Image Smoothing

김수환

https://www.soohwan.kim

학습목표

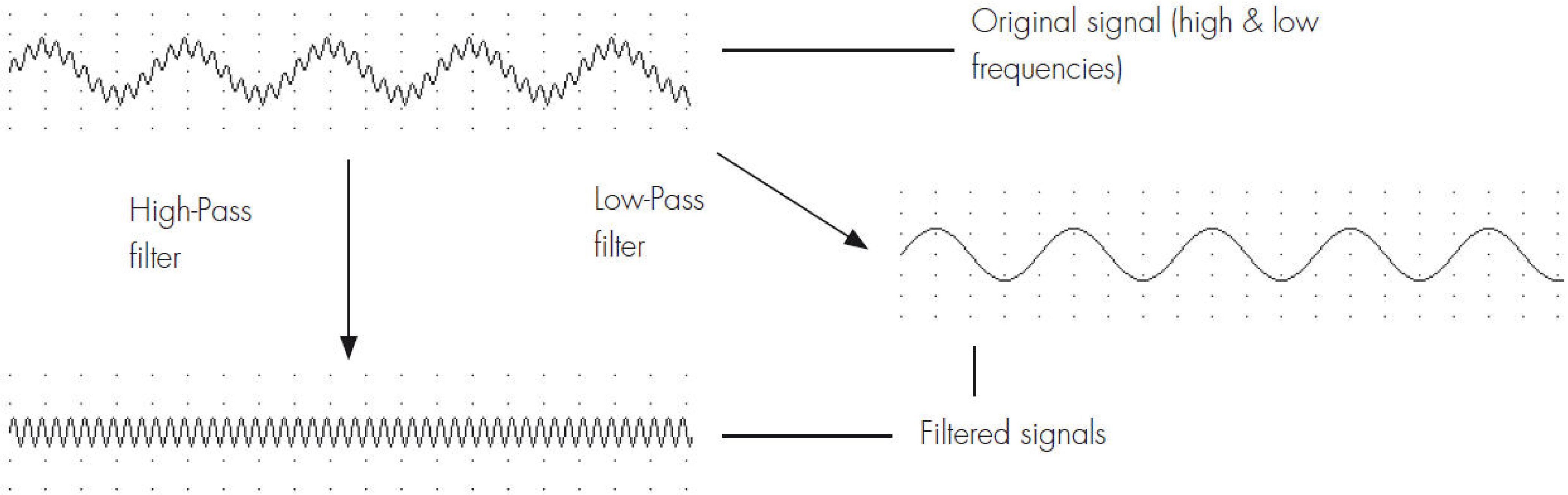
학습목표

1. Image Processing
    1. Image Thresholding
    2. Image Blending
    3. Image Filtering

Image Filtering

Signal Processing

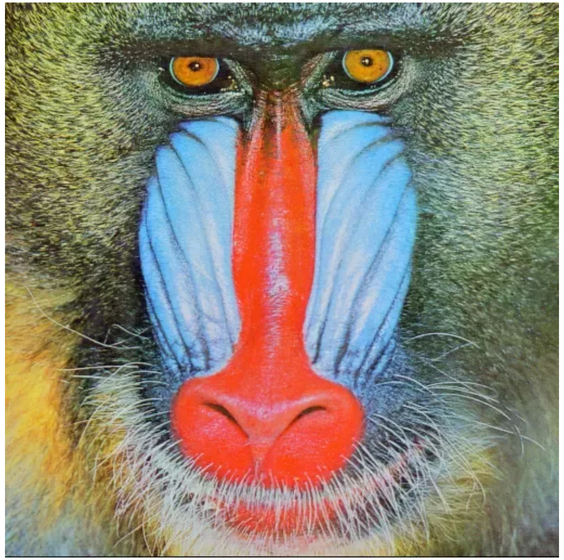
1-D Signals: LPF vs. HPF (Music: Bass vs. Treble)



https://www.edaq.com/wiki/Basics_of_Data_Acquisition

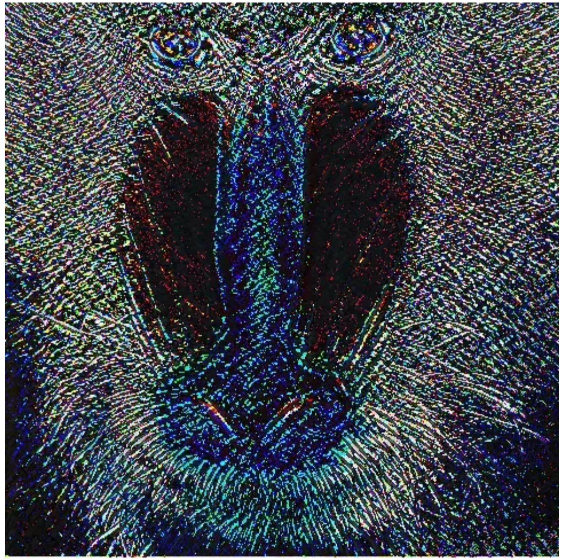
Image Processing

2-D Signals: LPF vs. HPF



Original

Low-Pass Filter



High-Pass Filter

- Low-Pass Filters: noise를 없애준다. image를 blurring 시킨다.
- High-Pass Filters: image에서 edge를 찾아준다.

https://www.quora.com/What-is-meant-by-low-pass-and-high-pass-filters

Convolutions

- Definition: a weighted average of a function $f(\tau)$ at the moment t where the weighting is given by $g(-\tau)$ simply shifted by amount t

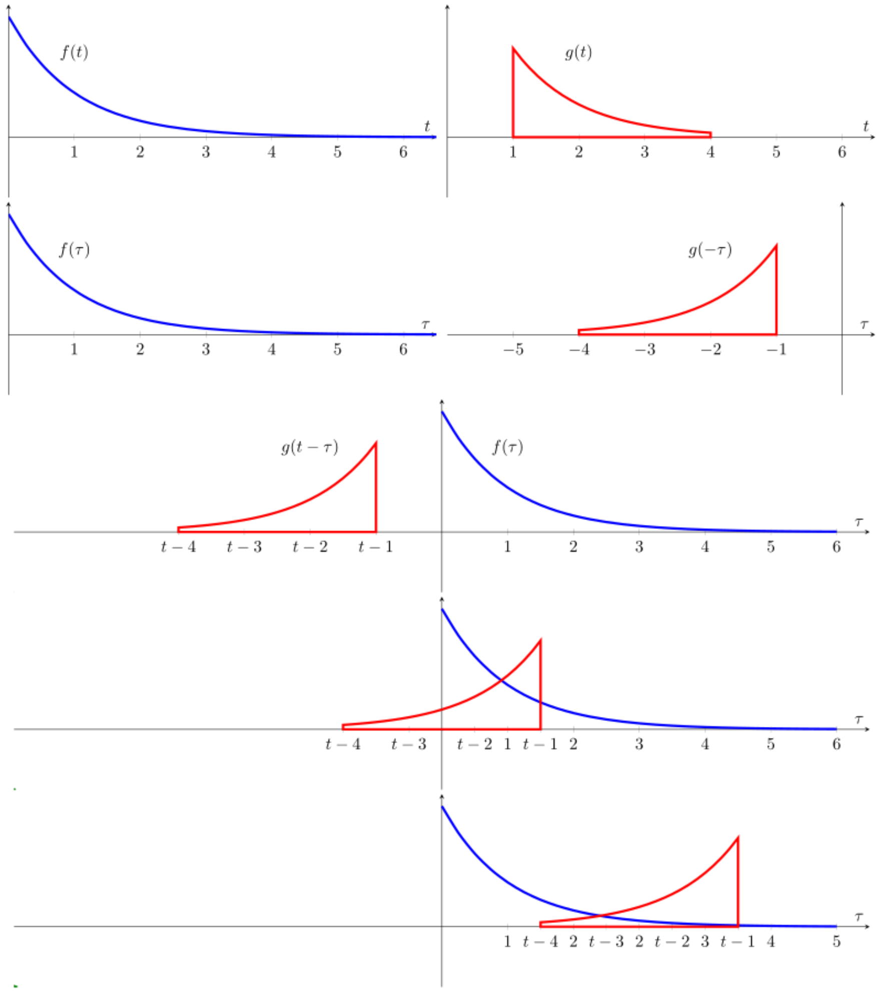
$$f(t) * g(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

- Commutative

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

https://en.wikipedia.org/wiki/Convolution

Convolutions



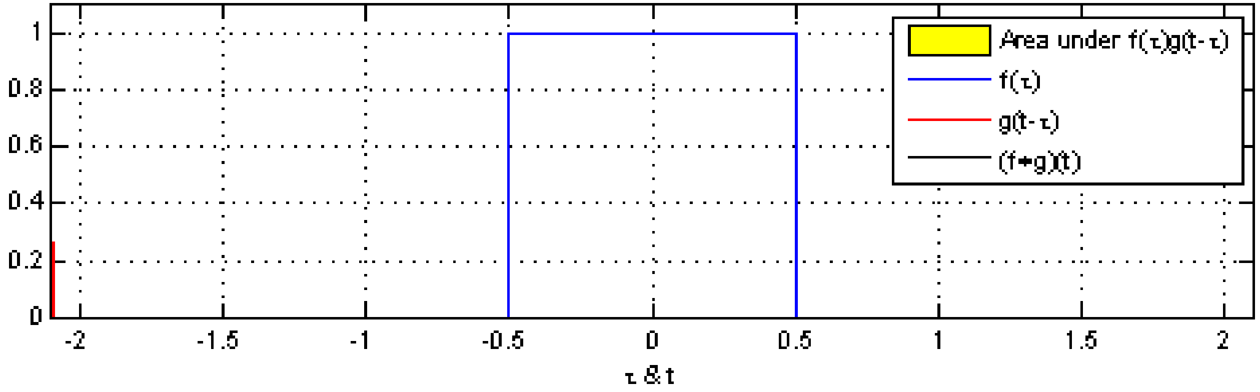
- Convolution

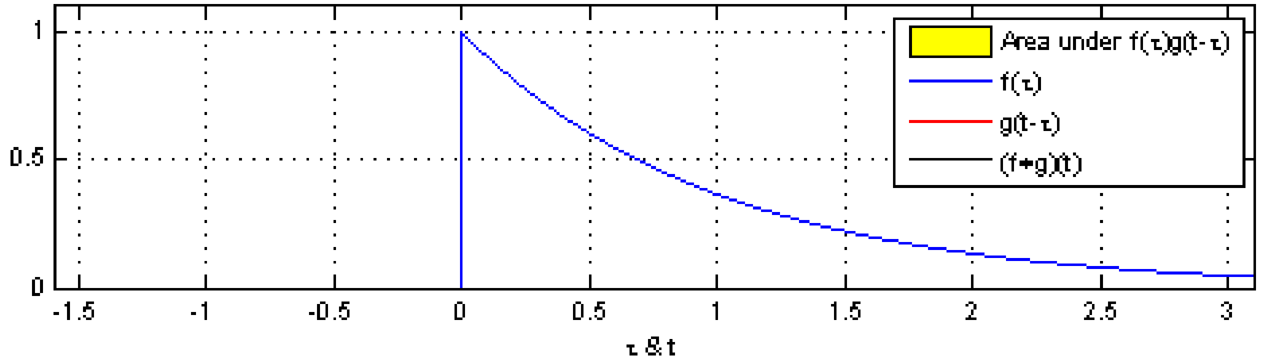
$$f(t) * g(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

1. 각각의 함수를 dummy variable τ로 표현한다.
2. 함수 g를 y축에 대해 대칭시킨다: $g(\tau) \rightarrow g(-\tau)$
3. $(f * g)(t)$: convolution at t
    1. 함수 g를 x축으로 t만큼 평행이동 시킨다.
    2. τ를 $-\infty$에서 ∞까지 함수 f와 g를 곱해서 더한다.
4. 즉, weight $g(-\tau)$를 x축으로 t만큼 sliding 시키면서 함수 $f(\tau)$와의 weighted-sum을 구한다.

https://en.wikipedia.org/wiki/Convolution

Convolutions: Examples





https://en.wikipedia.org/wiki/Convolution

2D Convolutions: Image Filtering

- Average Filter: e.g.) 5x5 Kernel

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- 각각의 픽셀에 대해서 window의 중심이 해당 픽셀에 오도록 위치한다.
- window안의 모든 픽셀들의 값을 더한뒤 윈도우 내부 총 픽셀의 개수로 나눈다.
- window를 sliding시키면서 모든 픽셀에 대해서 차례대로 윈도우 내부의 평균을 구한다.

Image Filtering: Result

Image Filtering: Code

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

# Load an image
img = cv2.imread('opencv_logo.jpg')

# Kernel
kernel = np.ones((5, 5), np.float32)/25

# Image Filtering
res = cv2.filter2D(img, -1, kernel)

# Display result
plt.subplot(121)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(122)
plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
```

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering

`cv.filter2D()`[1]

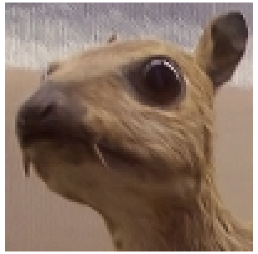
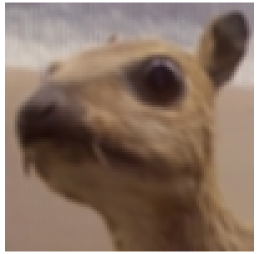
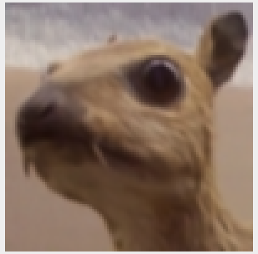
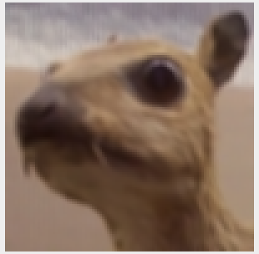
```
dst = cv.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])
```

- Image와 kernel을 convolution 한다. (실제로 kernel이 flip된 correlation을 구한다.)
  - `src`: input image)
  - `dst`: output array of the same size and the same number of channels as `src`
  - `ddepth`: desired depth of the destination image (-1 to use `src.depth()`)
  - `kernel`: convolution kernel, a single-channel floating point matrix; If you want to apply different kernels to different channels, split the image into separate color planes using split and process them individually
  - `anchor`: anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1, -1) means that the anchor is at the kernel center
  - `delta`: optional value added to the filtered pixels before storing them in `dst`
  - `borderType`: pixel extrapolation method

$$\mathrm{dst}(x, y) = \sum_{\substack{0 \leq x' < kernel.cols \\ 0 \leq y' < kernel.rows}} \mathrm{kernel}(x', y') * \mathrm{src}(x + x' - anchor.x, y + y' - anchor.y)$$

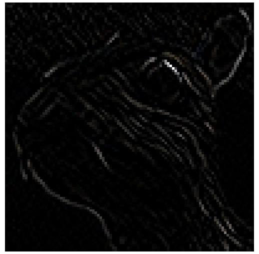
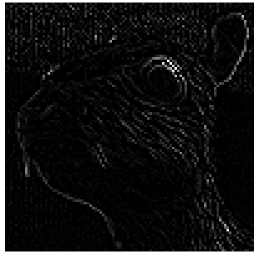
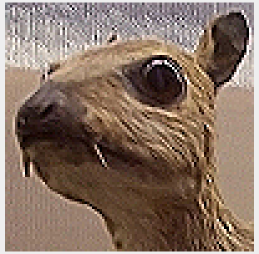
1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04

Image Filtering: Examples (Low-Pass Filters)

Operation	Kernel	Convolution	Operation	Kernel	Convolution
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		Box Blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian Blur 3x3 (approx.)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$		Gaussian Blur 5x5 (approx.)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

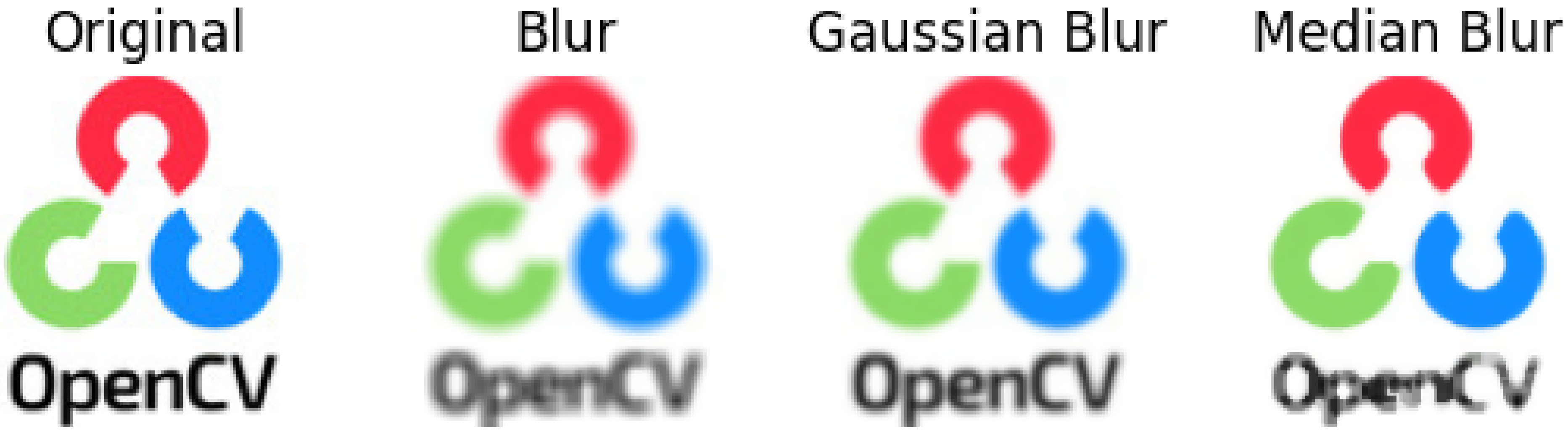
Image Filtering: Examples (High-Pass Filters)

Operation	Kernel	Convolution	Operation	Kernel	Convolution
Edge Detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		Edge Detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Edge Detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & 1 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Image Blurring/Smoothing

Image Blurring: Results



https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

Image Blurring: Code

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

# Load an image
img = cv2.imread('opencv_logo.jpg')

# Blurring
res_blur = cv2.blur(img, (5, 5))

# Gaussian Blurring
res_gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Median blurring
res_median_blur = cv2.medianBlur(img, 5)

# Display results
titles = ['Original', 'Blur', 'Gaussian Blur', 'Median Blur']
images = [img, res_blur, res_gaussian_blur, res_median_blur]

for i in range(4):
```

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

`cv.blur()`[1]

```
dst = cv.blur(src, ksize[, dst[, anchor[, borderType]]])
```

- Normalized box filter를 이용하여 이미지를 blur시킨다.
  - `src`: input image; it can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`
  - `dst`: output array of the same same and type as `src`
  - `ksize`: blurring kernel size
  - `anchor`: anchor point; default value (-1, -1) means that the anchor is at the kernel center
  - `borderType`: border mode used to extrapolate pixels outside of the image

$$K = \frac{1}{\text{ksize.width} * \text{ksize.height}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37

`cv.boxFilter()`[1]

```
dst = cv.boxFilter(src, ddepth, ksize[, dst[, anchor[, normalize[, borderType]]]])
```

- Unnormalized box filter를 이용하여 이미지를 blur시킨다.
  - `src`: input image
  - `dst`: output array of the same same and type as `src`
  - `ddepth`: output image depth (-1 to use `src.depth()`)
  - `ksize`: blurring kernel size
  - `anchor`: anchor point; default value (-1, -1) means that the anchor is at the kernel center
  - `normalize`: flag, specifying whether the kernel is normalized by its area or not
  - `borderType`: border mode used to extrapolate pixels outside of the image

$$K = \alpha \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \quad \text{where } \alpha = \begin{cases} \frac{1}{\text{ksize.width} * \text{ksize.height}}, & \text{when normalize=true} \\ 1, & \text{otherwise} \end{cases}$$

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37

`cv.GaussianBlur()`[1]

```
dst = cv.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
```

- Gaussian filter를 이용하여 이미지를 blur시킨다.
    - `src`: input image; it can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`
    - `dst`: output array of the same same and type as `src`
    - `ksize`: blurring kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma
    - `sigmaX`: Gaussian kernel standard deviation in X direction
    - `sigmaY`: Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height`, respectively
    - `borderType`: border mode used to extrapolate pixels outside of the image

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

`cv.medianBlur()`[1]

```
dst = cv.medianBlur(src, ksize[, dst])
```

- Median filter를 이용하여 이미지를 blur시킨다.
  - `src`: input 1-, 3-, or 4-channel image; when `ksize` is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`
  - `dst`: output array of the same same and type as `src`
  - `ksize`: aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7, ...
- Salt-and-pepper noise를 제거하는데 탁월하다.
- Box 혹은 Gaussian filter는 필터링 된 결과가 original image에 없는 값일 수 있지만 median filtering은 항상 어떤 픽셀의 값과 같다.

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

🔪 Bilateral Filtering

Bilateral Filtering (양방향 필터)

- Gaussian filter는 noise를 제거하는데 효과적이지만 edge가 blur 된다.

- Bilateral filter는 공간뿐만 아니라 intensity 차이에 대해서도 filtering을 한다.
- 하나의 Gaussian filter를 적용하고, 또 하나의 Gaussian filter를 주변 pixel의 intensity까지 고려해서 적용하는 방식이다.
- Bilateral filter는 edge를 보호하면서 noise를 제거하는데 효과적이다.
- Operation이 다른 filter에 비해서 오래걸린다.

$$\mathrm{dst} = \frac{1}{w} \sum_{q \in S} G_{\sigma_s}\left(\|p-q\|\right) G_{\sigma_r}\left(|I_p - I_q|\right) I_q$$

`cv.bilateralFilter()`[1]

```
dst = cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])
```

- Bilateral filter를 이미지에 적용시킨다.
    - `src`: source 8-bit or floating-point, 1-channel or 3-channel image`
    - `dst`: destination image of the same size and type as `src`
    - `d`: diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace
    - `sigmaColor`: filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood will be mixed together, resulting in larger areas of semi-equal color.
    - `sigmaSpace`: filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough. When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, `d` is proportional to `sigmaSpace`.
    - `borderType`: border mode used to extrapolate pixels outside of the image

1. https://docs.opencv.org/4.4.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

Bilateral Filter: Code

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

# Load an image
img = cv2.imread('taj.jpg')

# Blurring
res_blur = cv2.blur(img, (5, 5))

# Gaussian Blurring
res_gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Median blurring
res_median_blur = cv2.medianBlur(img, 5)

# Bilateral filter
res_bilateral = cv2.bilateralFilter(img, 15, 75, 75)

# Display results
titles = ['Original', 'Blur', 'Gaussian Blur', 'Median Blur', 'Bilateral Filter']
```

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

Bilateral Filter: Result

https://www.geeksforgeeks.org/python-bilateral-filtering/

Push Code to GitHub

References

References

- OpenCV Python Tutorials
  - Image Processing
    - Image Thresholding
    - Smoothing Images