CS-465 Computer Vision

Nazar Khan

PUCIT

3. Image Filtering

Convolution

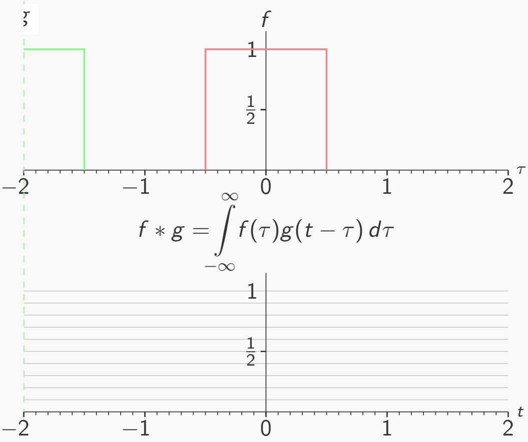
- For two functions $f(\tau)$ and $g(\tau)$ defined over $\mathbb{R}$, *continuous convolution* is defined as

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

- For two functions $f[m]$ and $g[m]$ defined over $\mathbb{Z}$, *discrete convolution* is defined as

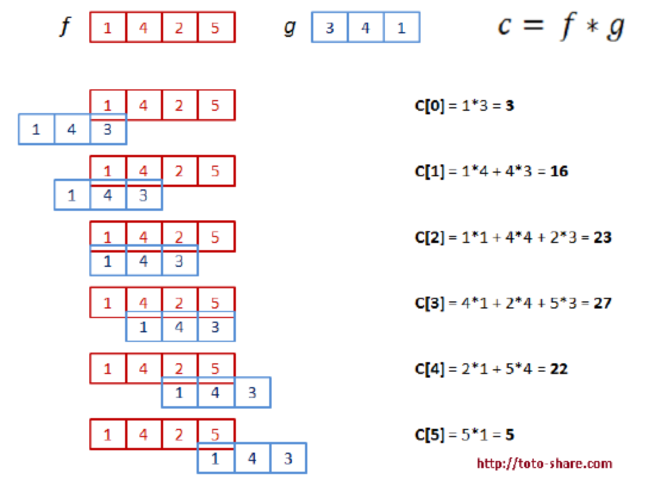
$$(f * g)[n] := \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$
$$= \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

1D Convolution
Example Continuous



Source: `http://www.texample.net/tikz/examples/convolution-of-two-functions/`

1D Convolution
Example Discrete

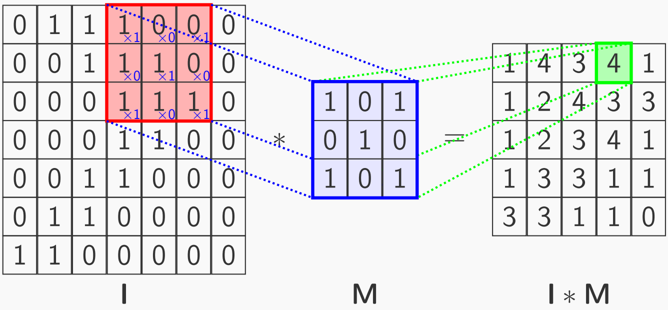


2D Convolution

- Integral/sum of the product of two functions after one is reversed and shifted.
- Central role in image processing.
- For 2D images I and M, convolution is defined as

$$
\begin{aligned}
(I * M)[i,j] &:= \sum_{k\in\mathbb{Z}} \sum_{l\in\mathbb{Z}} I[k,l]M[i-k,j-l] \\
&= \sum_{k\in\mathbb{Z}} \sum_{l\in\mathbb{Z}} I[i-k,j-l]M[k,l]
\end{aligned}
$$

2D Convolution
Example



Modified from `https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution`

M is usually called a *mask* or *kernel*.

2D Convolution

Applying a mask to an image

- In order to obtain $I_2 = I_1 * M$
  1. First flip the mask M in both dimensions.
  2. For each pixel p in I_1
     - 2.1 Place *mask origin* on top of the pixel.
     - 2.2 Multiply each mask weight with the pixel value under it.
     - 2.3 Sum the result and put in location of the pixel p in I_2.
- Any pixel can be defined as the origin of the mask. Usually it is the central pixel.

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

Dealing with boundaries

- What about edge and corner pixels where the mask goes outside the image boundaries?
  - Expand image I_1 with virtual pixels. Options are:
    1. Fill with a particular value, *e.g.* zeros.
    2. Replicating boundaries: fill with nearest pixel value.
    3. Reflecting boundaries: mirror the boundary
  - Fatalism: just ignore them. Not recommended since size of I_2 will shrink.

Dealing with boundaries
Expand by zeros

For a 5×5 image and 5×5 mask

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & d & e & 0 & 0 \\ 0 & 0 & f & g & h & i & j & 0 & 0 \\ 0 & 0 & k & l & m & n & o & 0 & 0 \\ 0 & 0 & p & q & r & s & t & 0 & 0 \\ 0 & 0 & u & v & w & x & y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Dealing with boundaries
Replicating boundaries

For a 5×5 image and 5×5 mask

$$\begin{bmatrix} a & a & a & b & c & d & e & e & e \\ a & a & a & b & c & d & e & e & e \\ a & a & \color{blue}{a} & \color{blue}{b} & \color{blue}{c} & \color{blue}{d} & \color{blue}{e} & e & e \\ f & f & \color{blue}{f} & \color{blue}{g} & \color{blue}{h} & \color{blue}{i} & \color{blue}{j} & j & j \\ k & k & \color{blue}{k} & \color{blue}{l} & \color{blue}{m} & \color{blue}{n} & \color{blue}{o} & o & o \\ p & p & \color{blue}{p} & \color{blue}{q} & \color{blue}{r} & \color{blue}{s} & \color{blue}{t} & t & t \\ u & u & \color{blue}{u} & \color{blue}{v} & \color{blue}{w} & \color{blue}{x} & \color{blue}{y} & y & y \\ u & u & u & v & w & x & y & y & y \\ u & u & u & v & w & x & y & y & y \end{bmatrix}$$

Dealing with boundaries
Reflecting boundaries

For a 5×5 image and 5×5 mask

$$\begin{bmatrix} m & l & k & l & m & n & o & n & m \\ h & g & f & g & h & i & j & i & h \\ c & b & \color{blue}{a} & \color{blue}{b} & \color{blue}{c} & \color{blue}{d} & \color{blue}{e} & d & c \\ h & g & \color{blue}{f} & \color{blue}{g} & \color{blue}{h} & \color{blue}{i} & \color{blue}{j} & i & h \\ m & l & \color{blue}{k} & \color{blue}{l} & \color{blue}{m} & \color{blue}{n} & \color{blue}{o} & n & m \\ r & q & \color{blue}{p} & \color{blue}{q} & \color{blue}{r} & \color{blue}{s} & \color{blue}{t} & s & r \\ w & v & \color{blue}{u} & \color{blue}{v} & \color{blue}{w} & \color{blue}{x} & \color{blue}{y} & x & w \\ r & q & p & q & r & s & t & s & r \\ m & l & k & l & m & n & o & n & m \end{bmatrix}$$

Convolution Masks

- Different masks lead to different effects.

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix} \quad \frac{1}{2}\begin{bmatrix}1 & 1 & 1\\0 & 0 & 0\\-1 & -1 & -1\end{bmatrix} \quad \frac{1}{18}\begin{bmatrix}2 & 3 & 2\\0 & 0 & 0\\-2 & -3 & -2\end{bmatrix}$$

Averaging y-Derivative y-Derivative & averaging

- Interactive demo at `http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo`
- Cost of convolving an $m \times n$ image with a $k \times k$ kernel is k^2 multiplications plus $k^2 - 1$ additions repeated for $m \times n$ locations. That is $mn(2k^2 - 1)$ operations.

Properties of Convolution

- Commutativity: $I * M = M * I$
  - Signal and kernel play an equal role.
- Associativity: $(I * M_1) * M_2 = I * (M_1 * M_2)$
  - Successive convolutions with kernels M_1 and M_2 is equivalent to a single convolution with kernel $M1 * M2$ *which is computationally much cheaper* since kernels are usually smaller than images.
- Shift Invariance: $\text{Translation}(I * M) = \text{Translation}(I) * M$
  - Translation of convolved signal is equivalent to convolution of translated signal.
- Linearity: $(aI + bJ) * M = a(I * M) + b(J * M)\ \forall\ a, b \in \mathbb{R}$
  - Single convolution of a linear combination of signals is equivalent to a linear combination of multiple convolutions.
  - Obviously, the single convolution is *computationally much cheaper*.

> Because of the last two properties, convolution is called *linear shift invariant (LSI)* filtering.

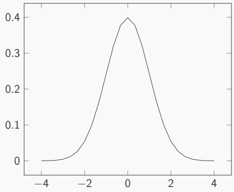
Gaussian Kernel

- A widely used mask for smoothing is the *Gaussian* kernel.

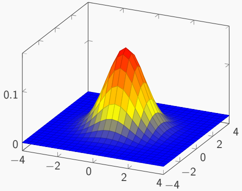
$$1D: \; g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$2D: \; G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x-\mu_1)^2 + (y-\mu_2)^2}{2\sigma^2}\right)$$

where μ is the 1D mean, (μ_1, μ_2) is the 2D mean and σ^2 is the variance.



$\mu = 0, \sigma = 1$



$(\mu_1, \mu_2) = (0,0), \sigma = 1$

Gaussian Kernel
Discrete approximation

$$1D: \begin{bmatrix} 0.006 & 0.061 & 0.242 & 0.343 & 0.242 & 0.061 & 0.006 \end{bmatrix}$$

$$2D: \begin{bmatrix} 0.006 & 0.061 & 0.242 & 0.343 & 0.242 & 0.061 & 0.006 \end{bmatrix} * \begin{bmatrix} 0.006 \\ 0.061 \\ 0.242 \\ 0.343 \\ 0.242 \\ 0.061 \\ 0.006 \end{bmatrix}$$

Separability of Gaussian Kernels: Convolution with 2D Gaussian can be performed via two successive convolutions with 1D Gaussians which are computationally much cheaper.

Gaussian Kernel

Discrete integer approximation

$$1D: \frac{1}{961}\begin{bmatrix}6 & 61 & 242 & 343 & 242 & 61 & 6\end{bmatrix}$$

$$2D: \frac{1}{961^2}\begin{bmatrix}6 & 61 & 242 & 343 & 242 & 61 & 6\end{bmatrix} * \begin{bmatrix}6\\61\\242\\343\\242\\61\\6\end{bmatrix}$$

For images stored in unsigned 8-bit integer format (uint8), integer operations are faster than floating point operations.

Python Examples

Convolution with 5×5 averaging mask

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('book.png')    #OpenCV uses BGR color order
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #matplotlib uses RGB

kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

Figure: Effect of convolving with averaging masks of increasing size.
Author: N. Khan (2018)

Python Examples
Other filters

- Kernel in line 8 can be defined according to your needs.
- Line 8 can be commented and line 9 can be replaced by
  - dst = cv2.blur(img,(5,5)) for the same effect
  - dst = cv2.GaussianBlur(img,(5,5),0) for Gaussian smoothing
  - dst = cv2.medianBlur(img,5) for median filtering
  - dst = cv2.bilateralFilter(img,9,75,75) for edge preserving smoothing
- Any filtering performed via convolution is *linear filtering*.
- *Non-linear filtering* yields additional benefits.
  - Median filtering
  - Bilateral filtering
  - Non-local means

Figure: Effect of convolving with Gaussian masks of increasing size.
Author: N. Khan (2018)

Figure: Effect of median filtering with masks of increasing size. Author: N. Khan (2018)

Figure: Comparison of 35×35 Gaussian smoothing with bilateral filter of diameter 35. Notice how bilateral filtering preserves edges. Author: N. Khan (2018)