Basic image processing

Group operators

Image Feature Extraction Techniques

Piotr Fulmański

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
University of Lodz

Theory

Convolution

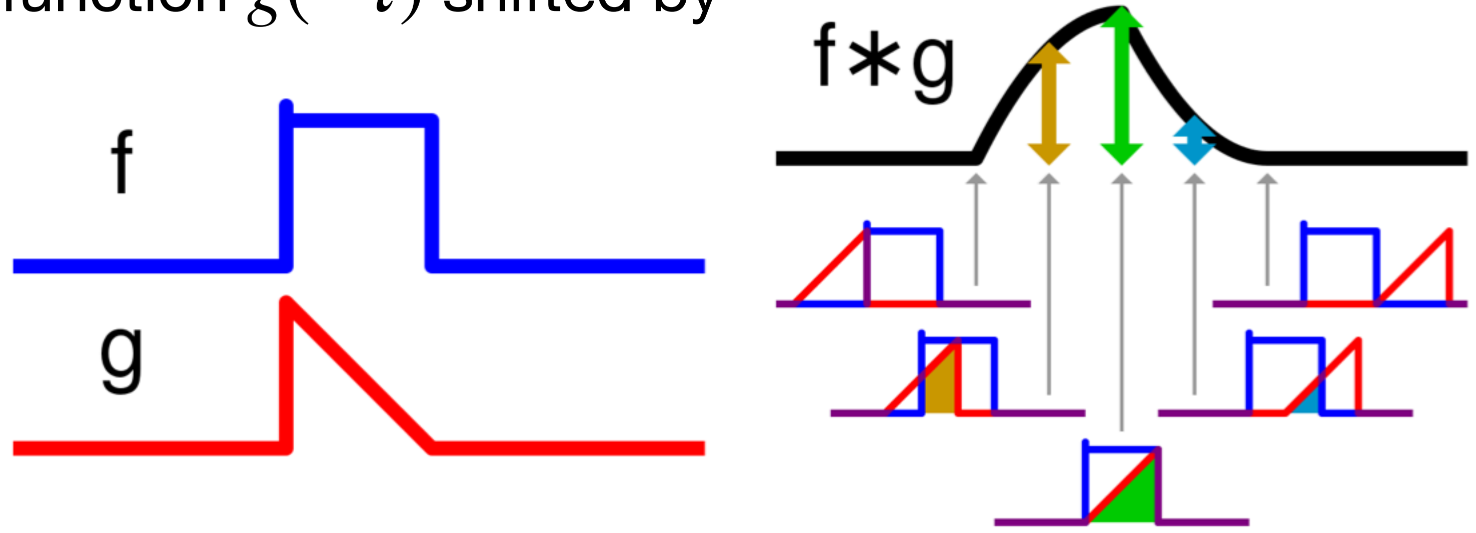
Definition

Convolution is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that **expresses how the shape of one is modified by the other**.

It is defined as the integral of the product of the two functions after one is reversed and shifted. The integral is evaluated for all values of shift, producing the convolution function:

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

The convolution formula can be described as the area under the function $f(\tau)$weighted by the function $g(-\tau)$ shifted by



The term *convolution* refers to both the result function and to the process of computing it.

Convolution

Definition

A common engineering notational convention is:

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau.}_{(f*g)(t)}$$

Properties:

- Commutativity:

$$f * g = g * f$$

- Associativity:

$$f * (g * h) = (f * g) * h$$

- Distributivity:

$$f * (g + h) = (f * g) + (f * h)$$

- Associativity with scalar multiplication:

$$a(f * g) = (af) * g$$

for any complex number a.

Convolution

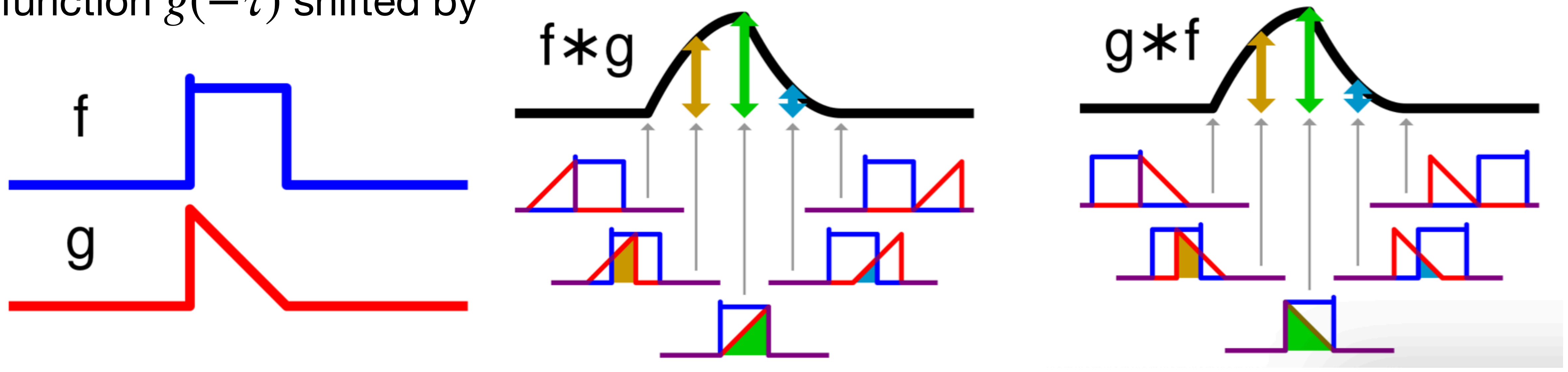
Definition

Convolution is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that **expresses how the shape of one is modified by the other.**

It is defined as the integral of the product of the two functions after one is reversed and shifted. The integral is evaluated for all values of shift, producing the convolution function:

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

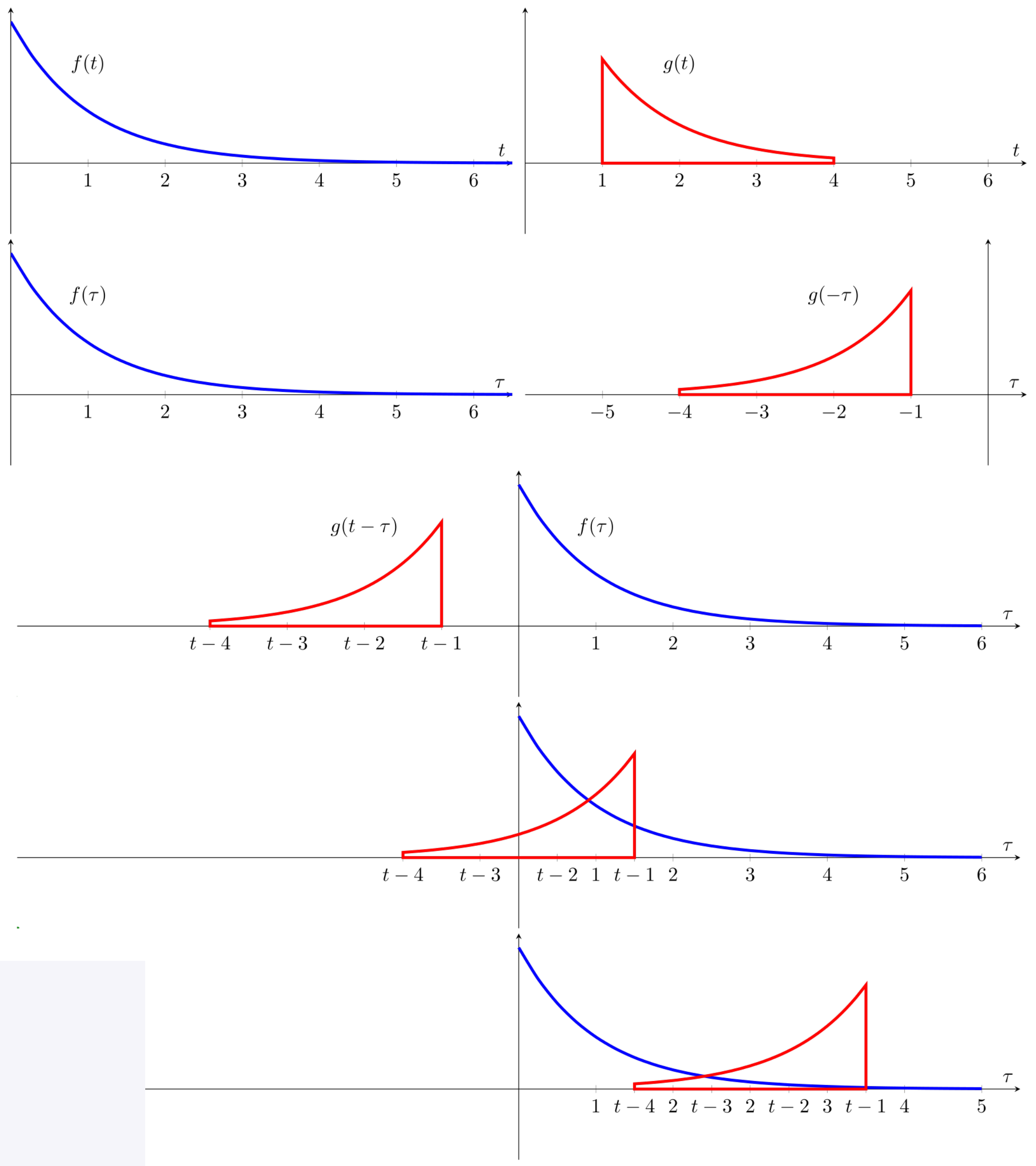
The convolution formula can be described as the area under the function $f(\tau)$weighted by the function $g(-\tau)$ shifted by

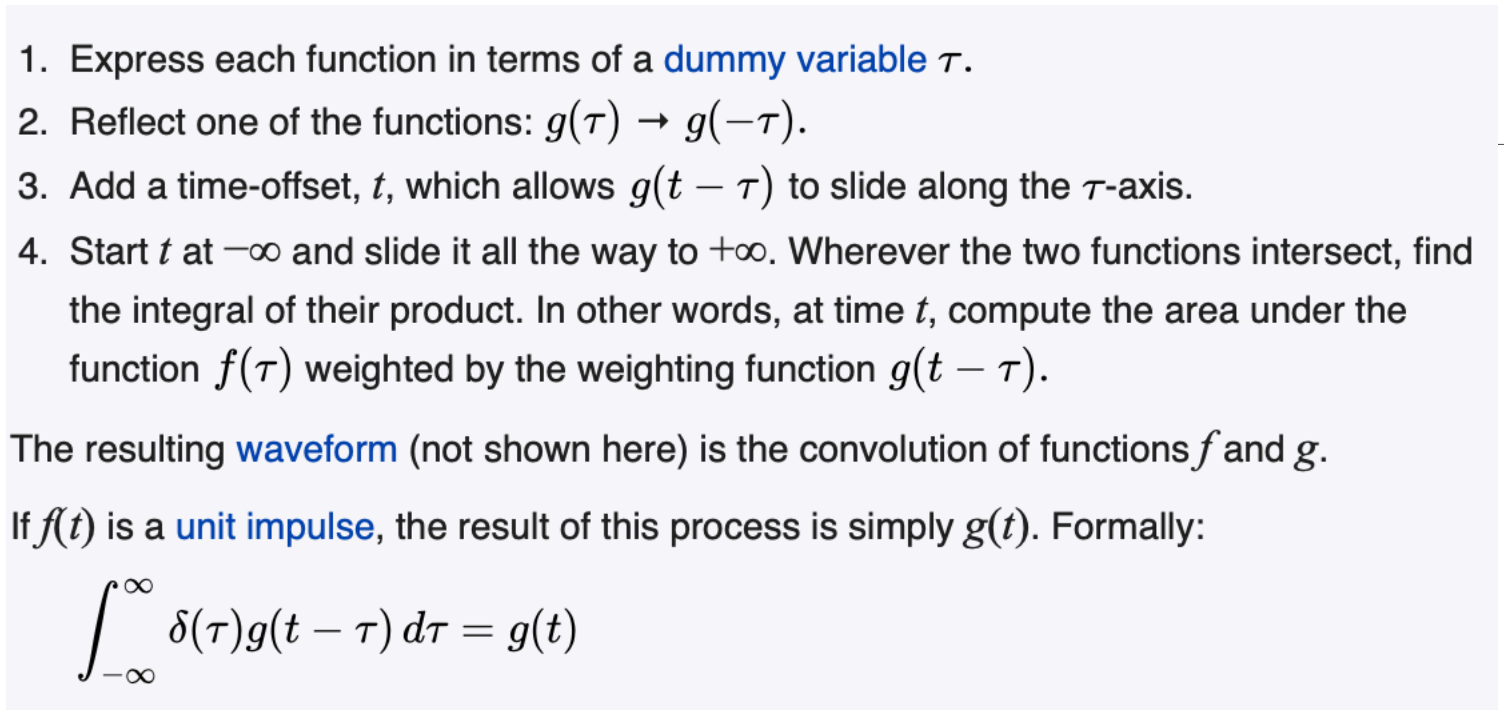


The term *convolution* refers to both the result function and to the process of computing it.

Convolution

Visual explanation [2]





1. Express each function in terms of a dummy variable τ.
2. Reflect one of the functions: $g(\tau) \rightarrow g(-\tau)$.
3. Add a time-offset, t, which allows $g(t-\tau)$ to slide along the τ-axis.
4. Start t at $-\infty$ and slide it all the way to $+\infty$. Wherever the two functions intersect, find the integral of their product. In other words, at time t, compute the area under the function $f(\tau)$ weighted by the weighting function $g(t-\tau)$.

The resulting waveform (not shown here) is the convolution of functions f and g.

If $f(t)$ is a unit impulse, the result of this process is simply $g(t)$. Formally:

$$\int_{-\infty}^{\infty} \delta(\tau) g(t-\tau)\, d\tau = g(t)$$

Convolution

Discrete convolution

For complex-valued functions f, g defined on the set Z of integers, the discrete convolution of f and g is given by:

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$

or equivalently (because of commutativity) by:

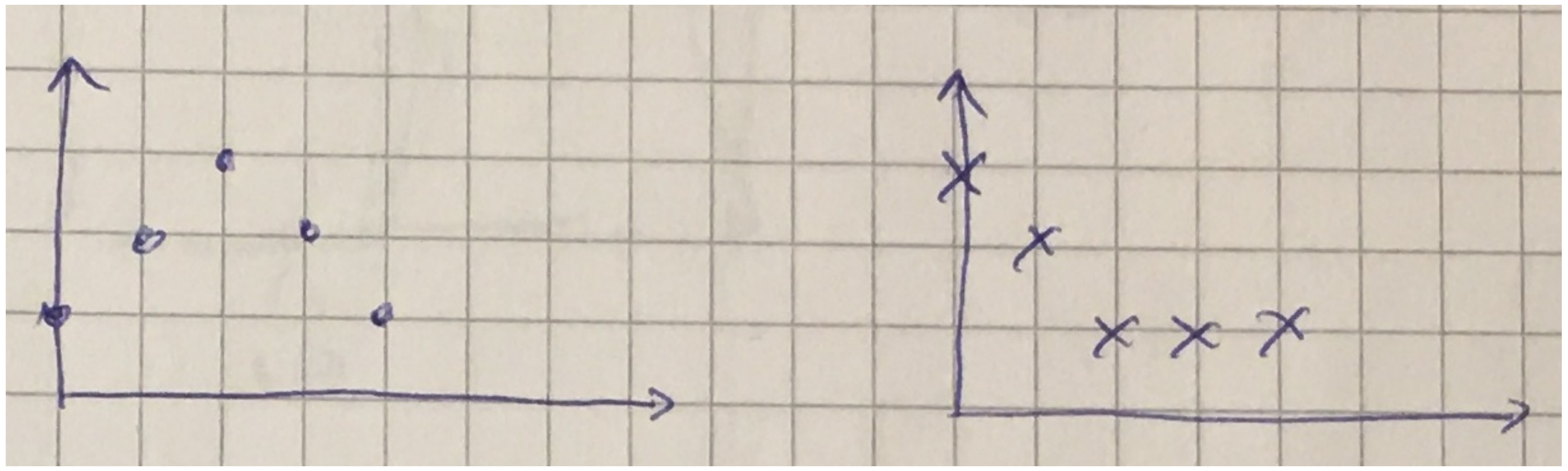
$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[n-k]g[k].$$

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4
f[n]  1 2 3 2 1
g[n]  3 2 1 1 1
```

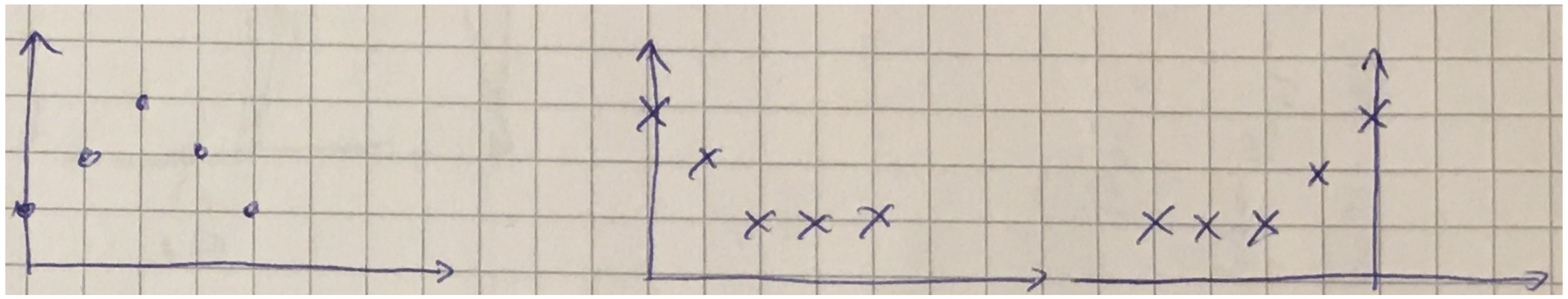


Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4  -4-3-2-1 0 1 2 3 4
f[n]  1 2 3 2 1           1 2 3 2 1
g[n]  3 2 1 1 1    1 1 1 2 3          <- matrix inversion
```

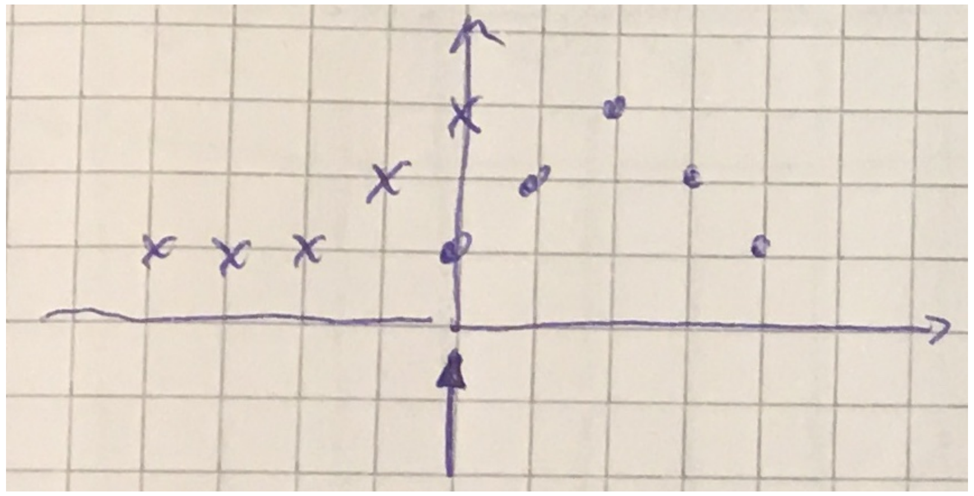


Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0  1  2  3  4   -4-3-2-1 0  1  2  3  4
f[n]  1  2  3  2  1            1  2  3  2  1
g[n]  3  2  1  1  1    1  1  1  2  3          <- matrix inversion
```



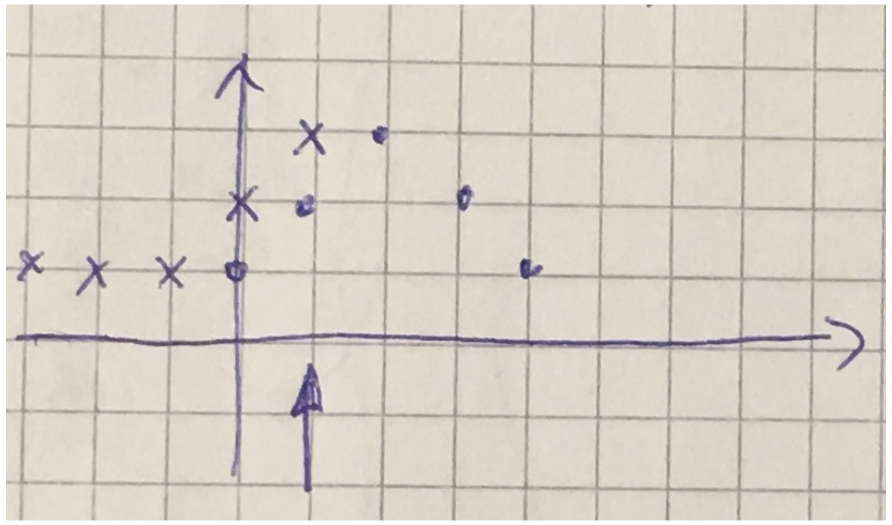
```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n    0 1 2 3 4  -4-3-2-1 0 1 2 3 4
f[n] 1 2 3 2 1           1 2 3 2 1
g[n] 3 2 1 1 1     1 1 1 2 3



y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
```

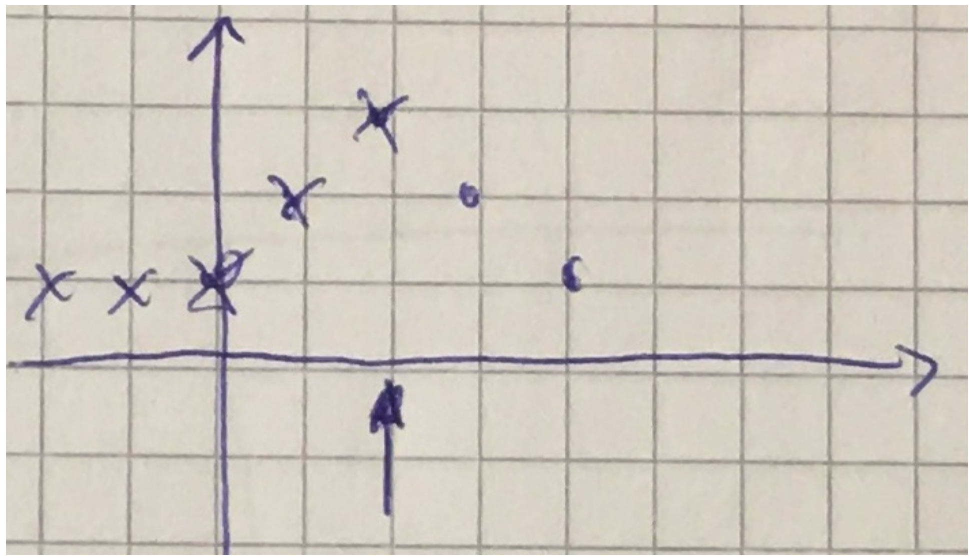


Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4  -4-3-2-1 0 1 2 3 4
f[n]  1 2 3 2 1           1 2 3 2 1
g[n]  3 2 1 1 1       1 1 1 2 3



y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
```

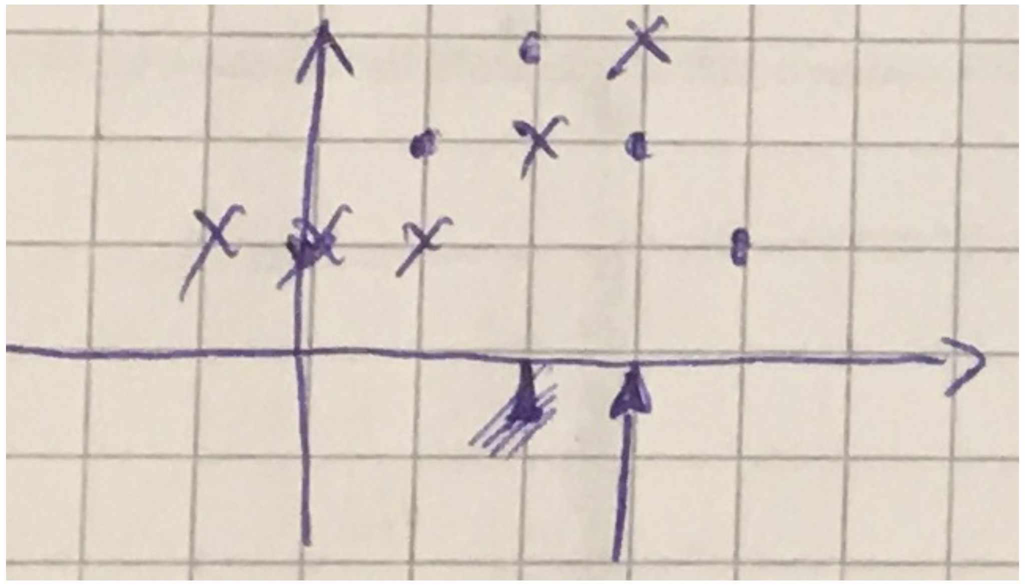


Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4   -4-3-2-1 0 1 2 3 4
f[n]  1 2 3 2 1            1 2 3 2 1
g[n]  3 2 1 1 1          1 1 1 2 3
```

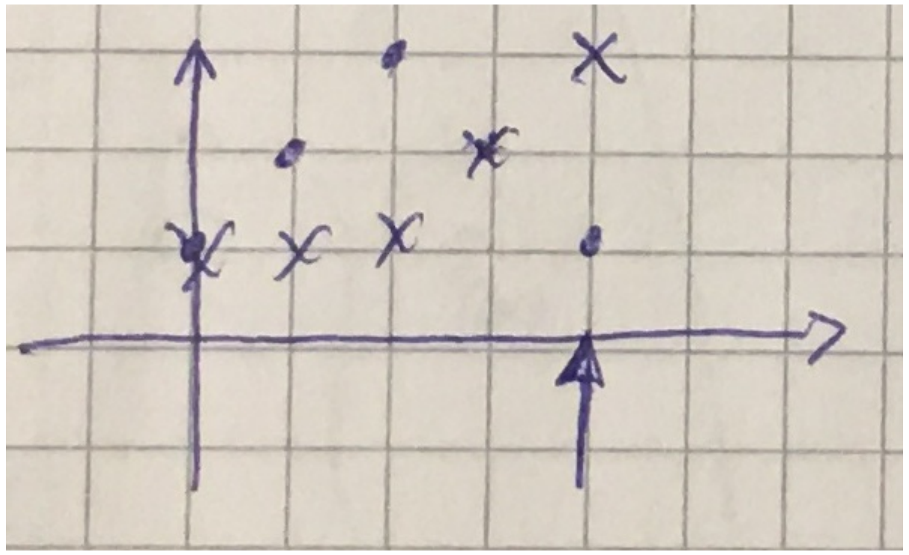


```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

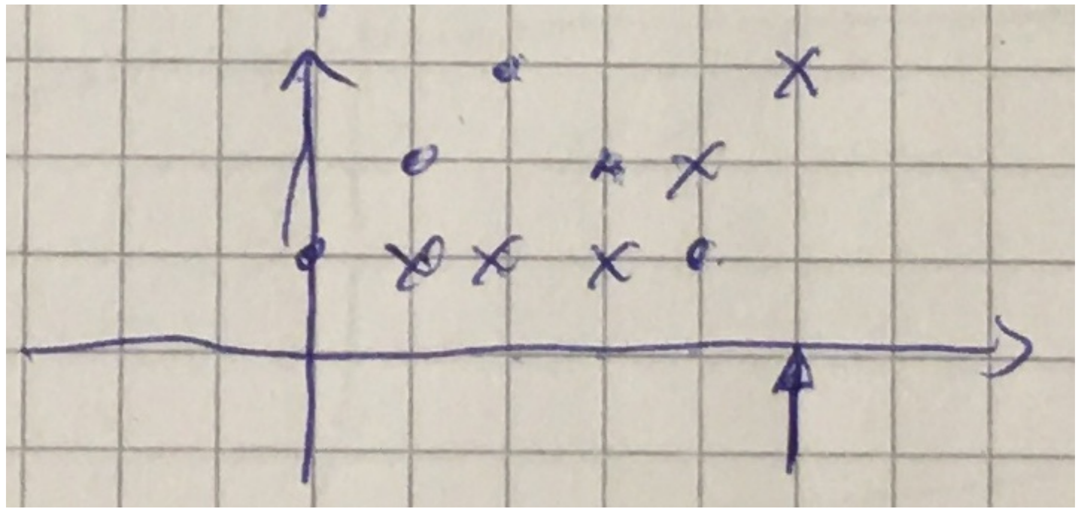


```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4  -4-3-2-1 0 1 2 3 4
f[n]  1 2 3 2 1           1 2 3 2 1
g[n]  3 2 1 1 1           1 1 1 2 3



y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$



```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0  1  2  3  4   -4-3-2-1  0  1  2  3  4  5
f[n]  1  2  3  2  1             1  2  3  2  1
g[n]  3  2  1  1  1                1  1  1  2  3
```

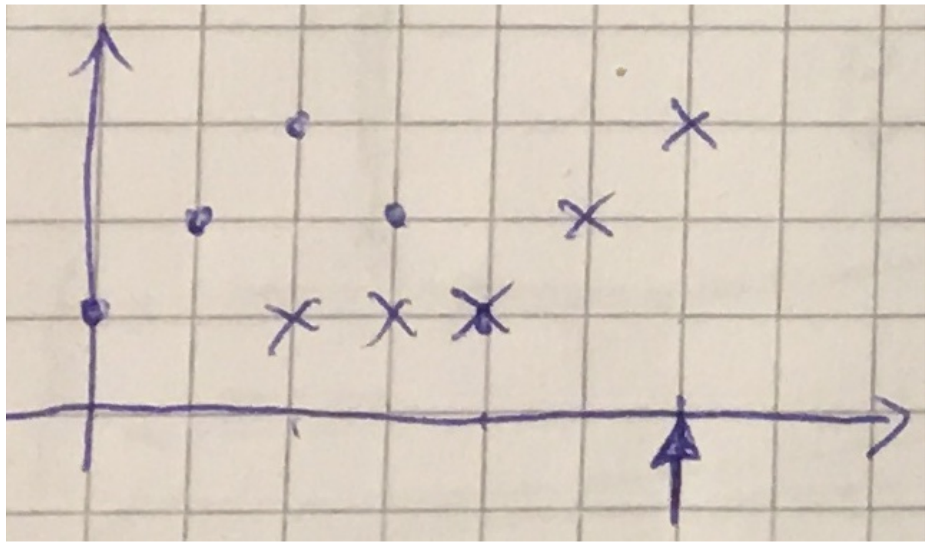
```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4   -4-3-2-1 0 1 2 3 4 5 6
f[n]  1 2 3 2 1            1 2 3 2 1
g[n]  3 2 1 1 1                1 1 1 2 3
```



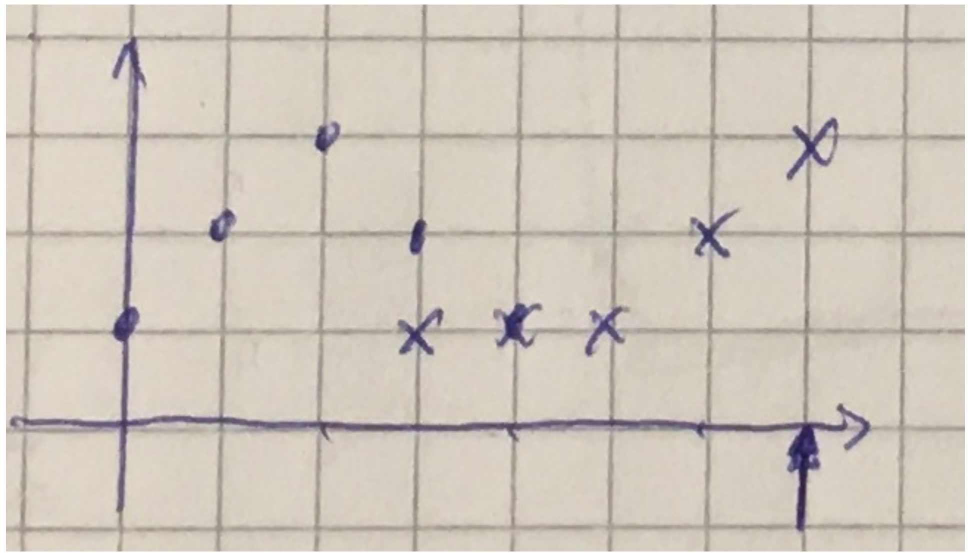
```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0 1 2 3 4  -4-3-2-1 0 1 2 3 4 5 6 7
f[n]  1 2 3 2 1           1 2 3 2 1
g[n]  3 2 1 1 1                 1 1 1 2 3
```



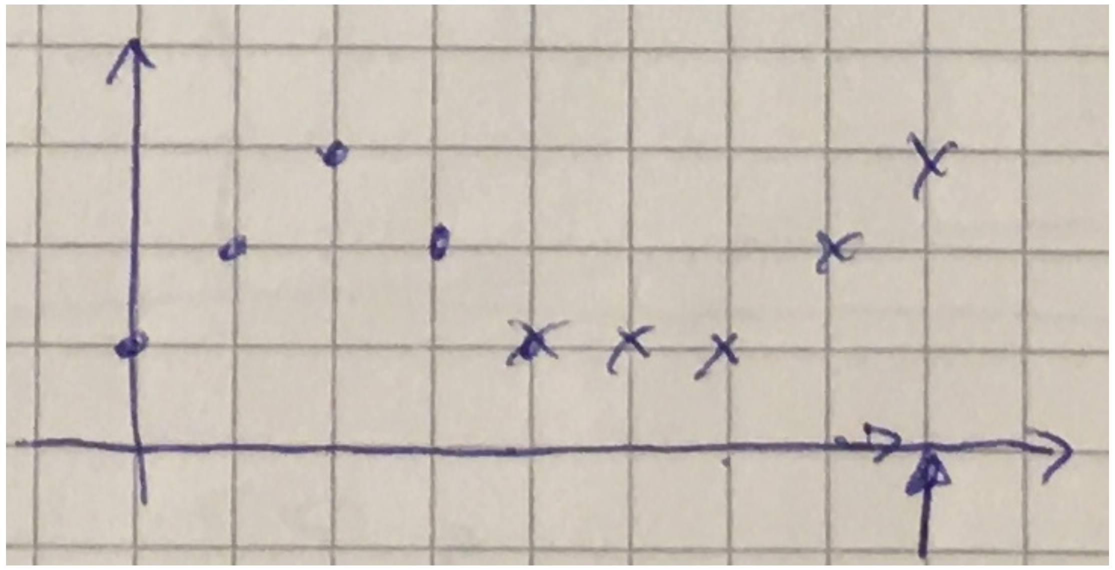
```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
y[7] = ... + f[-1]g[7+1] + f[0]g[7-0] + f[1]g[7-1] + f[2]g[7-2] + f[3]g[7-3] + f[4]g[7-4] + f[5]g[7-5] + ...
     = 2 + 1 = 3
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n    0 1 2 3 4  -4-3-2-1 0 1 2 3 4 5 6 7 8
f[n] 1 2 3 2 1           1 2 3 2 1
g[n] 3 2 1 1 1                   1 1 1 2 3
```



```
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
y[7] = ... + f[-1]g[7+1] + f[0]g[7-0] + f[1]g[7-1] + f[2]g[7-2] + f[3]g[7-3] + f[4]g[7-4] + f[5]g[7-5] + ...
     = 2 + 1 = 3
y[8] = ... + f[-1]g[8+1] + f[0]g[8-0] + f[1]g[8-1] + f[2]g[8-2] + f[3]g[8-3] + f[4]g[8-4] + f[5]g[8-5] + ...
     = 1
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n    0 1 2 3 4
f[n] 1 2 3 2 1
g[n] 3 2 1 1 1



y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
y[7] = ... + f[-1]g[7+1] + f[0]g[7-0] + f[1]g[7-1] + f[2]g[7-2] + f[3]g[7-3] + f[4]g[7-4] + f[5]g[7-5] + ...
     = 2 + 1 = 3
y[8] = ... + f[-1]g[8+1] + f[0]g[8-0] + f[1]g[8-1] + f[2]g[8-2] + f[3]g[8-3] + f[4]g[8-4] + f[5]g[8-5] + ...
     = 1


f[n]*g[n] = [3, 8, 14, 15, 13, 9, 6, 3, 1]
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0  1  2  3  4
f[n]  1  2  3  2  1
g[n]  3  2  1  1  1

y[-1] = ... + f[-1]g[-1+1] + f[0]g[-1-0] + f[1]g[-1-1] + ...
      = 0
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
y[7] = ... + f[-1]g[7+1] + f[0]g[7-0] + f[1]g[7-1] + f[2]g[7-2] + f[3]g[7-3] + f[4]g[7-4] + f[5]g[7-5] + ...
     = 2 + 1 = 3
y[8] = ... + f[-1]g[8+1] + f[0]g[8-0] + f[1]g[8-1] + f[2]g[8-2] + f[3]g[8-3] + f[4]g[8-4] + f[5]g[8-5] + ...
     = 1
y[9] = ... + f[-1]g[9+1] + f[0]g[9-0] + f[1]g[9-1] + f[2]g[9-2] + f[3]g[9-3] + f[4]g[9-4] + f[5]g[9-5] + ...
     = 0

f[n]*g[n] = [3, 8, 14, 15, 13, 9, 6, 3, 1]
```

Discrete convolution

Example - direct approach using convolution sum

$$y[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

```
f = [1, 2, 3, 2, 1]
g = [3, 2, 1, 1, 1]

n     0  1  2  3  4
f[n]  1  2  3  2  1
g[n]  3  2  1  1  1

y[-1] = ... + f[-1]g[-1+1] + f[0]g[-1-0] + f[1]g[-1-1] + ...
      = 0
y[0] = ... + f[-1]g[0+1] + f[0]g[0-0] + f[1]g[0-1] + ...
     = 3
y[1] = ... + f[-1]g[1+1] + f[0]g[1-0] + f[1]g[1-1] + f[2]g[1-2] + ...
     = 2 + 6 = 8
y[2] = ... + f[-1]g[2+1] + f[0]g[2-0] + f[1]g[2-1] + f[2]g[2-2] + f[3]g[2-3] + ...
     = 1 + 4 + 9 = 14
y[3] = ... + f[-1]g[3+1] + f[0]g[3-0] + f[1]g[3-1] + f[2]g[3-2] + f[3]g[3-3] + f[4]g[3-4] + ...
     = 1 + 2 + 6 + 6 = 15
y[4] = ... + f[-1]g[4+1] + f[0]g[4-0] + f[1]g[4-1] + f[2]g[4-2] + f[3]g[4-3] + f[4]g[4-4] + f[5]g[4-5] + ...
     = 1 + 2 + 3 + 4 + 3 = 13
y[5] = ... + f[-1]g[5+1] + f[0]g[5-0] + f[1]g[5-1] + f[2]g[5-2] + f[3]g[5-3] + f[4]g[5-4] + f[5]g[5-5] + ...
     = 2 + 3 + 2 + 2 = 9
y[6] = ... + f[-1]g[6+1] + f[0]g[6-0] + f[1]g[6-1] + f[2]g[6-2] + f[3]g[6-3] + f[4]g[6-4] + f[5]g[6-5] + ...
     = 3 + 2 + 1 = 6
y[7] = ... + f[-1]g[7+1] + f[0]g[7-0] + f[1]g[7-1] + f[2]g[7-2] + f[3]g[7-3] + f[4]g[7-4] + f[5]g[7-5] + ...
     = 2 + 1 = 3
y[8] = ... + f[-1]g[8+1] + f[0]g[8-0] + f[1]g[8-1] + f[2]g[8-2] + f[3]g[8-3] + f[4]g[8-4] + f[5]g[8-5] + ...
     = 1
y[9] = ... + f[-1]g[9+1] + f[0]g[9-0] + f[1]g[9-1] + f[2]g[9-2] + f[3]g[9-3] + f[4]g[9-4] + f[5]g[9-5] + ...
     = 0

f[n]*g[n] = [3, 8, 14, 15, 13, 9, 6, 3, 1]
```

Multidimensional discrete convolution

Multidimensional discrete convolution

Definition

The convolution of two complex-valued functions on $\mathbf{R}^n$ is a complex-valued function on $\mathbf{R}^n$, defined by:

$$(f * g)(t) = \int_{\mathbf{R}^n} f(\tau)g(t-\tau)\,d\tau = \int_{\mathbf{R}^n} f(t-\tau)g(\tau)\,d\tau.$$

An n-dimensional convolution would be written as:

$$y(t_1, t_2, \ldots, t_n) = f(t_1, t_2, \ldots, t_n) * \overset{n}{\cdots} * g(t_1, t_2, \ldots, t_n)$$

Similar to the one-dimensional case, an asterisk is used to represent the convolution operation. The number of dimensions in the given operation is reflected in the number of asterisks.

You can consider *multidimensional discrete convolution* which is directly computed via the following formula:

$$\sum_{\tau_1=-\infty}^{\infty} \sum_{\tau_2=-\infty}^{\infty} \ldots \sum_{\tau_n=-\infty}^{\infty} f(\tau_1, \tau_2, \ldots, \tau_n) g(t_1-\tau_1, t_2-\tau_2, \ldots, t_n-\tau_n)$$

Two-dimensional convolution is given by:

$$(f * g)[n_1][n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} f[k_1][k_2] g[n_1-k_1][n_2-k_2]$$

which is quite similar to one-dimensional case mentioned earlier:

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k].$$

Two-dimensional discrete convolution

Example - direct approach using convolution sum

Let's try to compute the pixel value of the output data resulting from the convolution of 5×5 sized data matrix f with the kernel g of size 3×3, shown below:

$$
f = \begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 \\
1 & 6 & 7 & 8 & 9 & 9 \\
2 & 8 & 7 & 6 & 5 & 4 \\
3 & 3 & 2 & 1 & 1 & 2 \\
4 & 3 & 4 & 5 & 6 & 7
\end{array}
\qquad
g = \begin{array}{c|ccc}
 & 0 & 1 & 2 \\
\hline
0 & 1 & 2 & 3 \\
1 & 4 & 5 & 6 \\
2 & 7 & 8 & 9
\end{array}
$$

Two-dimensional discrete convolution

Example - direct approach using convolution sum

$$(f * g)[n_1][n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} f[k_1][k_2] g[n_1 - k_1][n_2 - k_2]$$

```
     0   1   2   3   4                        0   1   2   3   4
     0 [1,  2,  3,  4,  5]          0   1   2   0   1
     1 [6,  7,  8,  9,  9]        0 [1,  2,  3]  1
f =  2 [8,  7,  6,  5,  4]  g = 1 [4,  5,  6]  2
     3 [3,  2,  1,  1,  2]        2 [7,  8,  9]  3
     4 [3,  4,  5,  6,  7]                      4


y[0][0] =
... + f[-1][-1]g[0+1][0+1] + f[-1][0]g[0+1][0-0] + f[-1][1]g[0+1][0-1] + ...
... + f[ 0][-1]g[0-0][0+1] + f[ 0][0]g[0-0][0-0] + f[ 0][1]g[0-0][0-1] + ...
... + f[ 1][-1]g[0-1][0+1] + f[ 1][0]g[0-1][0-0] + f[ 1][1]g[0-1][0-1] + ...
=
... + f[-1][-1]g[0+1][0+1] + f[-1][0]g[0+1][0-0] + f[-1][1]g[0+1][0-1] + ...
... + f[ 0][-1]g[0-0][0+1] + f[ 0][0]g[0-0][0-0] + f[ 0][1]g[0-0][0-1] + ...
... + f[ 1][-1]g[0-1][0+1] + f[ 1][0]g[0-1][0-0] + f[ 1][1]g[0-1][0-1] + ...
=
f[ 0][0]g[0-0][0-0]
=
1*1
=
1
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Instead of using direct approach and computing convolution sum from formula, you can apply "mechanical" or "manual" procedure similar to the inversion and shift method applied in one-dimensional case.

To do this, first you have to find kernel matrix inversion. You do this exactly the same way as you do it for one-dimensional case: you flip the kernel along, rows followed by a flip along columns (order is not important):

```
     initial          flip            flip
                     by row         by column

    [1, 2, 3]      [7, 8, 9]      [9, 8, 7]
g = [4, 5, 6]  g = [4, 5, 6]  g = [6, 5, 4]
    [7, 8, 9]      [1, 2, 3]      [3, 2, 1]
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
        1   2   3   4   5
        6   7   8   9   9             9   8   7
f  =    8   7   6   5   4    g  =     6   5   4
        3   2   1   1   2             3   2   1
        3   4   5   6   7
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
9   8   7
6   5   4
3   2  11  2   3   4   5
       6   7   8   9   9
  f =  8   7   6   5   4
       3   2   1   1   2
       3   4   5   6   7

       1   ?   ?   ?   ?   ?   ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
  9   8   7
  6   5   4
  3  12  21  3   4   5
     6   7   8   9   9
f  = 8   7   6   5   4
     3   2   1   1   2
     3   4   5   6   7

     1   4   ?   ?   ?   ?   ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
      9   8   7
      6   5   4
    1 3 2 2 3 1 4   5
    6   7   8   9   9
f = 8   7   6   5   4
    3   2   1   1   2
    3   4   5   6   7

    1   4   10  ?   ?   ?   ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
        9  8  7
        6  5  4
    1  23 32 41 5
    6  7  8  9  9
f = 8  7  6  5  4
    3  2  1  1  2
    3  4  5  6  7

    1  4  10 16 ?  ?  ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
           9  8  7
           6  5  4
    1  2  33 42 51
    6  7  8  9  9
f = 8  7  6  5  4
    3  2  1  1  2
    3  4  5  6  7

    1  4  10 16 22 ?  ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
              9  8  7
              6  5  4
    1  2  3  43 52  1
    6  7  8  9  9
f = 8  7  6  5  4
    3  2  1  1  2
    3  4  5  6  7

    1  4  10 16 22 22 ?
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
                     9    8    7
                     6    5    4
    1   2   3   4   5 3    2    1
    6   7   8   9   9
f = 8   7   6   5   4
    3   2   1   1   2
    3   4   5   6   7

    1   4   10  16  22  22  15
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
9  8   7
6  5  14  2   3   4   5
3  2  61  7   8   9   9
 f = 8    7   6   5   4
      3   2   1   1   2
      3   4   5   6   7

      1   4   10  16  22  22  15
      10
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
  9  8  7
  6 15 24 3   4   5
  3 62 71 8   9   9
f = 8   7   6   5   4
    3   2   1   1   2
    3   4   5   6   7

    1   4   10  16  22  22  15
    10  32
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

... many steps ...

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
9   8  17  2   3   4   5
6   5  64  7   8   9   9
3f  2  81  7   6   5   4
       3   2   1   1   2
       3   4   5   6   7

       1   4   10  16  22  22  15
       10  32  ...
       39
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
  9 18 27 3  4  5
  6 65 74 8  9  9
f 3=82 71 6  5  4
    3  2  1  1  2
    3  4  5  6  7

    1  4  10 16 22 22 15
    10 32 ...
    39 103
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
    19  28  37  4   5
    66  75  84  9   9
f = 83  72  61  5   4
    3   2   1   1   2
    3   4   5   6   7

    1   4   10  16  22  22  15
    10  32  ...
    39  103193
```

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

... many steps ...

Two-dimensional discrete convolution

Example - mechanical approach using sliding inverted matrix

Now you can move inverted kennel matrix over data matrix

```
     1   2   3   4   5
     6   7   8   9   9
f =  8   7   6   5   4
     3   2   1   1   2
     3   4   5   6   7 9   8   7
                       6   5   4
                       3   2   1
```

```
  1,  4, 10, 16, 22, 22, 15
 10, 32, 68, 89,109, 94, 57
 39,103,193,226,255,198,111
 77,173,287,291,291,210,111
 71,146,224,195,175,125, 69
 33, 69,108,106,119, 96, 60
 21, 52, 94,118,142,110, 63
```

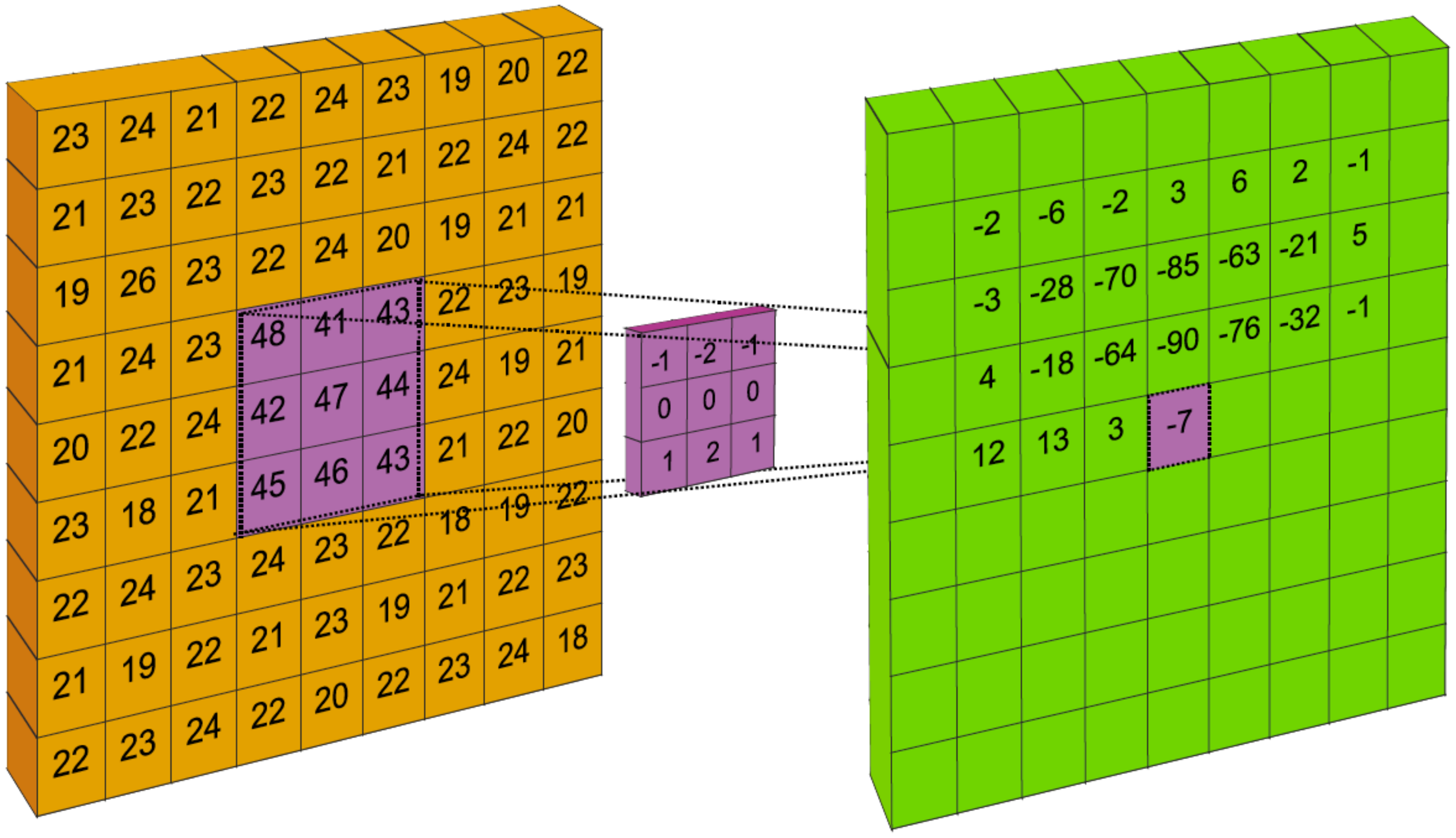
Group operators

Group operators

Very general definition

Group operations calculate new pixel values from a pixel’s neighbourhood by using a various "grouping" process.

The group operation is usually expressed in terms of *template convolution* where the template is a set of weighting coefficients.



Convolution

Practical approach

- The template is usually square.
- Its size is usually odd (for example 3×3 or 5×5) to ensure that the result positioned precisely on a pixel.
- For reasons of speed, the most common sizes are 3×3, 5×5 and 7×7.
- This template is used to calculate new pixel value for every pixel.
- New pixel values are calculated by placing the template (its central point) at the point of interest.
    - Source image pixel values are multiplied by the corresponding weighting coefficient and added to an overall sum. The sum (usually) evaluates a new value for the centre pixel (where the template is centered), and this becomes the pixel in the new image, the output image.

Convolution

Formula

Template convolution of input (source) image $p(x, y)$ and weight template $w(i, j)$ is given by the following formula:

$$q(x, y) = \sum_{\substack{i = 1,\ldots,W \\ j = 1,\ldots,H}} w(i, j) \cdot p(x(i), y(j))$$

where:

- w and h are the numbers of columns (width) and rows (height),
- $x \in \{1,\ldots,w\}$ and $y \in \{1,\ldots,h\}$,
- $w(i, j)$ is a weighting coefficient at (i, j) point of weight template,
- W is a width and H is a height of a template,
- $x(i)$ and $y(j)$ are the coordinates of image points laying within the template area "centered" at (x, y).

Convolution

Example

Pixel for which you make calculations (image reference point)

Input image:

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 1 & 2 \end{array}$$

Weights:

$$\begin{array}{ccc} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{array}$$

Template center point (template reference point)

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 9\;6 & 8\;7 & 7\;8 & 9 & 1 \\ 6\;2 & 5\;3 & 4\;4 & 5 & 6 \\ 3\;7 & 2\;8 & 1\;9 & 1 & 2 \end{array}$$

$$w = 5,\ h = 4,\ W = 3,\ H = 3,\ x = 3,\ y = 2$$

$$q(3,2) = \sum_{\substack{i=1,\ldots,3 \\ j=1,\ldots,3}} w(i,j)\cdot p(x(i),y(j))$$

$$\begin{aligned}
&= w(1,1)\cdot p(x(1),y(1)) + w(2,1)\cdot p(x(2),y(1)) + w(3,1)\cdot p(x(3),y(1)) \\
&\ + w(1,2)\cdot p(x(1),y(2)) + w(2,2)\cdot p(x(2),y(2)) + w(3,2)\cdot p(x(3),y(2)) \\
&\ + w(1,3)\cdot p(x(1),y(3)) + w(2,3)\cdot p(x(2),y(3)) + w(3,3)\cdot p(x(3),y(3)) \\
&= w(1,1)\cdot p(1,2) + w(2,1)\cdot p(2,2) + w(3,1)\cdot p(3,2) \\
&\ + w(1,2)\cdot p(1,3) + w(2,2)\cdot p(2,3) + w(3,2)\cdot p(3,3) \\
&\ + w(1,3)\cdot p(1,4) + w(2,3)\cdot p(2,4) + w(3,3)\cdot p(3,4) \\
&= 9\cdot 6 + 8\cdot 7 + 7\cdot 8 \\
&\ + 6\cdot 2 + 5\cdot 3 + 4\cdot 4 \\
&\ + 3\cdot 7 + 2\cdot 8 + 1\cdot 9
\end{aligned}$$

Convolution

Problem with borders

Note that we cannot ascribe values to the picture's borders.

To calculate values for the border pixels, we have three choices, but please keep in mind that none of them is optimal:

- Set the border to black (or deliver a smaller picture).
- Assume (as in Fourier) that the image replicates to infinity along both dimensions and calculate new values by cyclic shift from the far border.
- Calculate the border pixel value from a smaller area.

Convolution

Applications

In digital image processing convolutional filtering plays an important role in many algorithms such as edge detection, blurring and related processes.

Basic group methods

Averaging operator

Definition

For an averaging operator, the template weighting function is unity.

To avoid constraining you can use $1/(w \cdot h)$ as a weighting function - in this case the result of averaging pixels will not exceed upper range (255).

The averaging operator is then simply:

$$q(x,y) = \frac{1}{w \cdot h} \sum_{\substack{i=1,\ldots,W \\ j=1,\ldots,H}} p(x(i), y(j)).$$

Averaging operator

Examples

[tutu examples]

Averaging operator

Conclusions

The effect of averaging is to **reduce noise**, which is its advantage.

Disadvantage is that averaging **causes blurring** which **reduces details** in an image. It is also

The effect of larger averaging operators is to smooth the image more, to remove more detail whilst giving greater **emphasis to the large structures**.

Gaussian averaging operator

Definition

The template for the Gaussian operator has values set by the Gaussian relationship. The Gaussian function g at coordinates (x, y) is controlled by the variance σ^2 according to:

$$g(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

More generally we can consider general form of Gaussian function (here given for one variable):

$$f(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2c^2}\right),$$

where for a, b and c are arbitrary real constants (of course c must be non-zero).

In effect, the Gaussian function essentially removes the influence of points greater than 3σ in (radial) distance from the centre of the template.

The size of the template essentially dictates appropriate choice of the variance. The variance is chosen to ensure that template coefficients drop to near zero at the template's edge.

Gaussian averaging operator

Examples

[tutu examples and comparision with averaging operator]

Gaussian averaging operator

Conclusions

The Gaussian averaging operator has been considered to be optimal for image smoothing because more features are retained whilst the noise is removed.

Median filter

Definition

The **median** is the value separating the higher half from the lower half of a data sample, a population, or a probability distribution. For a data set, it may be thought of as "the middle" value.

The median filter is usually computed from a pixels taken from an area of input image laying within the template area centered on the point of interest. What is different in this case is that template has no weights - it is used only to delimit the area of interest. That is why you should **think about it rather as a window** through which you observe image data.

Consider the typical square shape arrangement of template's pixels (but you can consider any other alternative shapes like cross, horizontal or vertical line as well). To get a value of resulting pixel $q(x, y)$ you have to:

1. Collect all pixels constituting image area within template "centered" at (x, y) and put them into a list (or vector).
2. Sort list obtained in previous step.
3. The central component of the sorted list is the median value - this value is returned as an effect of applying median filter at point (x, y) of source image $p(x, y)$.

Median filter

Examples

[tutu examples]

Median filter

Conclusions

- The median filter has a well-known **ability to remove salt and pepper noise**.
  - This form of noise is typical for decoding-like errors in picture transmission systems and appears as isolated white and black points within an image.
  - It can also arise when rotating an image, when points remain unspecified by a standard rotation operator.

  When a median operator is applied, the salt and pepper noise points will appear at either end of the rank-ordered list and are removed by the median process.
- The median operator has practical advantage, due to its ability to retain edges (the boundaries of shapes in images) whilst suppressing the noise contamination.

Mode filter

Definition

The mode (pl. *dominanta*) is the value that appears most often in a set of data values.

For small set of data the correct mode is very difficult to determine. For example, it's not hard to imagine (it is highly probable) that within a 5×5 square template all 25 pixels are different, so each could be considered to be the mode.

As such you are **forced to estimate the mode** with the *truncated median filter*.

Mode filter

Truncated median filter - idea

The truncated median filter is based on the premise that for many non-Gaussian distributions, the order of the mean, the median and the mode is the same for many images, as illustrated below:

[img_02_02]

Accordingly, if you truncate the distribution, which means remove part of it, where the part selected to be removed is from the region beyond the mean, then the median of the truncated distribution will approach the mode of the original distribution:

[img_02_03]

Mode filter

Truncated median filter - algorithm

1. In implementation the operator first finds the *mean* and the *median* of the *current window*.

2. The distribution of intensity of points within the current window is truncated on the side of the mean so that the *median* now bisects the distribution of the remaining points.

   So that the median bisects the remaining distribution:

   A. If the ***median*** is **less than** the ***mean***, the point at which the distribution is truncated, **upper**, is:

   $$upper = 2 \cdot median - \min(distribution)$$

   B. If the ***median*** is **greater** than the ***mean***, then you need to truncate at a **lower** point (before the mean), given by:

   $$lower = 2 \cdot median - \max(distribution)$$

3. The median of the truncated vector is the output of the truncated median filter at that point. However, there can be several iterations at each position to ensure that the mode is approached.

Notes:

- In practice only few iterations are usually required for the median to converge to the mode.

- The window size is usually large, say 7×7, 9×9 or even more.

Mode filter

Truncated median filter - algorithm

Let the current window, of the size 5×5, contains the following values:

```
[50, 50, 230, 100, 70, 150, 70, 70, 150, 180, 70, 100, 70, 100,
100, 70, 100, 50, 70, 150, 70, 150, 180, 70, 50]
```

You can sort them:

```
[50, 50, 50, 50, 70, 70, 70, 70, 70, 70, 70, 70, 70, 100, 100,
100, 100, 100, 150, 150, 150, 150, 180, 180, 230]
```

and compute histogram:

```
[50 (4), 70 (9), 100 (5), 150 (4), 180 (2), 230 (1)]
```

1. In implementation the operator first finds the *mean* and the *median* of the *current window*:

   mean: 100.8

   median: 70

Note:
In this case it is possible to find mode directly and it is equal to 70 (you calculate it to verify if proposed algorithm works). Generally you want to approximate this value.

Mode filter

Truncated median filter - algorithm

A. If the ***median*** is **less than** the ***mean***, the point at which the distribution is truncated, **upper**, is:

$$\begin{aligned} upper &= 2 \cdot median - \min(distribution) \\ &= 2 \cdot 70 - 50 \\ &= 90 \end{aligned}$$

So the vector should be truncated at 90 resulting truncated distribution of the form:

```
[50, 50, 230, 100, 70, 150, 70, 70, 150, 180, 70, 100, 70,
100, 100, 70, 100, 50, 70, 150, 70, 150, 180, 70, 50]
```

You can sort them:

```
[50, 50, 50, 50, 70, 70, 70, 70, 70, 70, 70, 70, 70, 100,
100, 100, 100, 100, 150, 150, 150, 150, 180, 180, 230]
```

and compute histogram:

[50 (4), 70 (9), 100 (5), 150 (4), 180 (2), 230 (1)]

Mode filter

Truncated median filter - algorithm

Truncated vector:

```
[50, 50, 50, 50, 70, 70, 70, 70, 70, 70, 70,
70]
```

3. The median of the truncated vector is the output of the truncated median filter at that point:

   ```
   [50, 50, 50, 50, 70, 70, 70, 70, 70, 70,
   70, 70, 70]
   ```

   median: 70.

   70 is an approximate value of mode (which is in accordance with previous calculations).

Mode filter

Examples

[tutu examples]

Mode filter

Conclusions

- This has an ability to reduce noise whilst retaining feature boundaries.

Bibliography

Bibliography

1. Convolution, Visual explanation, `https://en.wikipedia.org/wiki/Convolution#Visual_explanation`

2. 2D Convolution in Image Processing, `https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/`

3. Convolutions with OpenCV and Python, `https://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/`

4. Image Filtering Using Convolution in OpenCV, `https://learnopencv.com/image-filtering-using-convolution-in-opencv/`

5. Basics of Kernels and Convolutions with OpenCV, `https://towardsdatascience.com/basics-of-kernels-and-convolutions-with-opencv-c15311ab8f55`

6. Convolution calculator, `https://www.rapidtables.com/calc/math/convolution-calculator.html`

7. Online Multidimensional Convolution Calculator, `https://leventozturk.com/engineering/convolution/`