\ast g
(t)');
    print('-dpng', '-r72', sprintf('tmp/conv_box_box_%06d.png', offset_i));
    drawnow
end
end

system('"C:\Program Files\ImageMagick-6.6.3-Q16\convert.exe" -layers Optimize -delay
5 tmp/conv_box_box_*.png conv_box_box.gif');
delete('tmp/*');
rmdir('tmp');
```

Photoshop was then used to crop the animation.



This is a *retouched picture*, which means that it has been digitally altered from its original version. Modifications: *Fixed asterisk, and put tau in..* The original can be viewed here: [Convolution of box signal with itself.gif](#). Modifications made by [Tinos](#).

I, the copyright holder of this work, hereby publish it under the following licenses:

This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **share alike** – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.



Permission is granted to copy, distribute and/or modify this document under the terms of the [GNU Free Documentation License](#), Version 1.2 or any later version published by the [Free Software Foundation](#); with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*.

You may select the license of your choice.

Original upload log [\[edit\]](#)

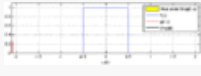
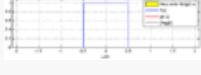
This image is a derivative work of the following images:

- File:Convolution_of_box_signal_with_itself.gif licensed with Cc-by-sa-3.0, GFDL
 - 2010-07-07T18:44:30Z Rubybrian 474x145 (79321 Bytes) *Cropped the image and compressed it using the commandline convert -layers Optimize -delay 5 -loop 0 conv_box_box_00*.png conv_box_box.gif*
 - 2010-07-07T18:14:59Z Rubybrian 587x159 (725000 Bytes) *{{Information |Description=An animation displaying the process and result of convolving a box signal with itself. For each position in the resulting hat function the input functions and the multiplied input functions are show*

Uploaded with [derivativeFX](#)

File history

Click on a date/time to view the file as it appeared at that time.

	Date/Time	Thumbnail	Dimensions	User	Comment
current	10:11, 26 July 2010		468 × 147 (83 KB)	Tinos	Synchronised with Convolution_of_spiky_function_with_box2.gif.
	23:47, 25 July 2010		470 × 150 (78 KB)	Tinos	{ {Information Description=An animation displaying the process and result of convolving a box signal with itself. For each position in the resulting hat function the input functions and the multiplied input functions are shown, and the area which is integ












File usage

The following pages on the English Wikipedia link to this file (pages on other projects are not listed):

- [Convolution](#)

Global file usage

The following other wikis use this file:

- Usage on de.wikipedia.org
 - [Faltung \(Mathematik\)](#) 
 - [Dreiecksfunktion](#) 
- Usage on it.wikipedia.org
 - [Convoluzione](#) 
- Usage on nl.wikipedia.org
 - [Convolutie](#) 
- Usage on pl.wikipedia.org
 - [Dyskusja wikipedysty:Konradek/Archiwum 13](#) 
 - [Funkcja trójkątna](#)
- Usage on pt.wikipedia.org
 - [Convolução](#) 
- Usage on ru.wikipedia.org
 - [Свёртка \(математический анализ\)](#)
- Usage on uk.wikipedia.org
 - [Згортка \(математичний аналіз\)](#)
- Usage on vi.wikipedia.org
 - [Hàm tri](#) 
 - [Tích chập](#) 
- Usage on zh.wikipedia.org



卷积 

Metadata

This file contains additional information, probably added from the digital camera or scanner used to create or digitize it. If the file has been modified from its original state, some details may not fully reflect the modified file.

Software used	Adobe Photoshop CS5 Windows
---------------	-----------------------------

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Cookie statement](#) [Mobile view](#)

