영상 전처리

ICTIS 김 보 연

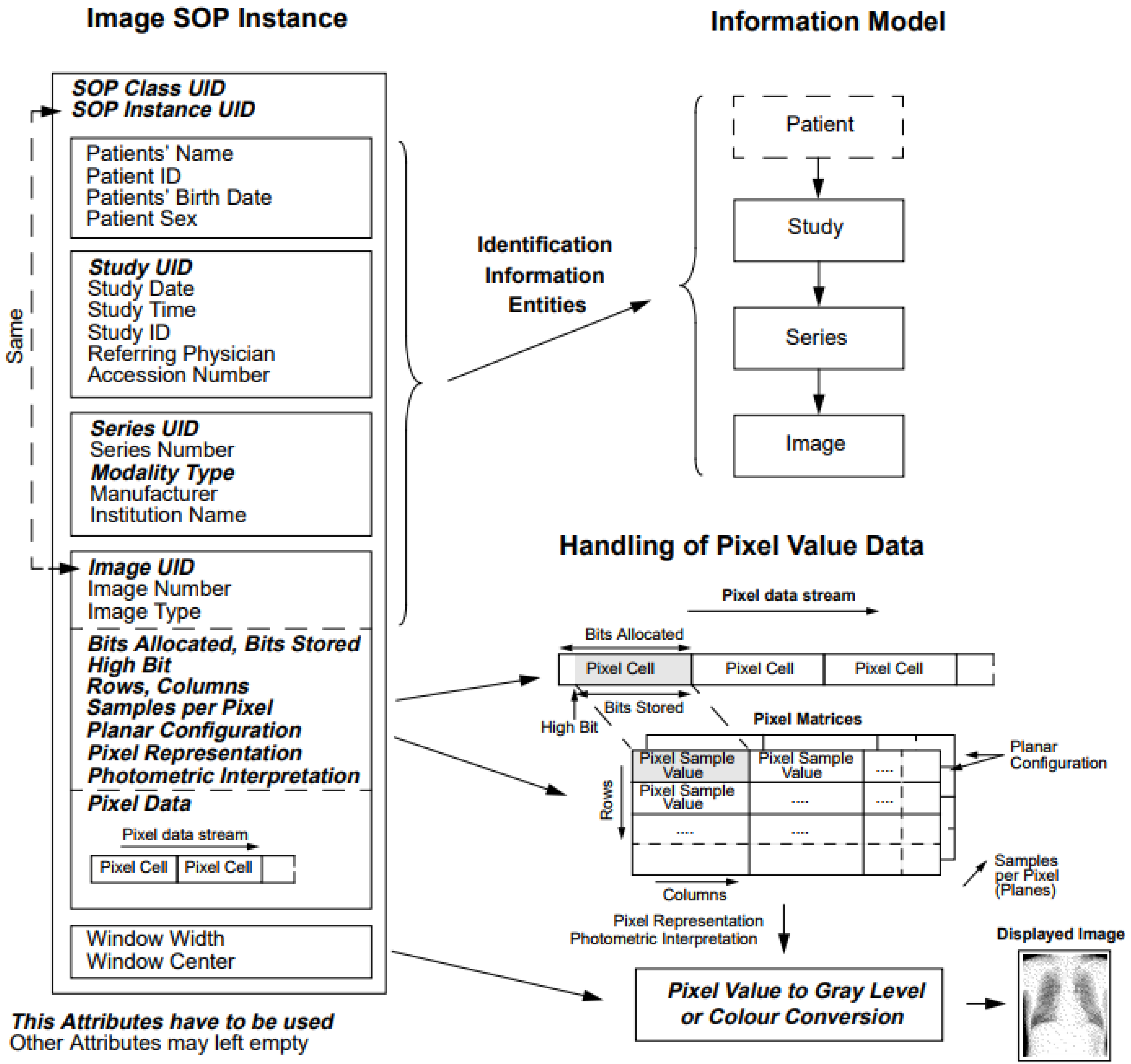


Image SOP Instance

Information Model

SOP Class UID
SOP Instance UID

Patients' Name
Patient ID
Patients' Birth Date
Patient Sex

Study UID
Study Date
Study Time
Study ID
Referring Physician
Accession Number

Series UID
Series Number
Modality Type
Manufacturer
Institution Name

Image UID
Image Number
Image Type

Bits Allocated, Bits Stored
High Bit
Rows, Columns
Samples per Pixel
Planar Configuration
Pixel Representation
Photometric Interpretation

Pixel Data
Pixel data stream
Pixel Cell | Pixel Cell

Window Width
Window Center

Same

Identification
Information
Entities

Patient
Study
Series
Image

Handling of Pixel Value Data

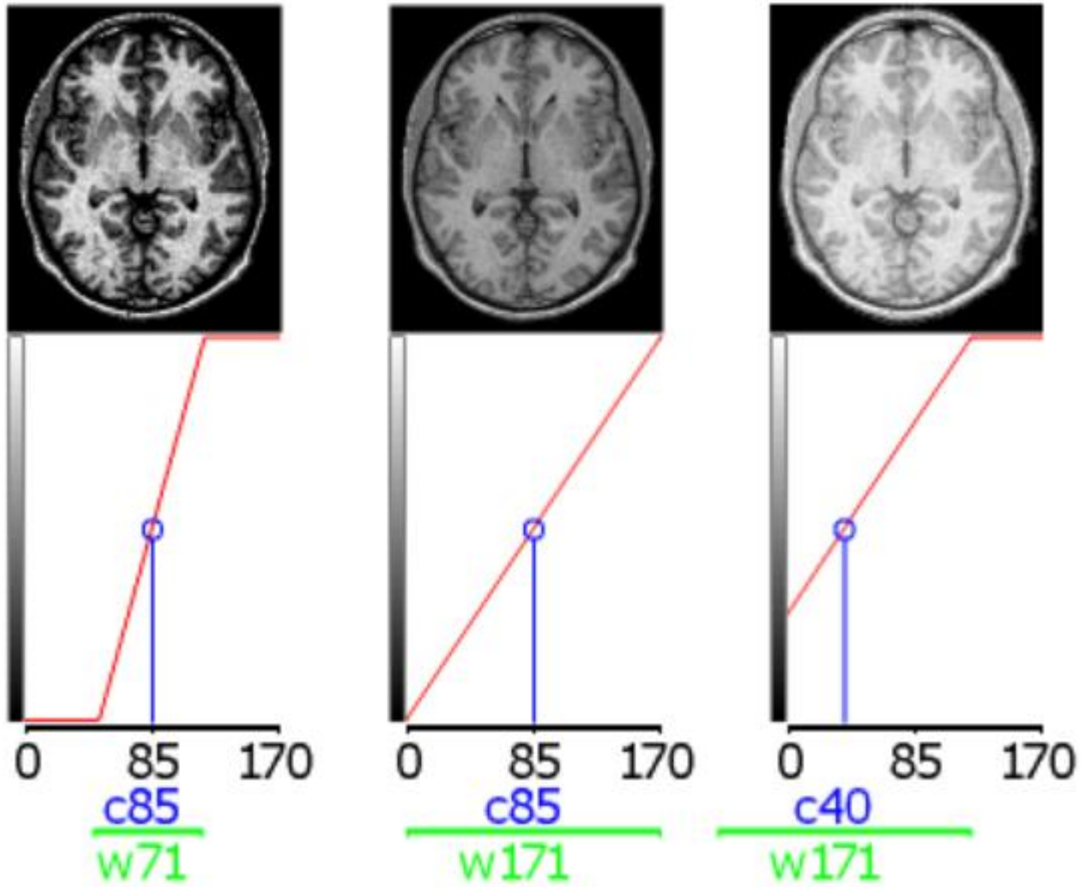
Pixel data stream
Bits Allocated
Pixel Cell
Pixel Cell
Pixel Cell
High Bit
Bits Stored
Pixel Matrices
Pixel Sample Value
Pixel Sample Value
Pixel Sample Value
Rows
Columns
Planar Configuration
Samples per Pixel (Planes)
Pixel Representation
Photometric Interpretation
Pixel Value to Gray Level or Colour Conversion
Displayed Image

This Attributes have to be used
Other Attributes may left empty

Figure 2-5: Basic Set of Attributes Image SOP Instances

Window center and window width (aka brightness and contrast)

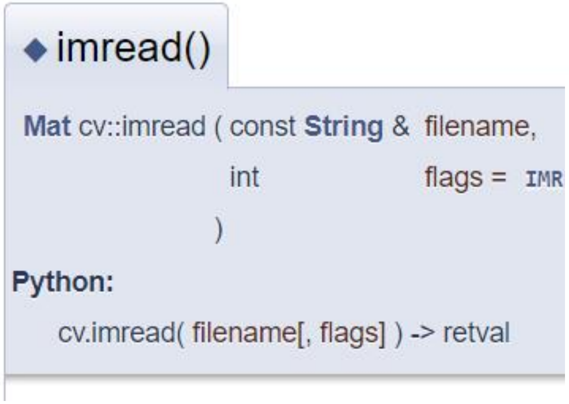
People familiar with the medical imaging typically talk about the 'window center' and the 'window width' of an image. This is simply a way of describing the 'brightness' and 'contrast' of the image. These values are particularly important for Xray/CT/PET scanners that tend to generate consistently calibrated intensities so you can use a specific C:W pair for every image you see (e.g. 400:2000 might be good for visualising bone, while 50:350 might be a better choice for soft tissue). Note that contrast in MRI scanners is relative, and so a C:W pair that works well for one protocol will probably be useless with a
different protocol or on a diffe es
to'window center' and 'window ge
with different C:W settings. Th e
vertical axis of the graph show
intensity). Consider this image this
image might be a center of 85 e
middle panel. Reducing the wi l,
keeping a width of 171 but re er.



OpenCV 활용 의료 영상 전처리

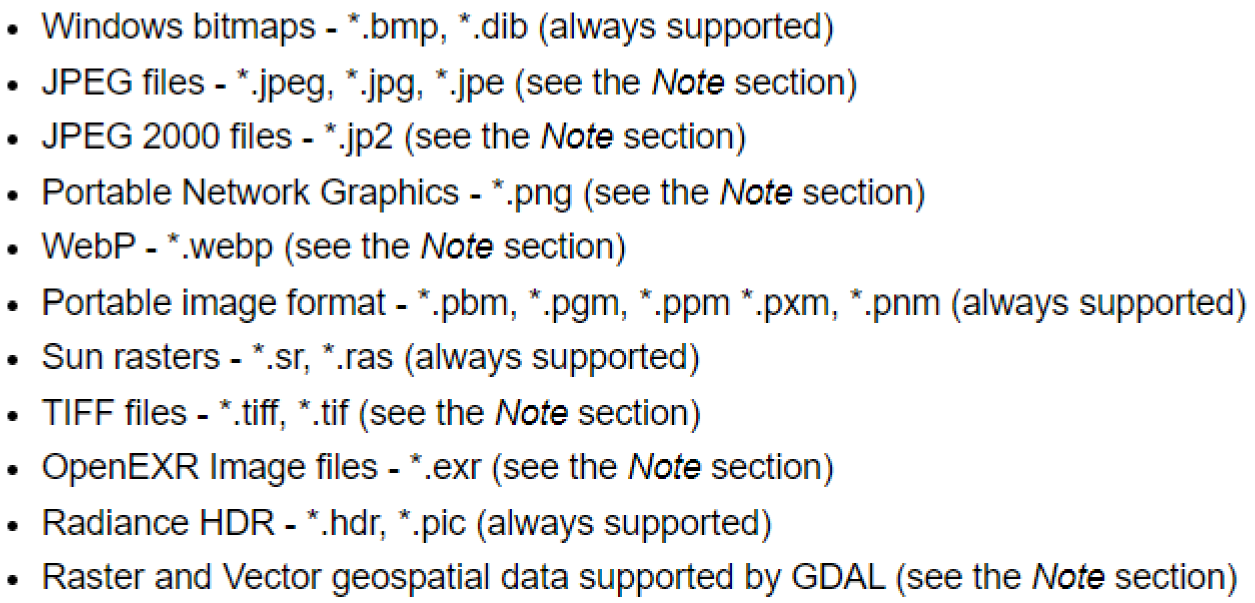
영상 파일 읽기

- import cv2
- 영상 읽기



◆ imread()

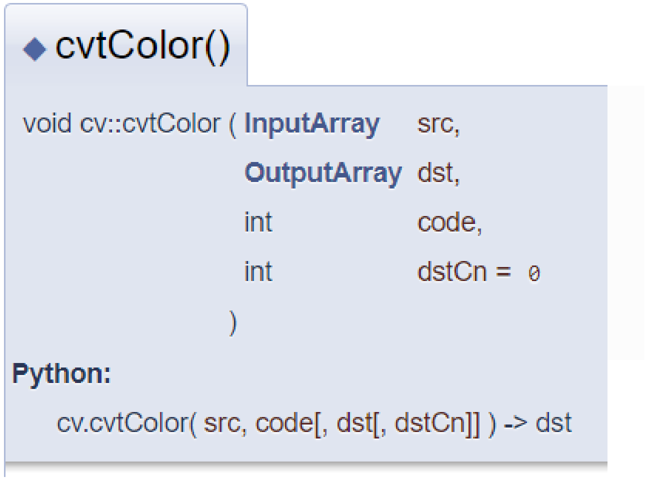
```
Mat cv::imread ( const String &  filename,
                 int             flags = IMR
               )
Python:
    cv.imread( filename[, flags] ) -> retval
```



- Windows bitmaps - *.bmp, *.dib (always supported)
- JPEG files - *.jpeg, *.jpg, *.jpe (see the *Note* section)
- JPEG 2000 files - *.jp2 (see the *Note* section)
- Portable Network Graphics - *.png (see the *Note* section)
- WebP - *.webp (see the *Note* section)
- Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm (always supported)
- Sun rasters - *.sr, *.ras (always supported)
- TIFF files - *.tiff, *.tif (see the *Note* section)
- OpenEXR Image files - *.exr (see the *Note* section)
- Radiance HDR - *.hdr, *.pic (always supported)
- Raster and Vector geospatial data supported by GDAL (see the *Note* section)

flags:
•cv2.IMREAD_COLOR : 이미지 파일을 Color로 읽어들임. 투명한 부분은 무시되며, Default값임.
•cv2.IMREAD_GRAYSCALE : 이미지를 Grayscale로 읽어 들임.. 실제 이미지 처리시 중간단계로 많이 사용.
•cv2.IMREAD_UNCHANGED : 이미지파일을 alpha channel까지 포함하여 읽어 들임.

Color map 변환

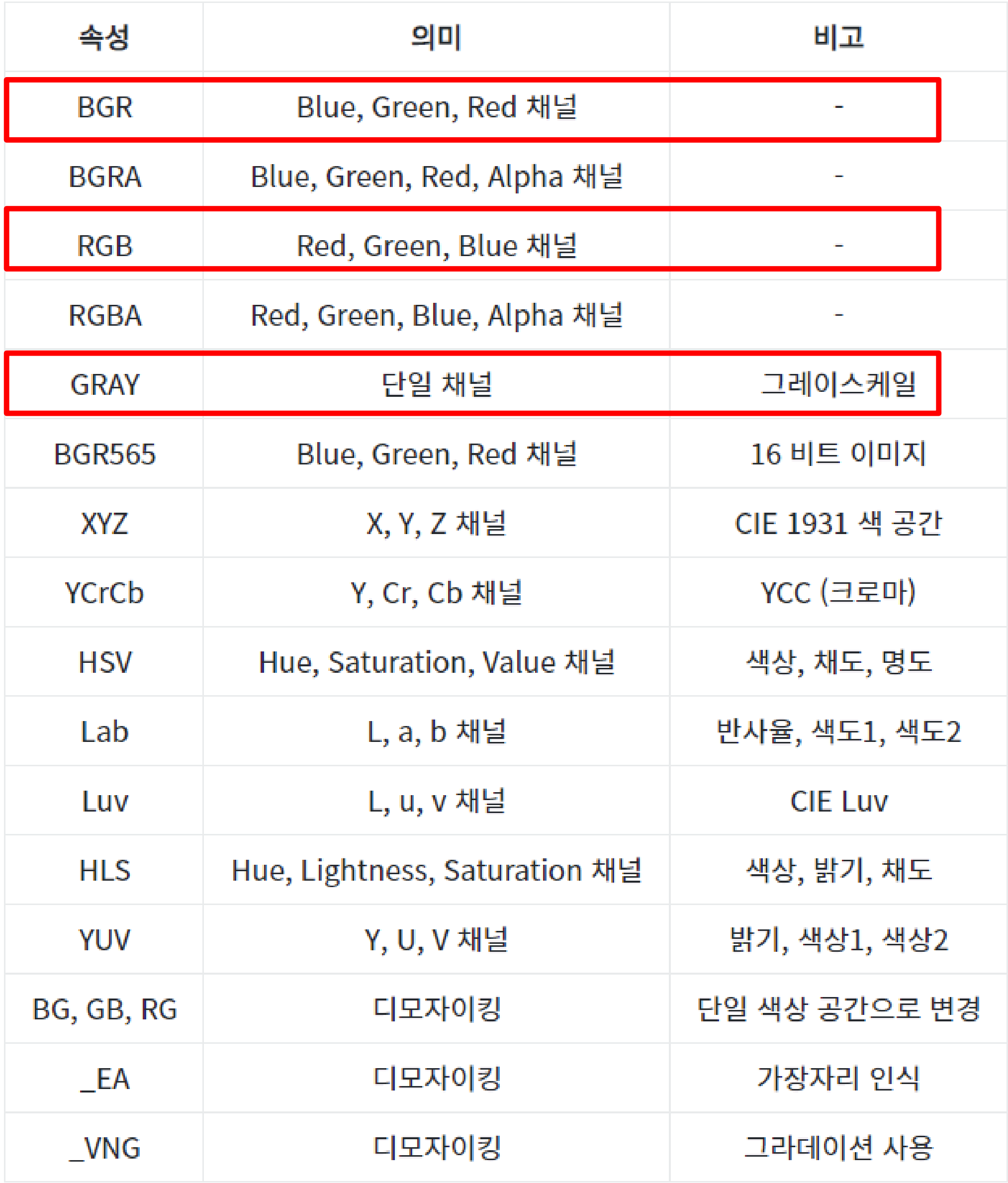


◆ cvtColor()

```
void cv::cvtColor ( InputArray    src,
                    OutputArray   dst,
                    int           code,
                    int           dstCn = 0
                  )
Python:
    cv.cvtColor( src, code[, dst[, dstCn]] ) -> dst
```

cv2.COLOR_BGR2GRAY BGR →Grayscale로 변
cv2.COLOR_BGR2HSV BGR →HSV로 변환
cv2.COLOR_BGR2RGB BGR → RGB로 변환

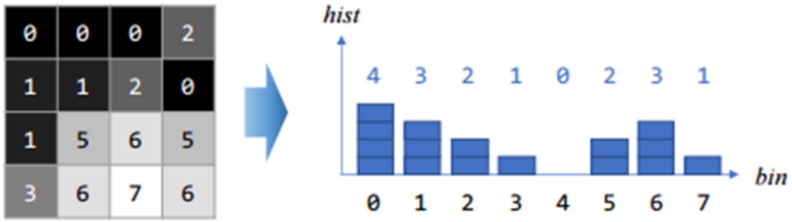
색상 공간 코드



속성	의미	비고
BGR	Blue, Green, Red 채널	-
BGRA	Blue, Green, Red, Alpha 채널	-
RGB	Red, Green, Blue 채널	-
RGBA	Red, Green, Blue, Alpha 채널	-
GRAY	단일 채널	그레이스케일
BGR565	Blue, Green, Red 채널	16 비트 이미지
XYZ	X, Y, Z 채널	CIE 1931 색 공간
YCrCb	Y, Cr, Cb 채널	YCC (크로마)
HSV	Hue, Saturation, Value 채널	색상, 채도, 명도
Lab	L, a, b 채널	반사율, 색도1, 색도2
Luv	L, u, v 채널	CIE Luv
HLS	Hue, Lightness, Saturation 채널	색상, 밝기, 채도
YUV	Y, U, V 채널	밝기, 색상1, 색상2
BG, GB, RG	디모자이킹	단일 색상 공간으로 변경
_EA	디모자이킹	가장자리 인식
_VNG	디모자이킹	그라데이션 사용

히스토그램 (Histogram)

- 영상의 각 값 혹은 값의 범위에 대한 픽셀 수 혹은 비율



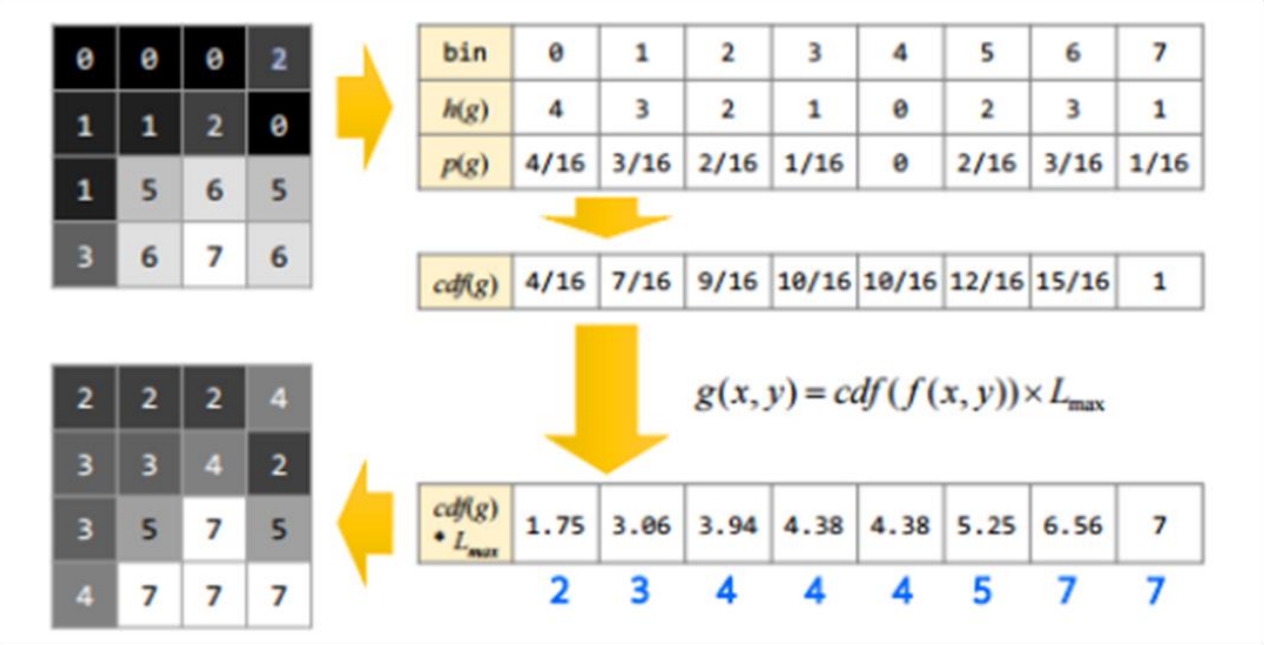
0	0	0	2
1	1	2	0
1	5	6	5
3	6	7	6

- 영상과 히스토그램의 관계
  - 밝은 영상
    - 히스토그램이 오른쪽으로 치우쳐져 있음
  - 어두운 영상
    - 히스토그램이 왼쪽으로 치우쳐져 있음
  - 명암 대비가 확실한 영상
    - 히스토그램이 전체적으로 퍼져있음

히스토그램 평활화 (histogram equa;ization)

- 명암비가 낮아 히스토그램이 특정 구간에 집중되어 있을 때, 전구간(0~255)에 펼치는 기법
- 히스토그램 평활화(histogram equalization) 원리

–히스토그램 누적 분포를 활용



0	0	0	2
1	1	2	0
1	5	6	5
3	6	7	6

bin	0	1	2	3	4	5	6	7
$h(g)$	4	3	2	1	0	2	3	1
$p(g)$	4/16	3/16	2/16	1/16	0	2/16	3/16	1/16

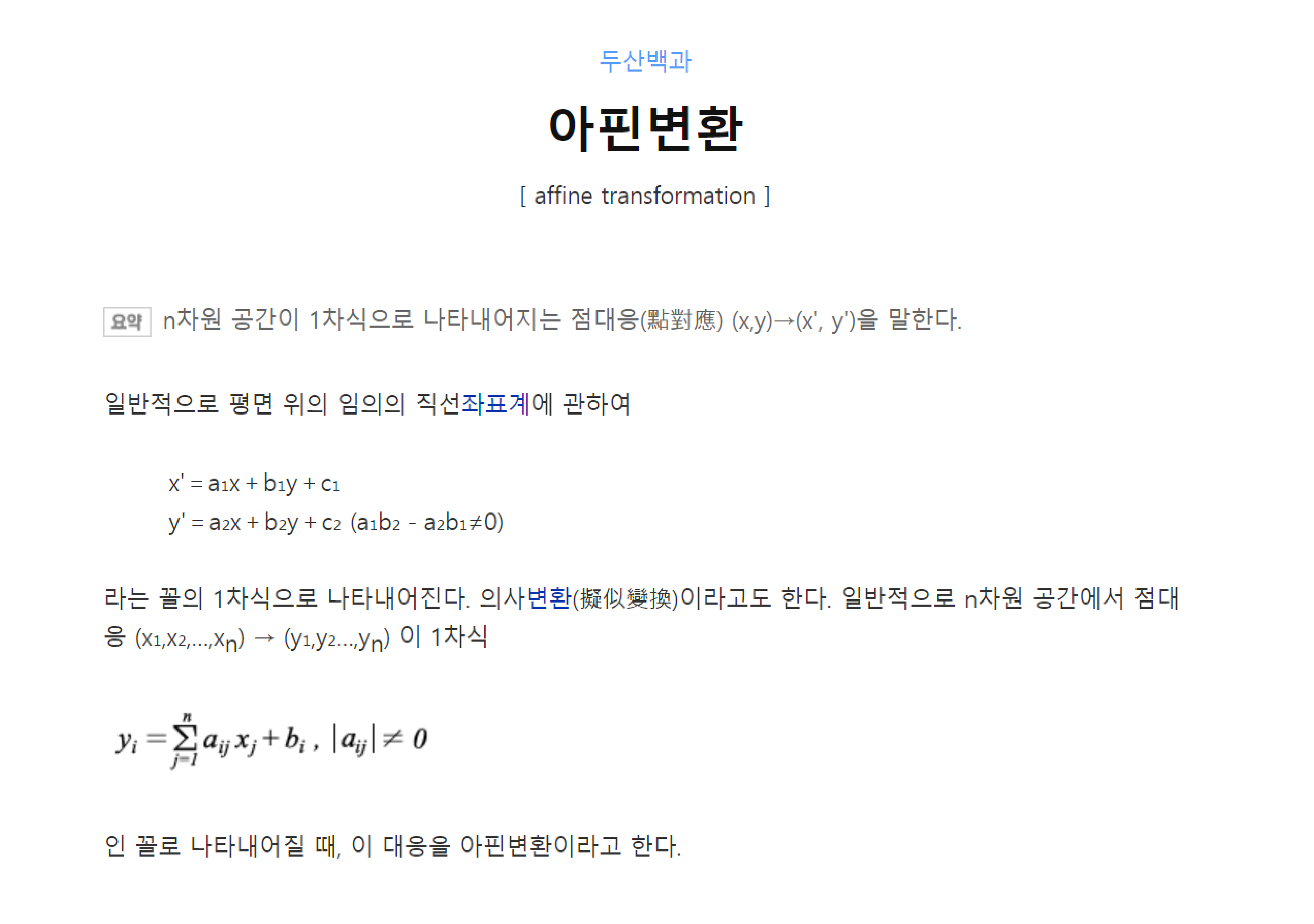
$cdf(g)$	4/16	7/16	9/16	10/16	10/16	12/16	15/16	1

$$g(x,y) = cdf(f(x,y)) \times L_{max}$$

$cdf(g) \bullet L_{max}$	1.75	3.06	3.94	4.38	4.38	5.25	6.56	7
	2	3	4	4	4	5	7	7

2	2	2	4
3	3	4	2
3	5	7	5
4	7	7	7

* Source: OpenCV4로 배우는 컴퓨터비전과 머신러닝



두산백과

아핀변환

[affine transformation]

요약 n차원 공간이 1차식으로 나타내어지는 점대응(點對應) (x,y)→(x', y')을 말한다.

일반적으로 평면 위의 임의의 직선좌표계에 관하여

$$x' = a_1x + b_1y + c_1$$
$$y' = a_2x + b_2y + c_2 \quad (a_1b_2 - a_2b_1 \neq 0)$$

라는 꼴의 1차식으로 나타내어진다. 의사변환(擬似變換)이라고도 한다. 일반적으로 n차원 공간에서 점대응 $(x_1,x_2,\ldots,x_n) \to (y_1,y_2\ldots,y_n)$ 이 1차식

$$y_i = \sum_{j=1}^{n} a_{ij} x_j + b_i \, , \ |a_{ij}| \neq 0$$

인 꼴로 나타내어질 때, 이 대응을 아핀변환이라고 한다.

영상의 기하학적 변환

- 이동
  - 영상을 특정 pixel 만큼 이동시키는 변환
  - 공식

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

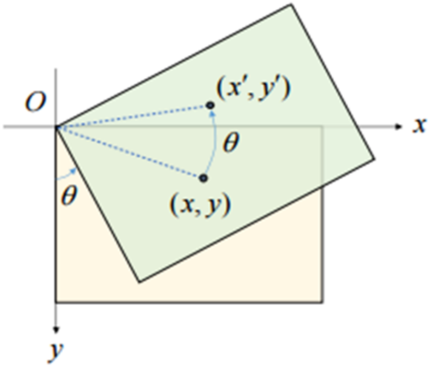
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Affine Transformation Matrix } A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

영상의 기하학적 변환

- 회전(rotate)
  - 영상을 특정 각도만큼 회전시키는 변환 (반
  - 공식

$$\begin{cases} x' = \cos\theta \cdot x + \sin\theta \cdot y \\ y' = -\sin\theta \cdot x + \cos\theta \cdot y \end{cases}$$

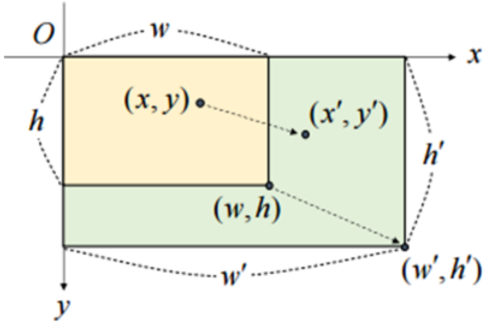
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\begin{vmatrix} \alpha & \beta & 1-\alpha \ast center.x - \beta \ast center.y \\ -\beta & \alpha & \beta \ast center.x + 1-\alpha \ast center.y \end{vmatrix}$$

영상의 기하학적 변환

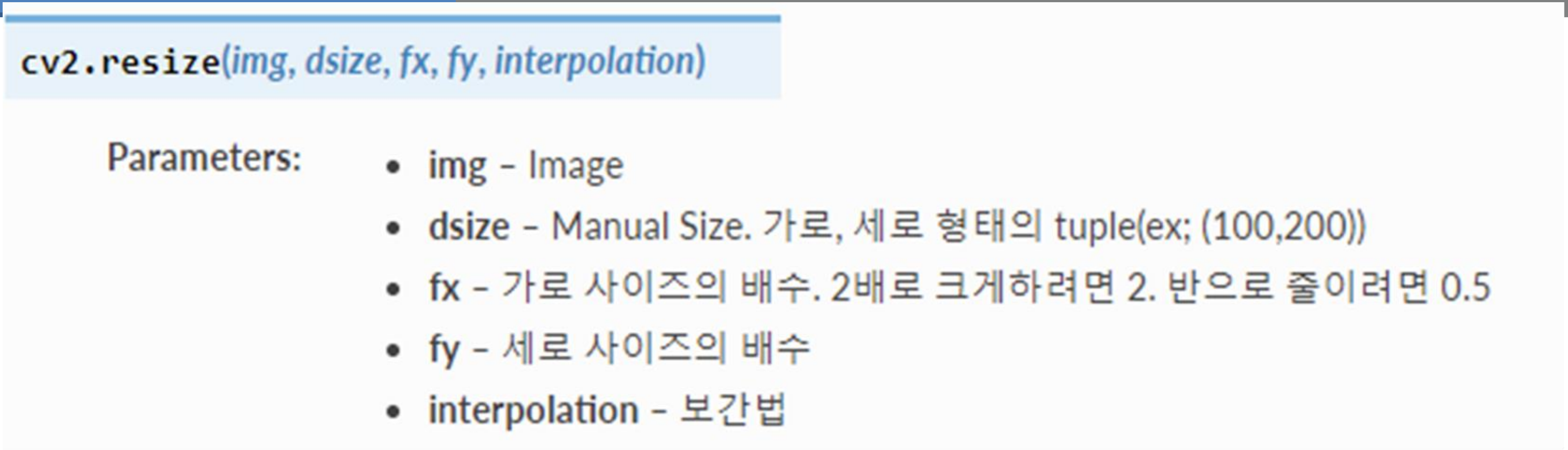
- 확대 & 축소 (scale transform)
  - 알고리즘에서 요구하는 해상도가 있다면 입력 이미지의 크기를 변경해 영상 처리를 진행
  - 검출하려는 객체가 너무 작거나 입력 이미지가 너무 큰 경우 입력 이미지 자체를 변환해서 영상 처리를 진행



$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases} \qquad \begin{cases} s_x = w' / w \\ s_y = h' / h \end{cases}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \boxed{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

* Source: OpenCV4로 배우는 컴퓨터비전과 머신러닝

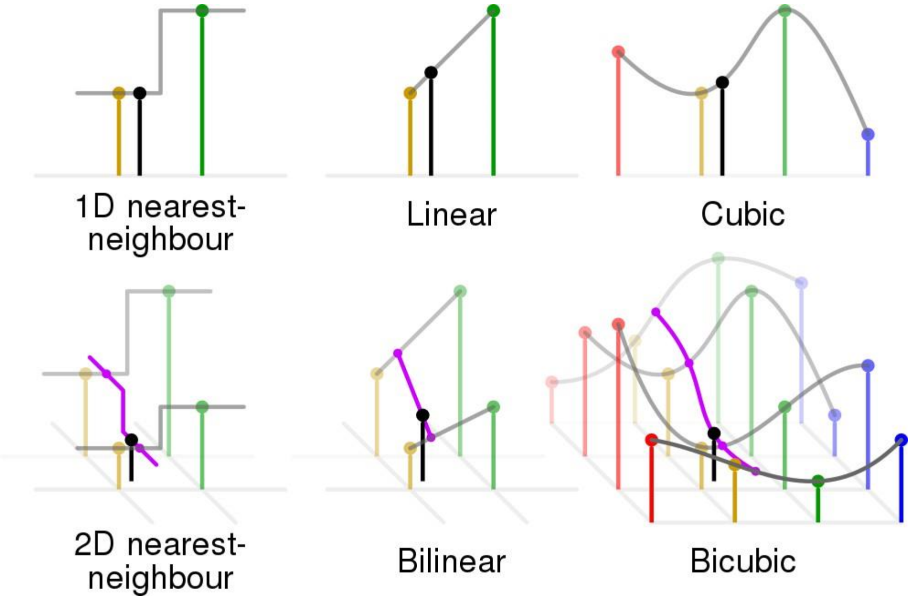


```
cv2.resize(img, dsize, fx, fy, interpolation)
```

Parameters:

- **img** – Image
- **dsize** – Manual Size. 가로, 세로 형태의 tuple(ex; (100,200))
- **fx** – 가로 사이즈의 배수. 2배로 크게하려면 2. 반으로 줄이려면 0.5
- **fy** – 세로 사이즈의 배수
- **interpolation** – 보간법

사이즈를 줄일 때는 cv2.INTER_AREA ,
사이즈를 크게할 때는 cv2.INTER_CUBIC , cv2.INTER_LINEAR 을 사용



1D nearest-neighbour

Linear

Cubic

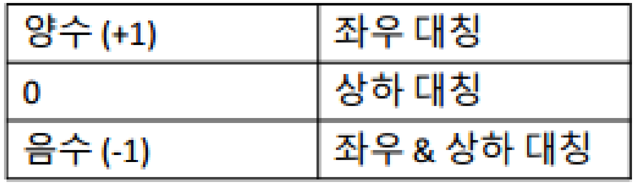
2D nearest-neighbour

Bilinear

Bicubic

영상의 기하학적 변환

- 대칭(flip):
  - 기하학적인 측면에서 **반사(reflection)**의 의미
- 대칭 변환의 방향

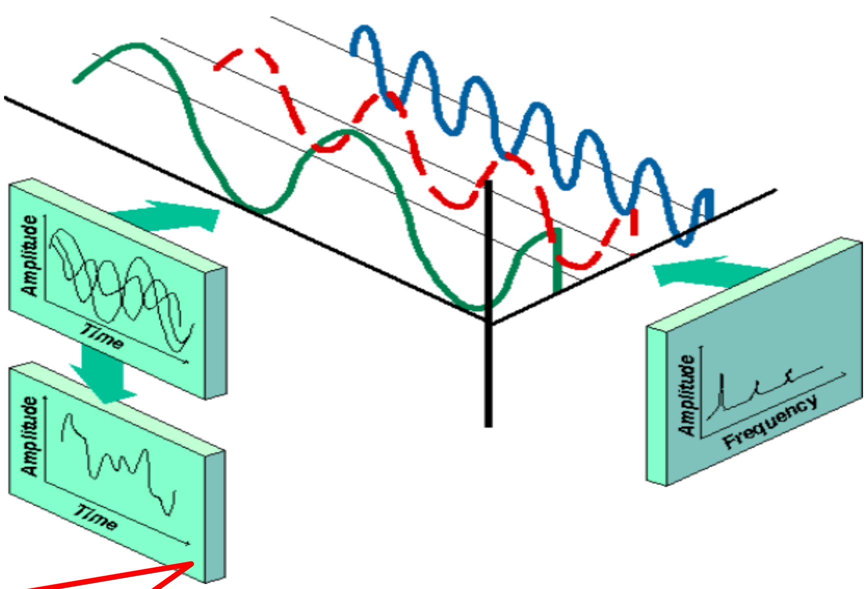


양수 (+1)	좌우 대칭
0	상하 대칭
음수 (-1)	좌우 & 상하 대칭

* Source: OpenCV4로 배우는 컴퓨터비전과 머신러닝

▪ 푸리에 변환(Fourier transform)

- 주파수 성분을 가지는 신호, 음성, 통신분야에서 중요하게 사용되는 처리방법
  - 입력 신호를 다양한 주파수를 갖는 주기함수들의 합으로 분해하여 표현하는 것
  - 주기함수: sin, cos 함수

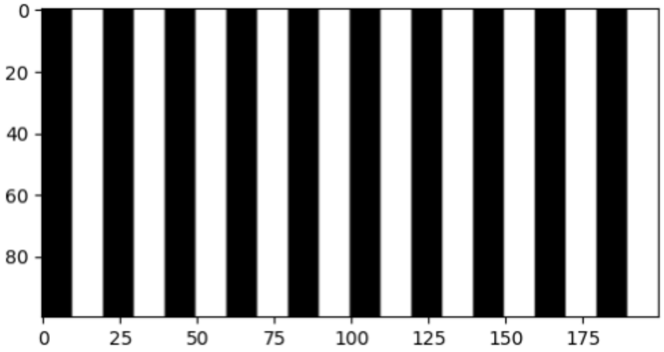


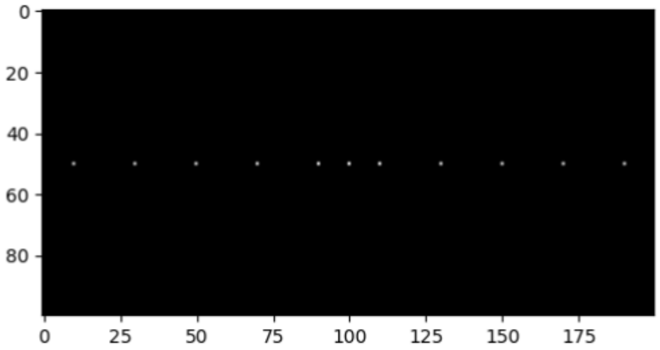
원본신호는 주기 신호들의 합

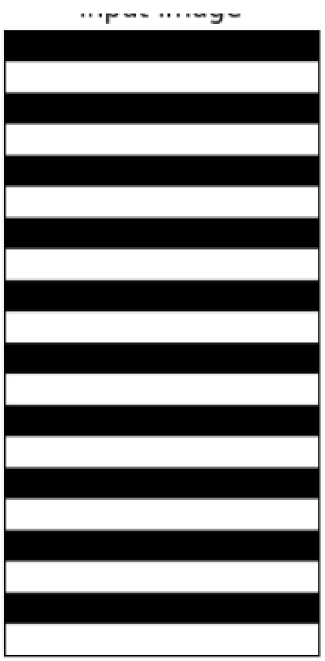
$$Y(f) = \int_{-\infty}^{\infty} y(t) e^{-i2\pi ft} dt$$

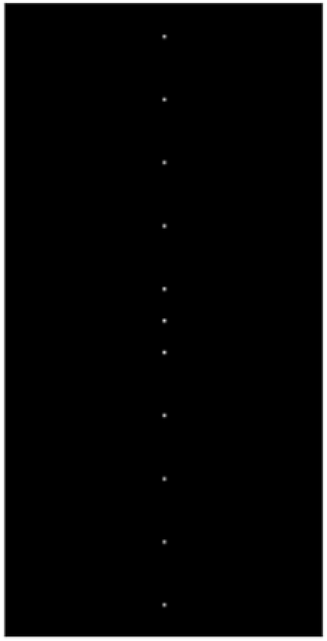
$$y(t) = \int_{-\infty}^{\infty} Y(f) e^{i2\pi ft} df$$

- 2차원 푸리에 변환
  - –https://docs.opencv.org/3.4/d8/d01/tutorial_discrete_fourier_transform.html



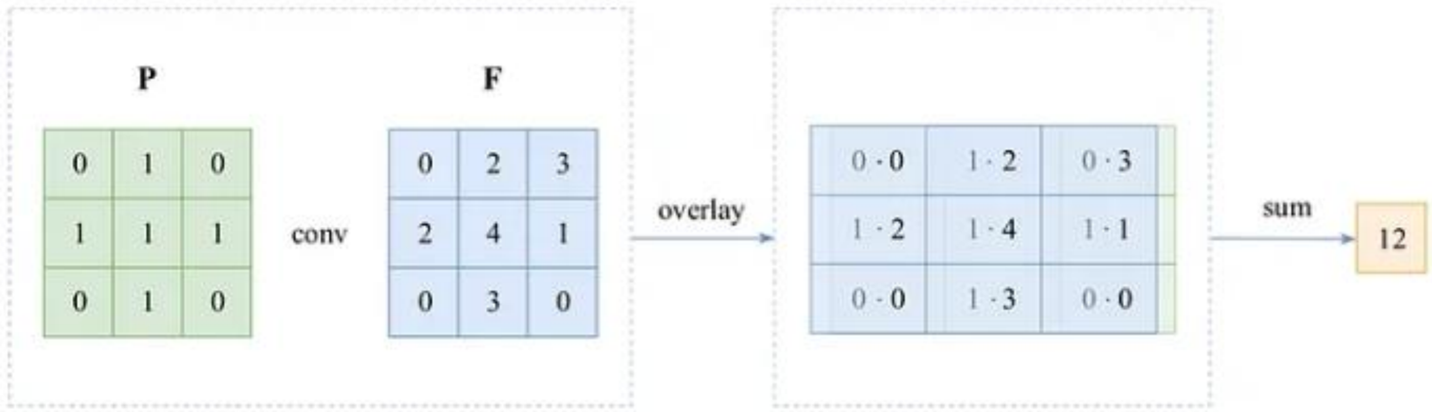






컨볼루션 연산

$$y[x,y]=\sum_{k=0}^{M}\sum_{l=0}^{N}h[k,l]x[x-k,y-l]$$



P

0	1	0
1	1	1
0	1	0

F

0	2	3
2	4	1
0	3	0

overlay

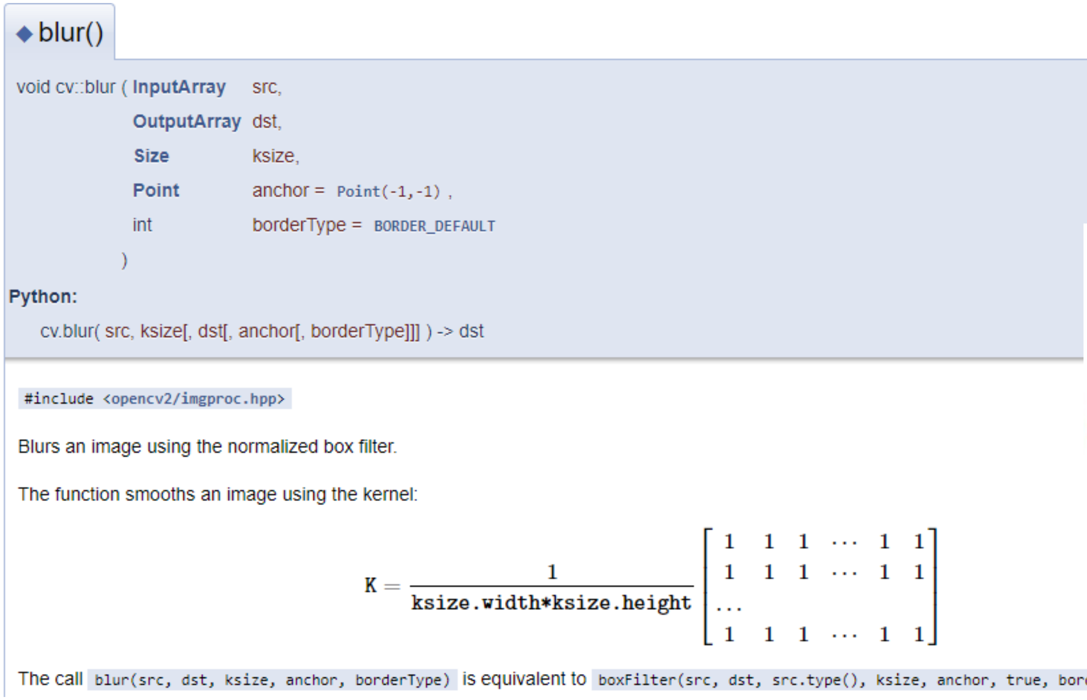
0 · 0	1 · 2	0 · 3
1 · 2	1 · 4	1 · 1
0 · 0	1 · 3	0 · 0

sum → 12

- Source: https://medium.com/analytics-vidhya/introduction-to-convolutional-neural-network-6942c189a723

영상 블러링

- 블러링(blurring)
  - 날카로운 에지가 무뎌지고, 영상에 있는 잡음의 영향이 사라지는 효과
  - 평균값 필터:
    - 영상의 특정 좌표 값을 주변 픽셀 값들의 산술 평균으로 설정
  - 마스크 크기가 커질수록 평균 값 필터 결과가 더욱 부드러워 짐



◆ blur()

```
void cv::blur ( InputArray   src,
                OutputArray  dst,
                Size         ksize,
                Point        anchor = Point(-1,-1) ,
                int          borderType = BORDER_DEFAULT
              )
```

Python:

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

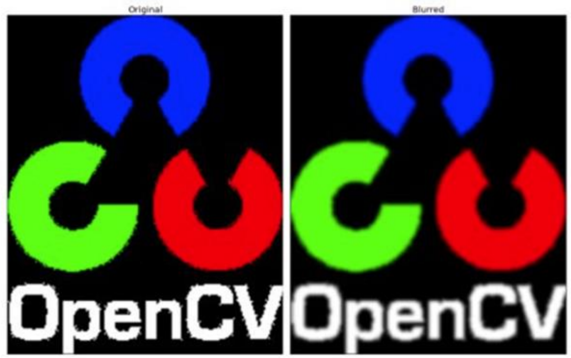
```
#include <opencv2/imgproc.hpp>
```

Blurs an image using the normalized box filter.

The function smooths an image using the kernel:

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \dots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

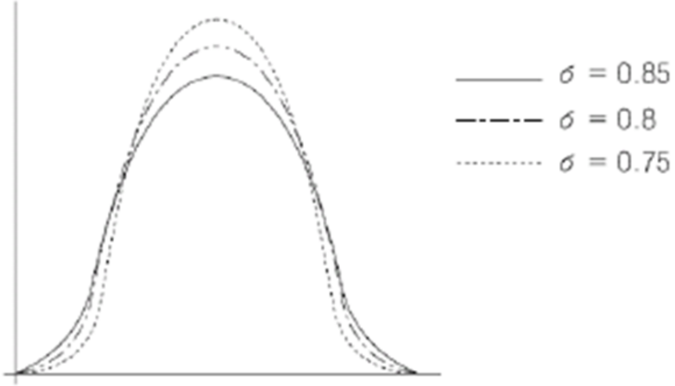
The call `blur(src, dst, ksize, anchor, borderType)` is equivalent to `boxFilter(src, dst, src.type(), ksize, anchor, true, borderType)`.



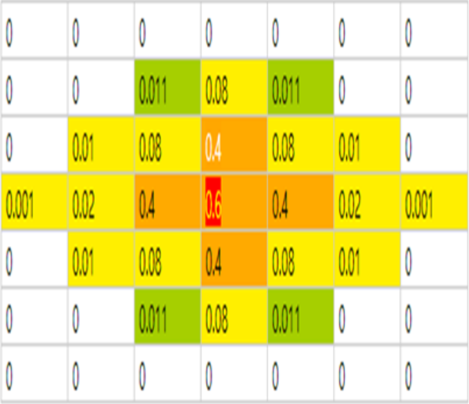
영상 블러링

- 가우시안 블러링

$$G[x,y]=\frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{2\pi\sigma^2}$$



- –σ 값이 클수록 높이는 낮지만 폭은 넓어지므.
  많은 저주파 성분을 통과
- –즉, σ 값을 조절하여 고주파량과 저주파량을 조절
  - sigmaX, sigmaY 로 조절



0	0	0	0	0	0	0
0	0	0.011	0.08	0.011	0	0
0	0.01	0.08	0.4	0.08	0.01	0
0.001	0.02	0.4	0.6	0.4	0.02	0.001
0	0.01	0.08	0.4	0.08	0.01	0
0	0	0.011	0.08	0.011	0	0
0	0	0	0	0	0	0

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

영상의 가장자리(edge) 구하기

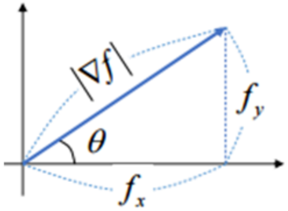
- 가장자리(edge)
  - 밝기가 큰 폭으로 변하는 지점
  - 픽셀의 밝기 변화율이 높은 경계선을 찾기 위해 미분(Derivative)과 기울기(Gradient) 연산을 수행
  - 영상에서 그래디언트의 크기는 픽셀값의 차이
  - 그래디언트의 방향은 픽셀값이 가장 급격하게 증가하는 방향

$$|\nabla f| = \sqrt{f_x^{\,2} + f_y^{\,2}}$$

그래디언트 크기

$$\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$$

그래디언트 방향



엣지 검출(edge extraction) 필터들

- 로버츠(Roberts) : 계산 속도가 빠른 필터
- 소벨(Sobel): 뚜렷한 에지 검출
- 프리위트(Prewitt) : 소벨보다 간단하며 속도빠름

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}.$$

로버츠 공식

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$

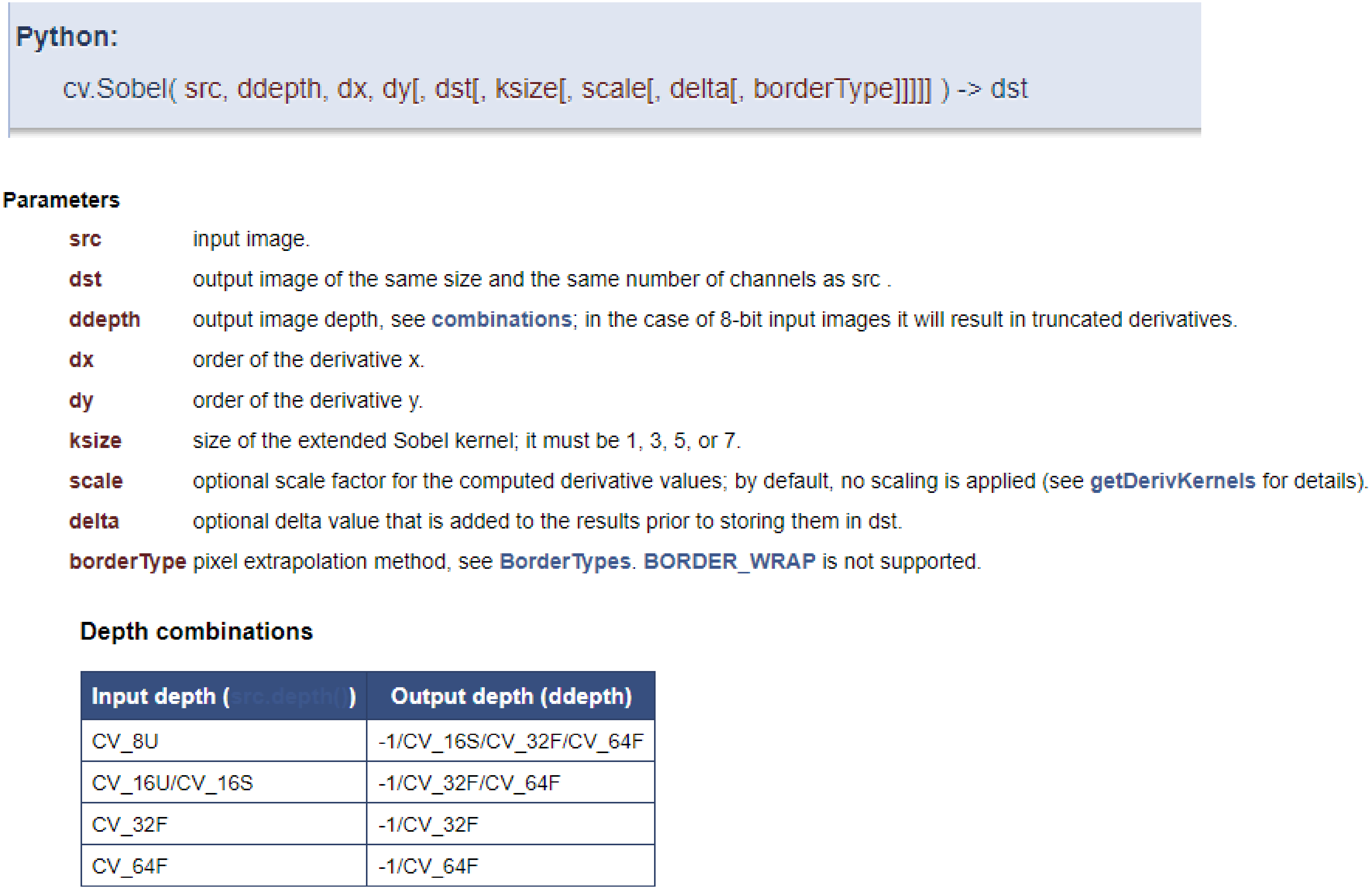
프리윗 공식

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

라플라시안

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

소벨 공색



Python:

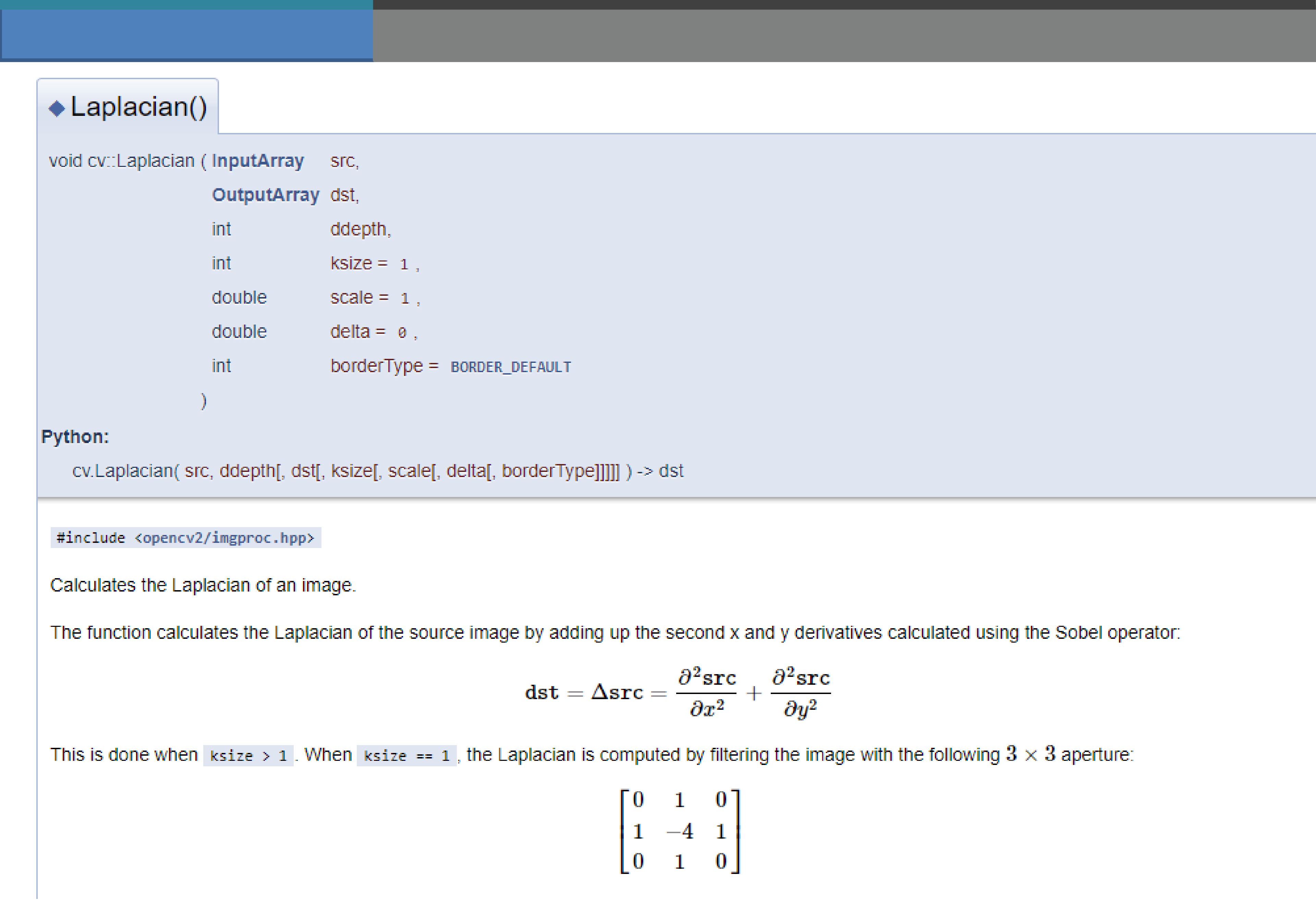
```
cv.Sobel( src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]] ) -> dst
```

Parameters

- **src** input image.
- **dst** output image of the same size and the same number of channels as src .
- **ddepth** output image depth, see **combinations**; in the case of 8-bit input images it will result in truncated derivatives.
- **dx** order of the derivative x.
- **dy** order of the derivative y.
- **ksize** size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
- **scale** optional scale factor for the computed derivative values; by default, no scaling is applied (see **getDerivKernels** for details).
- **delta** optional delta value that is added to the results prior to storing them in dst.
- **borderType** pixel extrapolation method, see **BorderTypes**. **BORDER_WRAP** is not supported.

Depth combinations

Input depth (src.depth())	Output depth (ddepth)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F
CV_64F	-1/CV_64F



◆ Laplacian()

```
void cv::Laplacian ( InputArray   src,
                     OutputArray  dst,
                     int          ddepth,
                     int          ksize = 1 ,
                     double       scale = 1 ,
                     double       delta = 0 ,
                     int          borderType = BORDER_DEFAULT
                   )
```

Python:

```
cv.Laplacian( src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]] ) -> dst
```

`#include <opencv2/imgproc.hpp>`

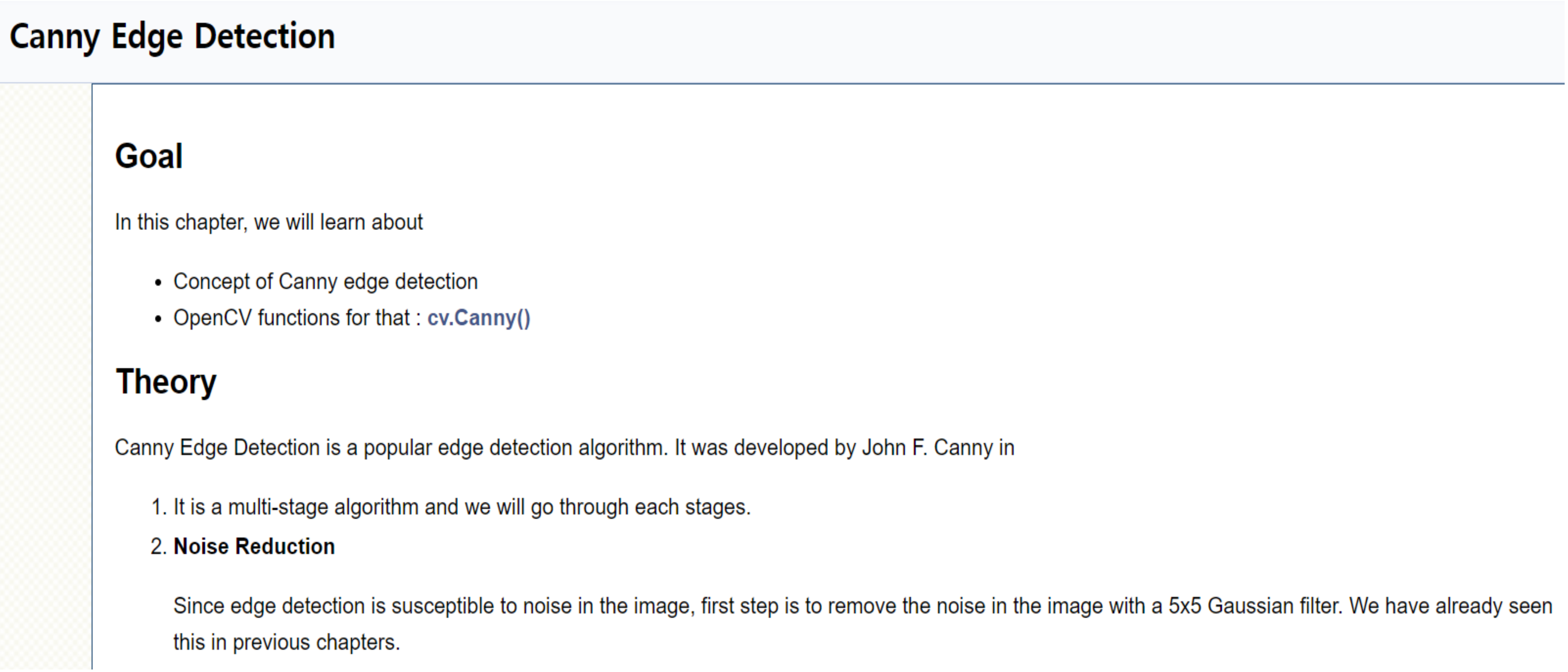
Calculates the Laplacian of an image.

The function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator:

$$\texttt{dst} = \Delta \texttt{src} = \frac{\partial^2 \texttt{src}}{\partial x^2} + \frac{\partial^2 \texttt{src}}{\partial y^2}$$

This is done when `ksize > 1`. When `ksize == 1`, the Laplacian is computed by filtering the image with the following 3×3 aperture:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Canny Edge Detection

Goal

In this chapter, we will learn about

- Concept of Canny edge detection
- OpenCV functions for that : **cv.Canny()**

Theory

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

1. It is a multi-stage algorithm and we will go through each stages.
2. **Noise Reduction**

   Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

2. Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:
    a. Apply a pair of convolution masks (in x and y directions:

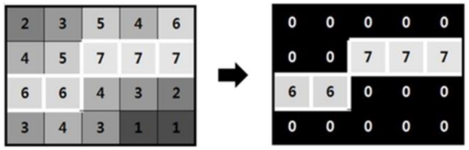
$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

    b. Find the gradient strength and direction with:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan(\frac{G_y}{G_x})$$



2	3	5	4	6
4	5	7	7	7
6	6	4	3	2
3	4	3	1	1

0	0	0	0	0
0	0	7	7	7
6	6	0	0	0
0	0	0	0	0

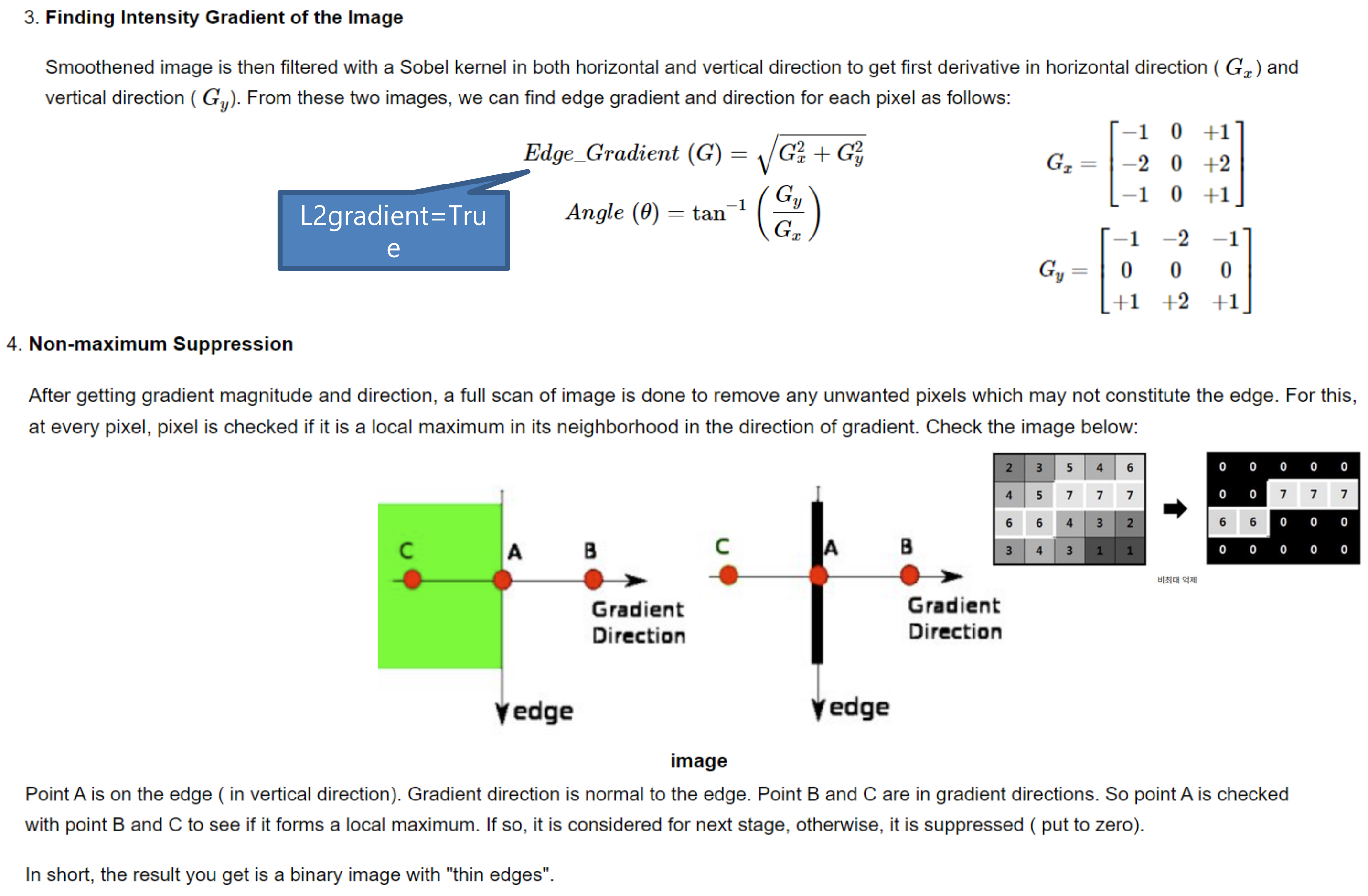
비최대 억제

The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

3. *Non-maximum* suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.
4. *Hysteresis*: The final step. Canny does use two thresholds (upper and lower):

    a. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
    b. If a pixel gradient value is below the *lower* threshold, then it is rejected.
    c. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

Canny recommended a *upper*:*lower* ratio between 2:1 and 3:1.



3. **Finding Intensity Gradient of the Image**

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\;(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\;(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

L2gradient=True

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

4. **Non-maximum Suppression**

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:

2	3	5	4	6
4	5	7	7	7
6	6	4	3	2
3	4	3	1	1

➡

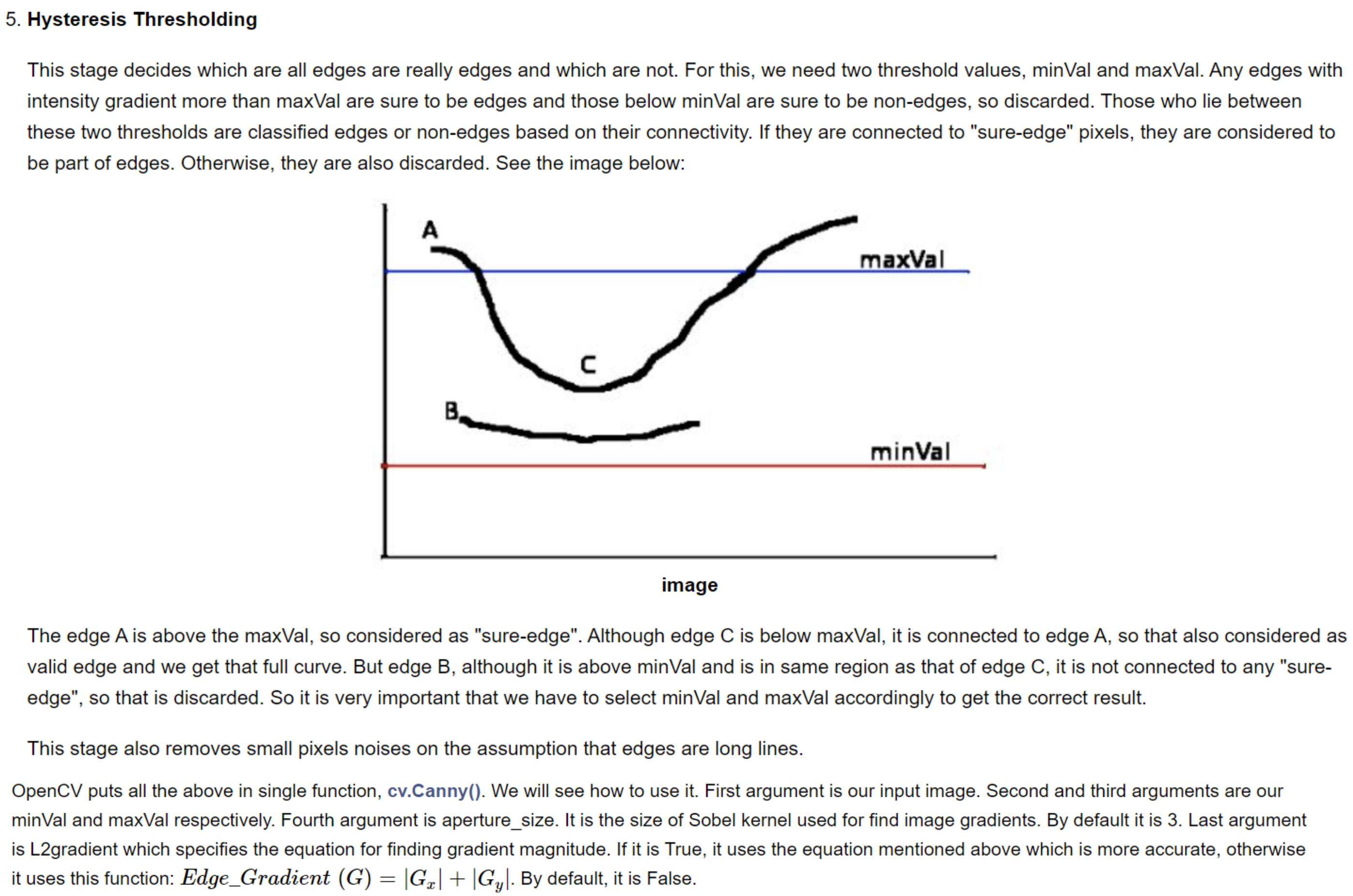
0	0	0	0	0
0	0	7	7	7
6	6	0	0	0
0	0	0	0	0

비최대 억제

image

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

In short, the result you get is a binary image with "thin edges".



5. **Hysteresis Thresholding**

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

image

The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

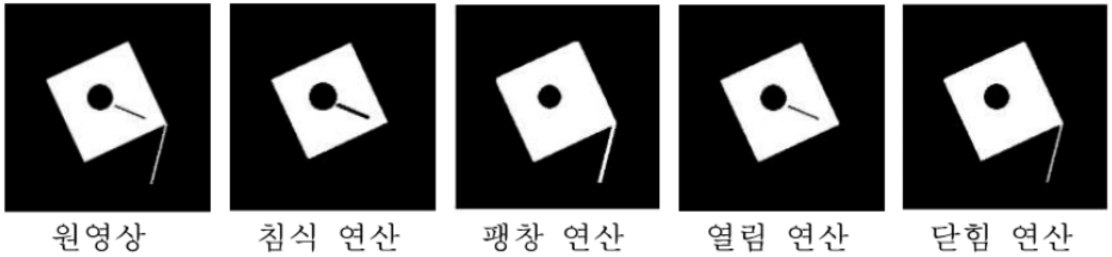
This stage also removes small pixels noises on the assumption that edges are long lines.

OpenCV puts all the above in single function, **cv.Canny()**. We will see how to use it. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Fourth argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is L2gradient which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function: $Edge\_Gradient\;(G) = |G_x| + |G_y|$. By default, it is False.

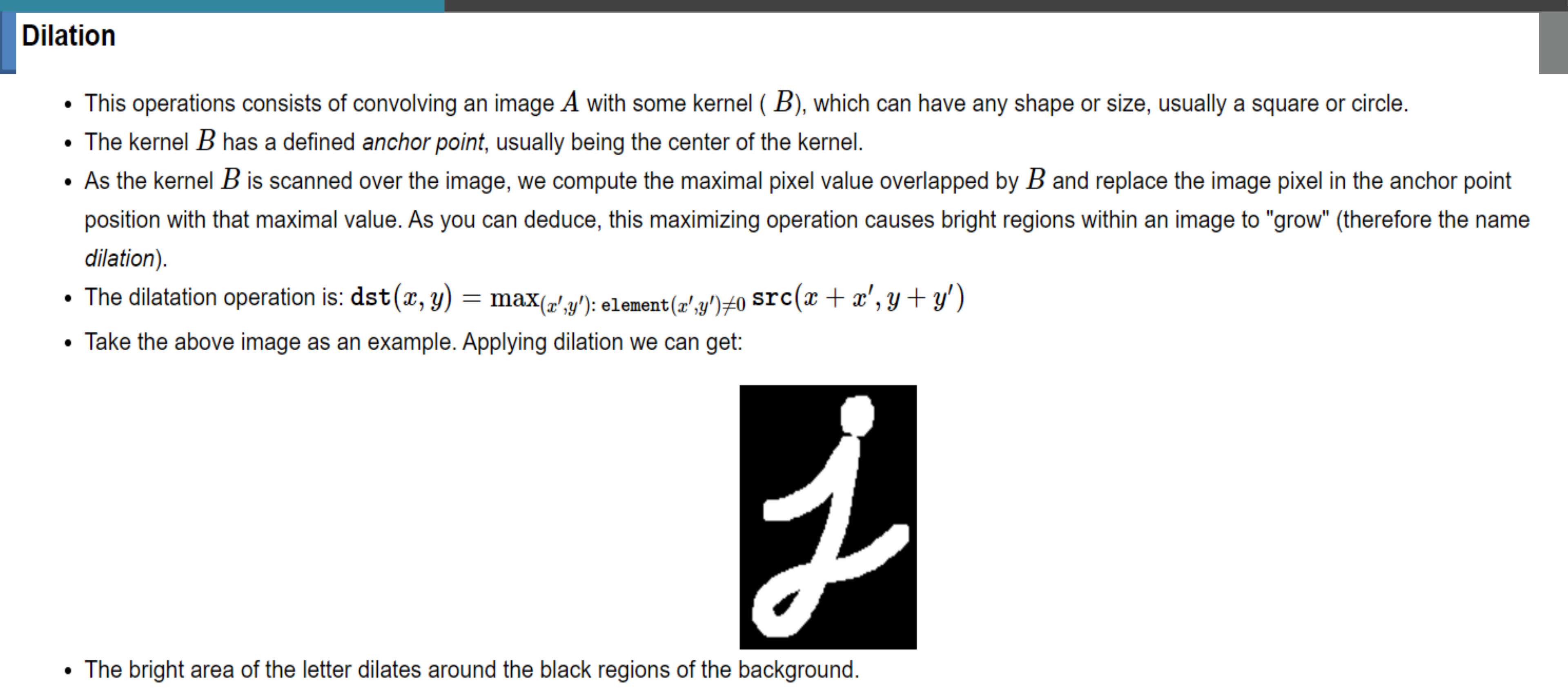
L2gradient=False

모폴로지 변환

- 모폴로지(morphology, 형태학)
  - 영상을 형태(모양)학적인 측면으로 접근하는 방법
  - 객체 세그멘테이션이나 문자인식 전처리에 이용됨
  - 침식, 팽창, 열림, 닫힘
- 침식(erosion) 연산
  - 객체 외곽을 깍아내는 연산
  - 객체 영역(흰색)이 줄어들어 잡음 제거 효과가 있음
- 팽창(dilation) 연산
  - 객체가 커지는 연산
  - 객체 영역(흰색)이 커지는 연산
  - 객체 내부의 홀(구멍)이 채워짐
  - 노이즈 제거 후 줄어든 크기를 복구하고자 할 때 주로 사용



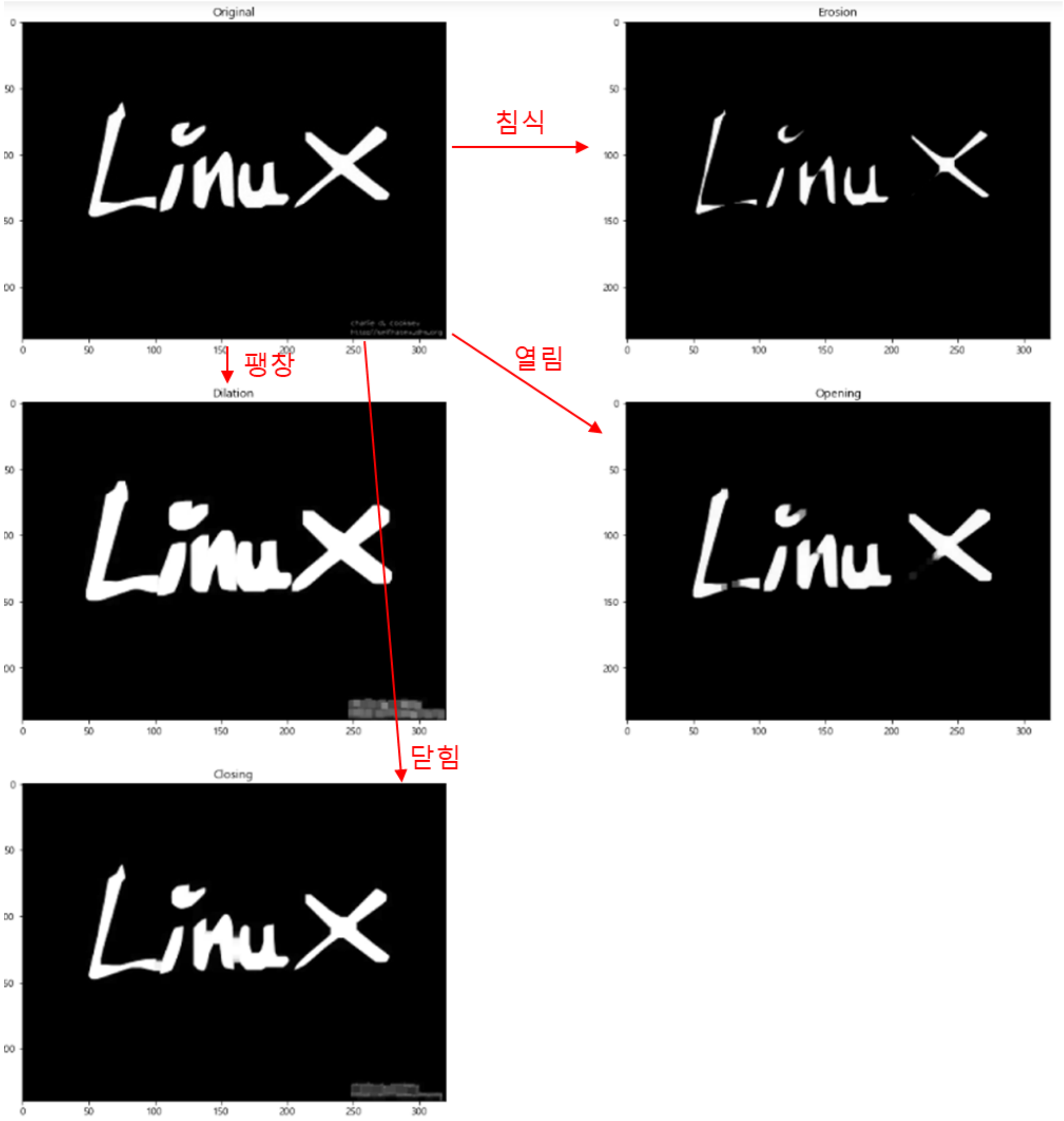
Source: OpenCV로 [illegible]



Dilation

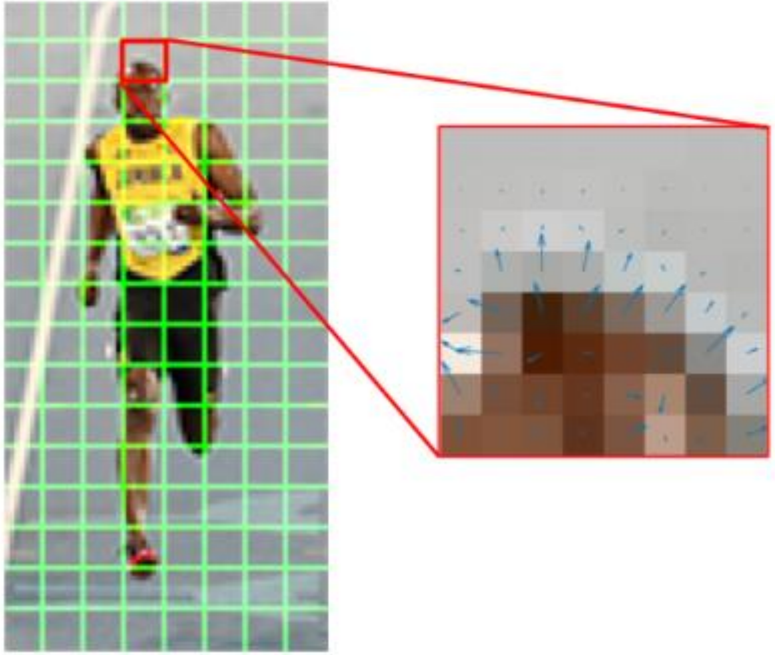
- This operations consists of convolving an image A with some kernel (B), which can have any shape or size, usually a square or circle.
- The kernel B has a defined *anchor point*, usually being the center of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*).
- The dilatation operation is: $\texttt{dst}(x,y) = \max_{(x',y'):\,\texttt{element}(x',y') \ne 0} \texttt{src}(x+x',y+y')$
- Take the above image as an example. Applying dilation we can get:
- The bright area of the letter dilates around the black regions of the background.

https://docs.opencv.org/4.x/db/df6/tutorial_erosion_dilatation.html

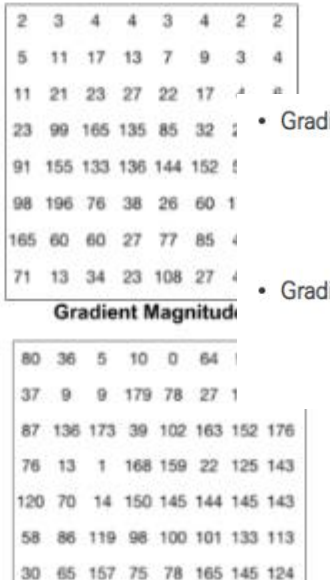

Original
Erosion
침식
팽창
열림
Dilation
Opening
닫힘
Closing
Linux
0
50
100
150
200
250
300

- Vision-Based Classification of Skin Cancer Using Deep Learning
  - https://web.stanford.edu/~kalouche/docs/Vision_Based_Classification_of_Skin_Cancer_using_Deep_Learning_(Kalouche).pdf

- Ben Graham's preprocessing method
  - https://www.kaggle.com/code/habibmrad1983/aptos-eye-preprocessing-in-diabetic-retinopathy



가운데 : 화살표를 사용하여 RGB 패치와 그라디언트를 나타냅니다. 오른쪽 : 동일



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	2	
91	155	133	136	144	152	5	
98	196	76	38	26	60	1	
165	60	60	27	77	85	4	
71	13	34	23	108	27	4	

Gradient Magnitud

80	36	5	10	0	64		
37	9	9	179	78	27	1	
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124

- Gradient orientation: 그래디언트의 방향

$$\theta = tan^{-1}(\frac{dy}{dx})$$

- Gradient magnitude: intensity의 변화량(크기)

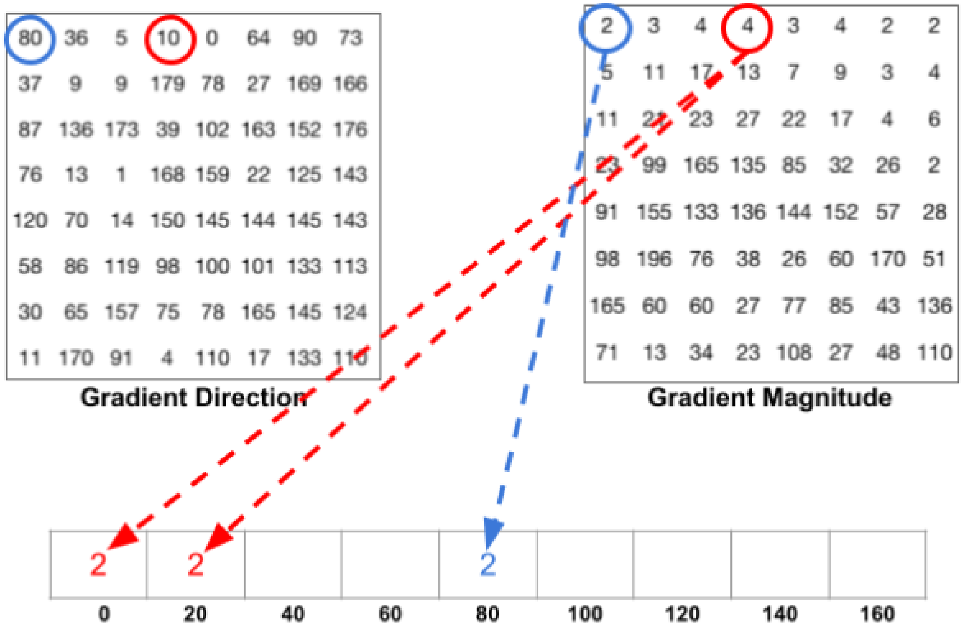
$$magnitude = \sqrt{dy^2 + dx^2}$$

$$\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Sobel

소벨필터



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

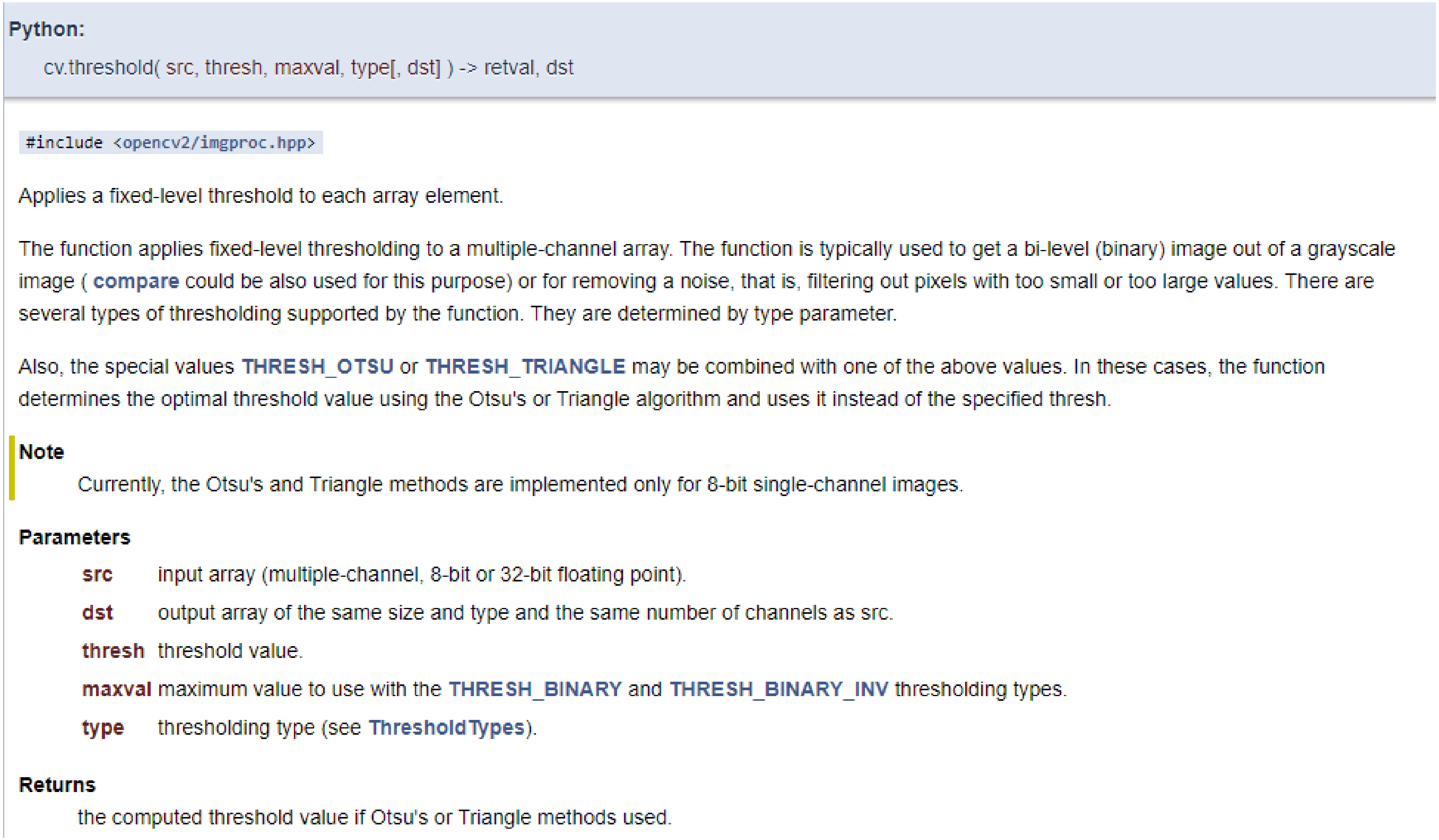
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

0	20	40	60	80	100	120	140	160
2	2			2				

Histogram of Gradients

*출처: Histograms of oriented gradients for human detection, Dalal, 2005



Python:

```
cv.threshold( src, thresh, maxval, type[, dst] ) -> retval, dst
```

```
#include <opencv2/imgproc.hpp>
```

Applies a fixed-level threshold to each array element.

The function applies fixed-level thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image (**compare** could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values. There are several types of thresholding supported by the function. They are determined by type parameter.

Also, the special values **THRESH_OTSU** or **THRESH_TRIANGLE** may be combined with one of the above values. In these cases, the function determines the optimal threshold value using the Otsu's or Triangle algorithm and uses it instead of the specified thresh.

Note

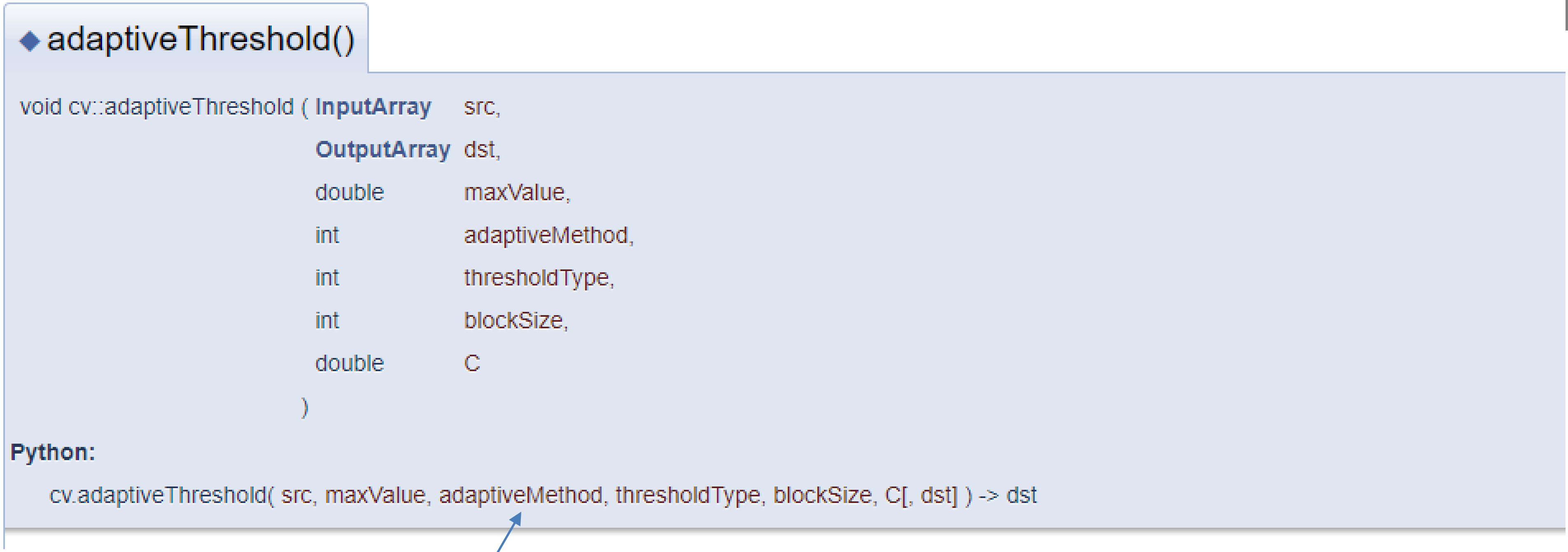
Currently, the Otsu's and Triangle methods are implemented only for 8-bit single-channel images.

Parameters

src	input array (multiple-channel, 8-bit or 32-bit floating point).
dst	output array of the same size and type and the same number of channels as src.
thresh	threshold value.
maxval	maximum value to use with the **THRESH_BINARY** and **THRESH_BINARY_INV** thresholding types.
type	thresholding type (see **ThresholdTypes**).

Returns

the computed threshold value if Otsu's or Triangle methods used.



◆ adaptiveThreshold()

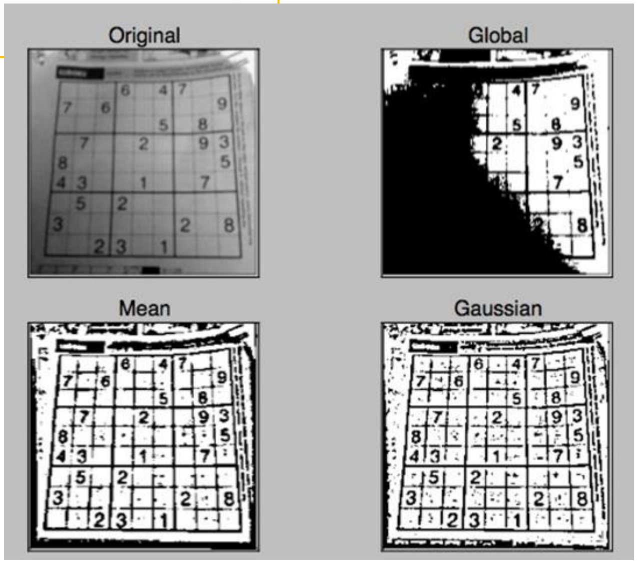
```
void cv::adaptiveThreshold ( InputArray   src,
                             OutputArray  dst,
                             double       maxValue,
                             int          adaptiveMethod,
                             int          thresholdType,
                             int          blockSize,
                             double       C
                           )
```

Python:

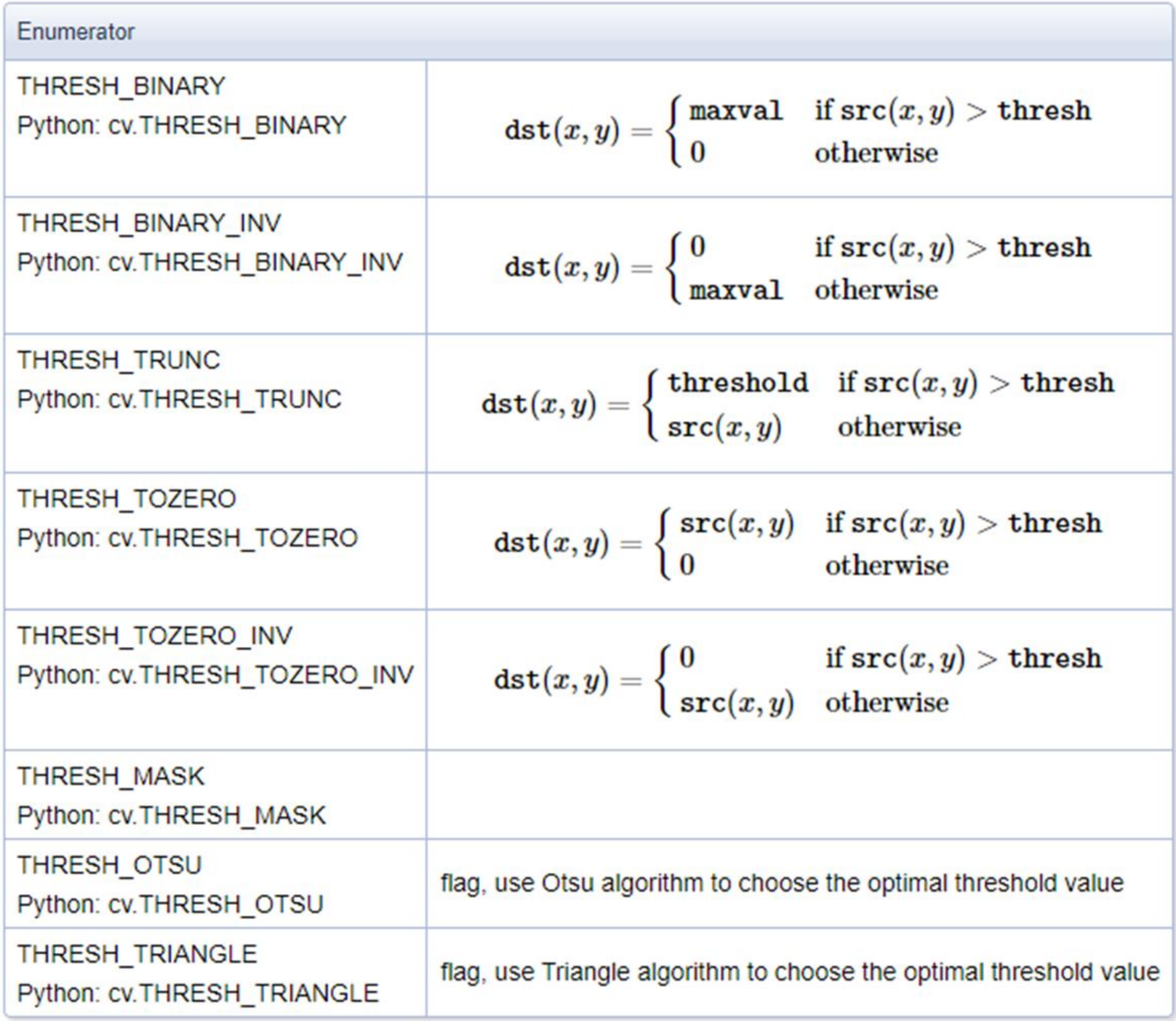
```
cv.adaptiveThreshold( src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst] ) -> dst
```

Adaptive Method

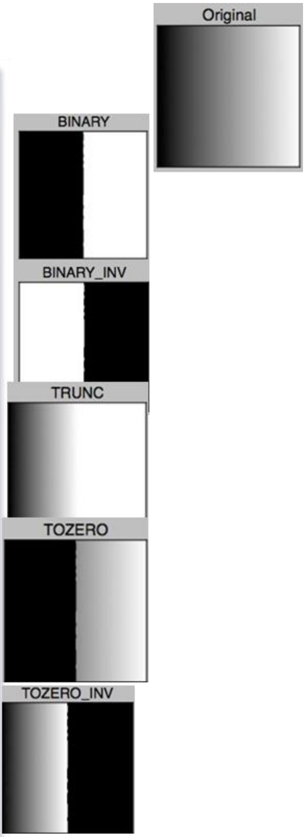
- cv2.ADAPTIVE_THRESH_MEAN_C : 주변영역의 평균값으로 결정
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C :

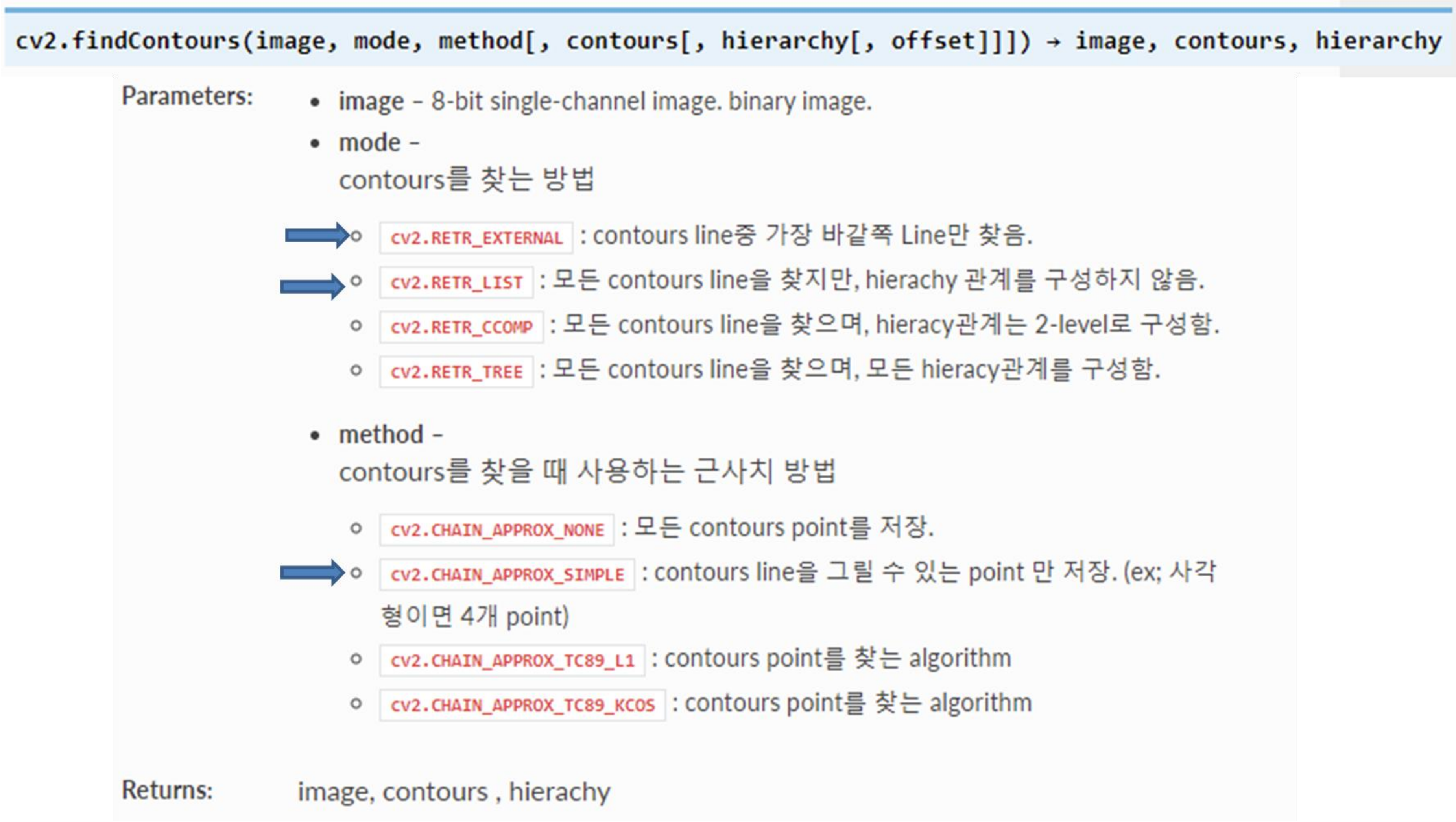


Threshold types



	Enumerator	
0	THRESH_BINARY Python: cv.THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
1	THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
2	THRESH_TRUNC Python: cv.THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
3	THRESH_TOZERO Python: cv.THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
4	THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
	THRESH_MASK Python: cv.THRESH_MASK	
	THRESH_OTSU Python: cv.THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
	THRESH_TRIANGLE Python: cv.THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value



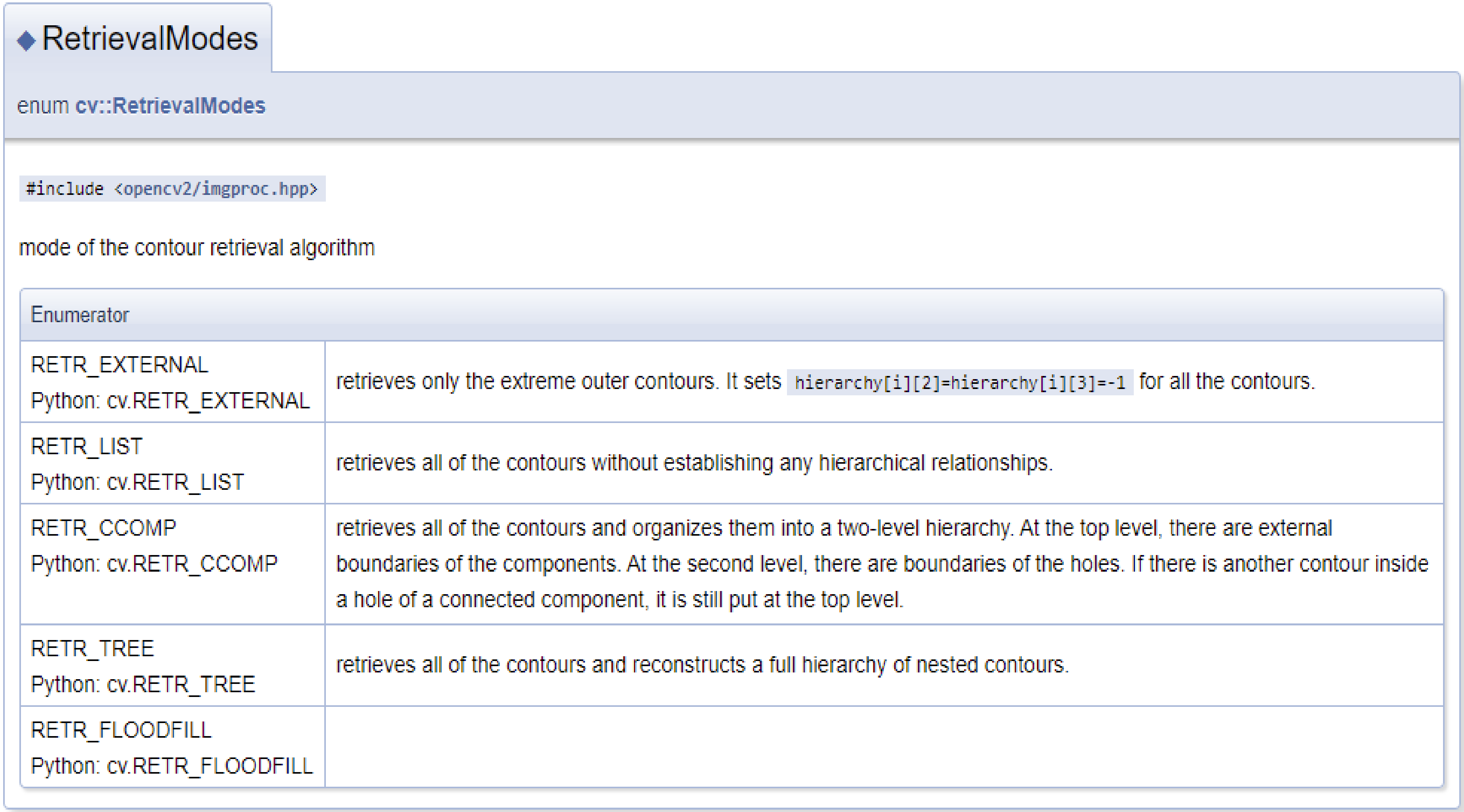


`cv2.findContours(image, mode, method[, contours[, hierarchy[, offset]]]) → image, contours, hierarchy`

Parameters:

- **image** – 8-bit single-channel image. binary image.
- **mode** –
  contours를 찾는 방법
  - `cv2.RETR_EXTERNAL` : contours line중 가장 바깥쪽 Line만 찾음.
  - `cv2.RETR_LIST` : 모든 contours line을 찾지만, hierachy 관계를 구성하지 않음.
  - `cv2.RETR_CCOMP` : 모든 contours line을 찾으며, hieracy관계는 2-level로 구성함.
  - `cv2.RETR_TREE` : 모든 contours line을 찾으며, 모든 hieracy관계를 구성함.
- **method** –
  contours를 찾을 때 사용하는 근사치 방법
  - `cv2.CHAIN_APPROX_NONE` : 모든 contours point를 저장.
  - `cv2.CHAIN_APPROX_SIMPLE` : contours line을 그릴 수 있는 point 만 저장. (ex; 사각형이면 4개 point)
  - `cv2.CHAIN_APPROX_TC89_L1` : contours point를 찾는 algorithm
  - `cv2.CHAIN_APPROX_TC89_KCOS` : contours point를 찾는 algorithm

Returns: image, contours , hierachy



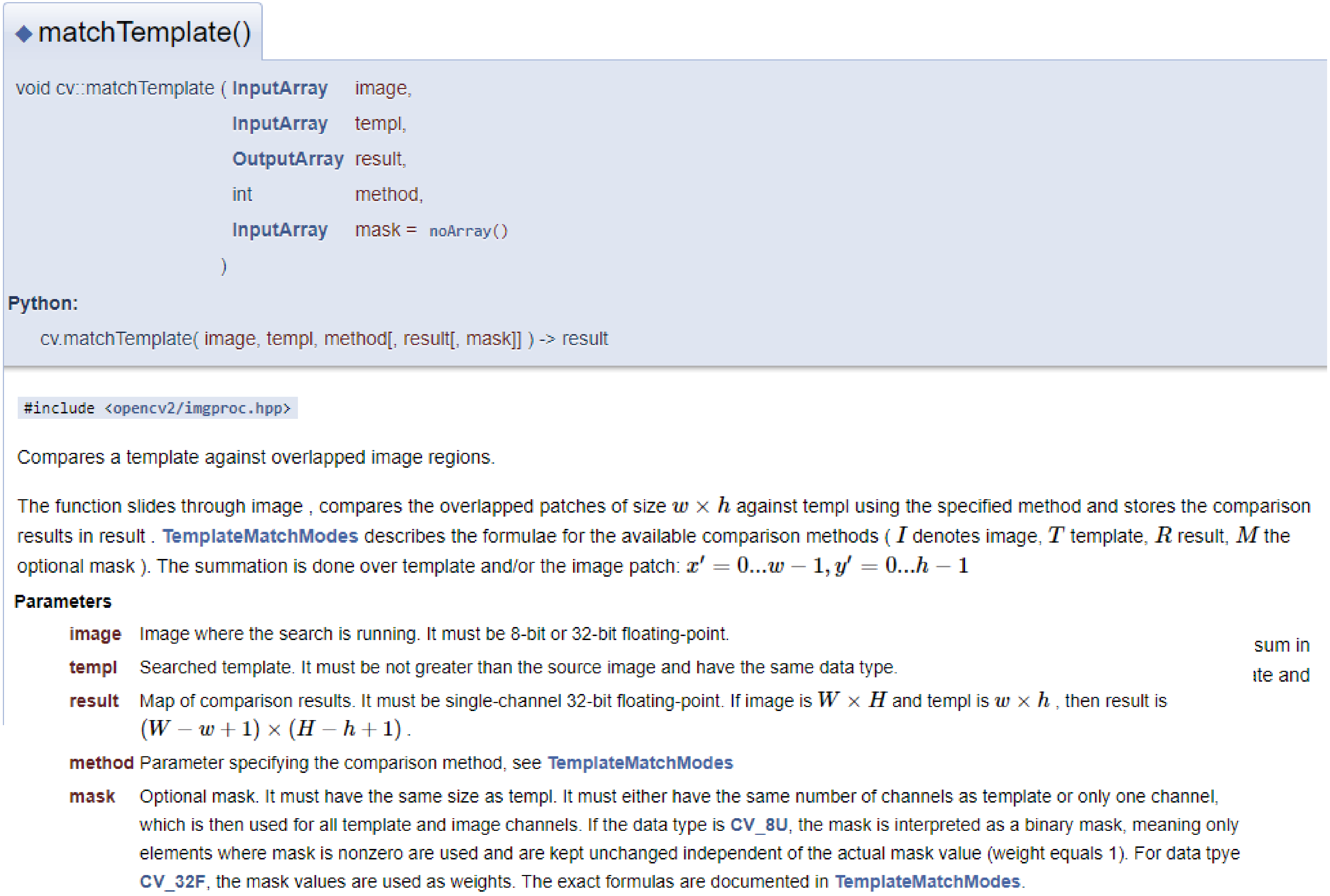
◆ RetrievalModes

enum **cv::RetrievalModes**

```
#include <opencv2/imgproc.hpp>
```

mode of the contour retrieval algorithm

Enumerator	
RETR_EXTERNAL Python: cv.RETR_EXTERNAL	retrieves only the extreme outer contours. It sets `hierarchy[i][2]=hierarchy[i][3]=-1` for all the contours.
RETR_LIST Python: cv.RETR_LIST	retrieves all of the contours without establishing any hierarchical relationships.
RETR_CCOMP Python: cv.RETR_CCOMP	retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
RETR_TREE Python: cv.RETR_TREE	retrieves all of the contours and reconstructs a full hierarchy of nested contours.
RETR_FLOODFILL Python: cv.RETR_FLOODFILL	



◆ matchTemplate()

```
void cv::matchTemplate ( InputArray   image,
                         InputArray   templ,
                         OutputArray  result,
                         int          method,
                         InputArray   mask = noArray()
                       )
```

Python:

```
cv.matchTemplate( image, templ, method[, result[, mask]] ) -> result
```

```
#include <opencv2/imgproc.hpp>
```

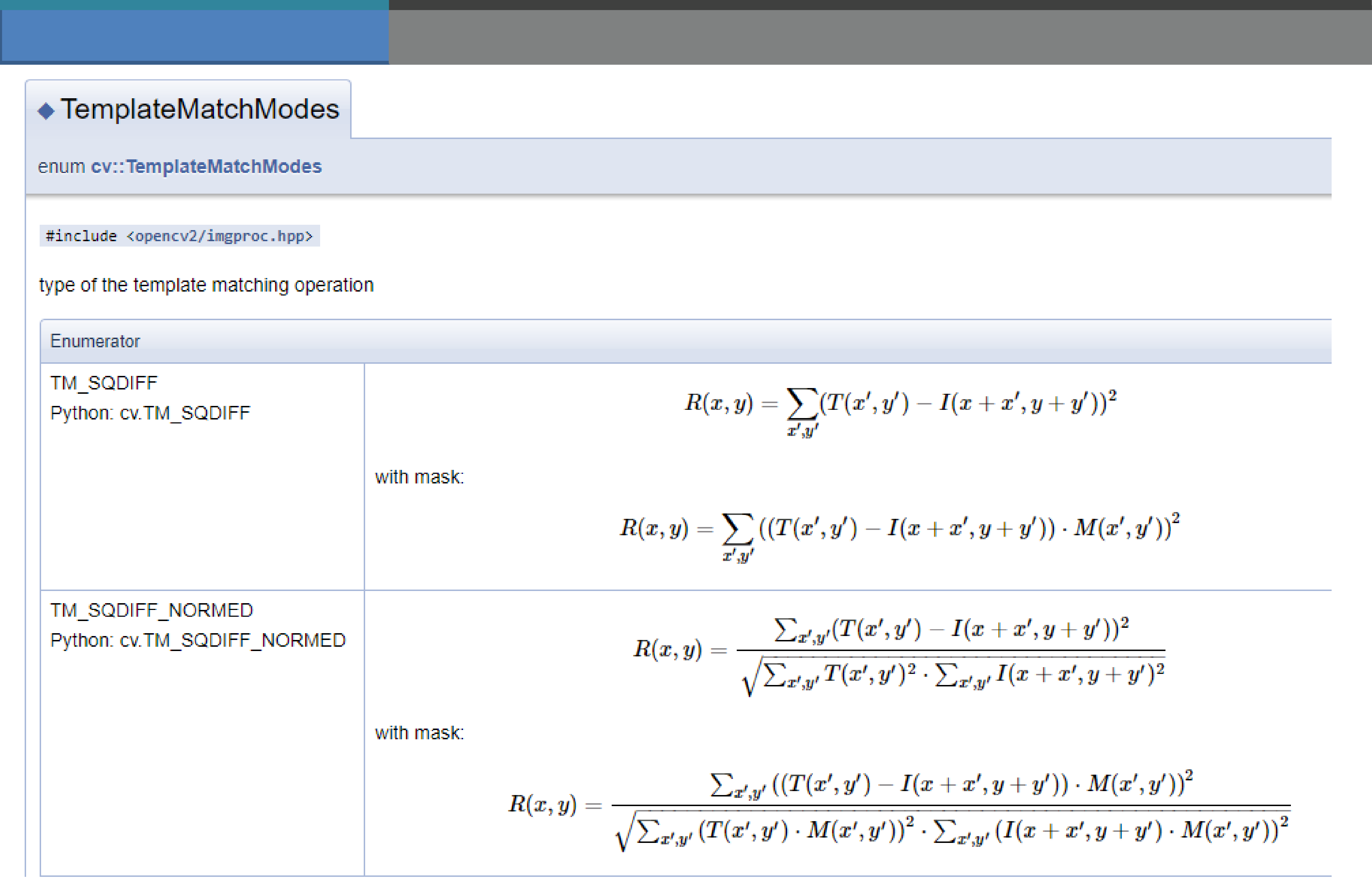
Compares a template against overlapped image regions.

The function slides through image , compares the overlapped patches of size $w \times h$ against templ using the specified method and stores the comparison results in result . **TemplateMatchModes** describes the formulae for the available comparison methods (I denotes image, T template, R result, M the optional mask). The summation is done over template and/or the image patch: $x' = 0...w-1, y' = 0...h-1$

Parameters

- **image** Image where the search is running. It must be 8-bit or 32-bit floating-point.
- **templ** Searched template. It must be not greater than the source image and have the same data type.
- **result** Map of comparison results. It must be single-channel 32-bit floating-point. If image is $W \times H$ and templ is $w \times h$, then result is $(W-w+1) \times (H-h+1)$.
- **method** Parameter specifying the comparison method, see **TemplateMatchModes**
- **mask** Optional mask. It must have the same size as templ. It must either have the same number of channels as template or only one channel, which is then used for all template and image channels. If the data type is **CV_8U**, the mask is interpreted as a binary mask, meaning only elements where mask is nonzero are used and are kept unchanged independent of the actual mask value (weight equals 1). For data tpye **CV_32F**, the mask values are used as weights. The exact formulas are documented in **TemplateMatchModes**.

sum in
ite and



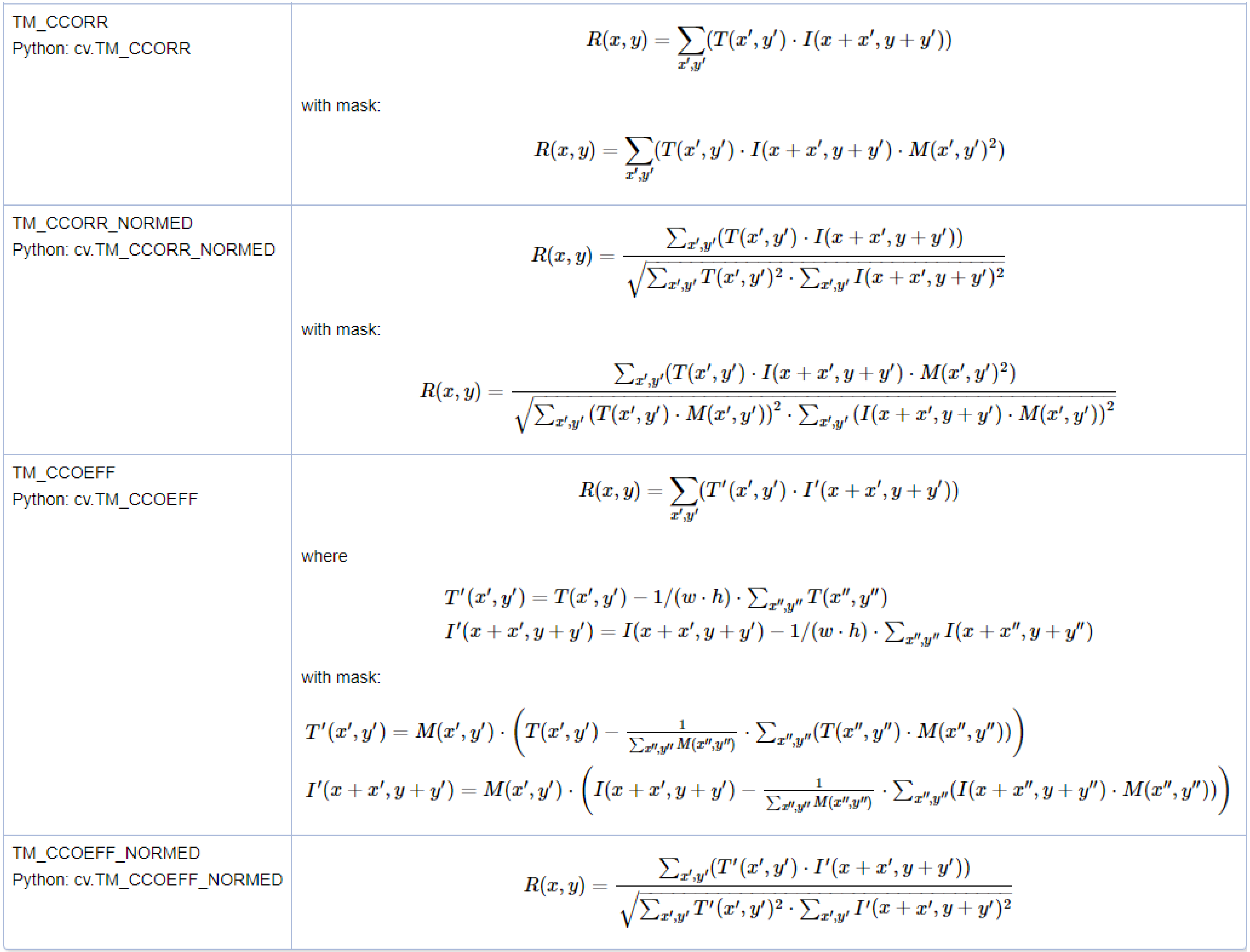
◆ TemplateMatchModes

enum **cv::TemplateMatchModes**

```
#include <opencv2/imgproc.hpp>
```

type of the template matching operation

Enumerator	
TM_SQDIFF Python: cv.TM_SQDIFF	$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$$ with mask: $$R(x,y)=\sum_{x',y'}\left((T(x',y')-I(x+x',y+y'))\cdot M(x',y')\right)^2$$
TM_SQDIFF_NORMED Python: cv.TM_SQDIFF_NORMED	$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$ with mask: $$R(x,y)=\frac{\sum_{x',y'}\left((T(x',y')-I(x+x',y+y'))\cdot M(x',y')\right)^2}{\sqrt{\sum_{x',y'}\left(T(x',y')\cdot M(x',y')\right)^2\cdot\sum_{x',y'}\left(I(x+x',y+y')\cdot M(x',y')\right)^2}}$$

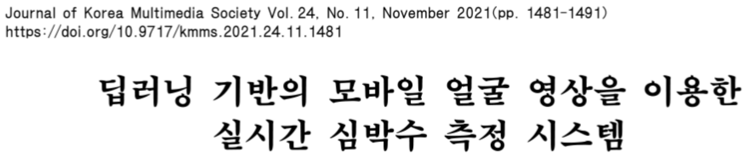


TM_CCORR Python: cv.TM_CCORR	$R(x,y)= \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$ with mask: $R(x,y)= \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y') \cdot M(x',y')^2)$
TM_CCORR_NORMED Python: cv.TM_CCORR_NORMED	$R(x,y)= \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$ with mask: $R(x,y)= \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y') \cdot M(x',y')^2)}{\sqrt{\sum_{x',y'} \left(T(x',y') \cdot M(x',y') \right)^2 \cdot \sum_{x',y'} \left(I(x+x',y+y') \cdot M(x',y') \right)^2}}$
TM_CCOEFF Python: cv.TM_CCOEFF	$R(x,y)= \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))$ where $T'(x',y')=T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$ $I'(x+x',y+y')=I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$ with mask: $T'(x',y')=M(x',y') \cdot \left(T(x',y') - \frac{1}{\sum_{x'',y''} M(x'',y'')} \cdot \sum_{x'',y''} (T(x'',y'') \cdot M(x'',y'')) \right)$ $I'(x+x',y+y')=M(x',y') \cdot \left(I(x+x',y+y') - \frac{1}{\sum_{x'',y''} M(x'',y'')} \cdot \sum_{x'',y''} (I(x+x'',y+y'') \cdot M(x'',y'')) \right)$
TM_CCOEFF_NORMED Python: cv.TM_CCOEFF_NORMED	$R(x,y)= \frac{ \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y')) }{ \sqrt{\sum_{x',y'}T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2} }$

안면 이미지 데이터를 이용한 실시간 생체징후 측정시스템

Real-time Vital Signs Measurement System using Facial Image Data

- https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002699582



Journal of Korea Multimedia Society Vol. 24, No. 11, November 2021(pp. 1481-1491)
https://doi.org/10.9717/kmms.2021.24.11.1481

딥러닝 기반의 모바일 얼굴 영상을 이용한
실시간 심박수 측정 시스템

- https://koreascience.kr/article/JAKO202102539948593.pdf