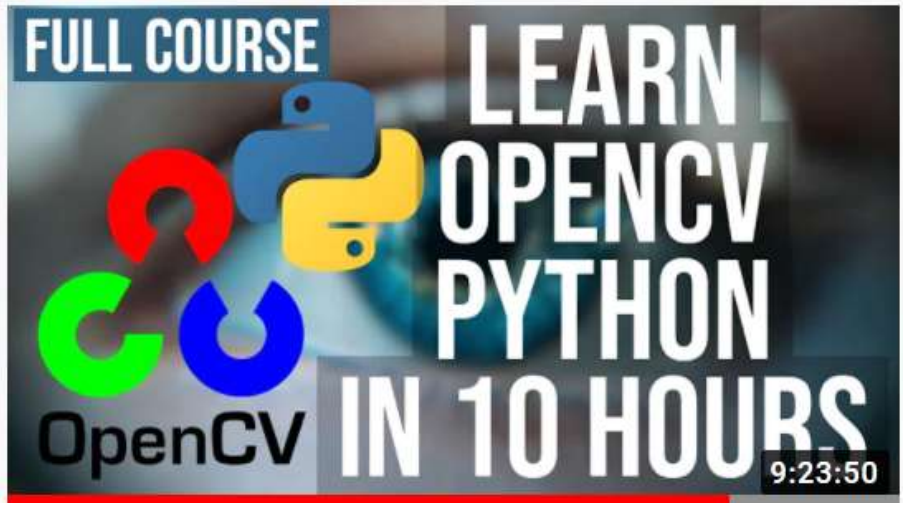
OpenCV Python Tutorial For Beginners (10 hrs)

https://www.youtube.com/watch?v=N81PCpADwKQ&t=27645s

https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K



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

OpenCV-Python Tutorials

OpenCV was started at Intel in 1999 by Gary Bradsky

OpenCV documentation index

1. Introduction to OpenCV

- OpenCV is an image processing library created by Intel and later supported by Willow Garage and now maintained by Itseez.
- Available on Mac, Windows, Linux.
- Works in C, C++, and Python.
- Open Source and free.
- Easy to use and install.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

What is Numpy

- Numpy is a highly optimized library for numerical operations.
- Array structure is important because digital images are 2D arrays of P-I-X-E-L-S
- All the OpenCV array structures are converted to-and-from Numpy arrays.
- You can use more convenient indexing system rather than using for loops.

NumPy 能夠支援 Python 進行大量的陣列與矩陣運算，同時也提供了大量的數學 library 提升 Python 運算速度。因此未來在進行機器學習的時候，我們將不可避免地大量使用到 NumPy 的功能。

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)          # [1 2 3 4 5]
type(arr)           # numpy.ndarray
```

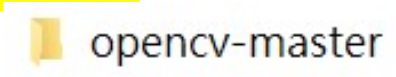
這裡你可能會問，為什麼我們不能直接操作 Python list，而要建立一個新的 NumPy array 呢？因為 NumPy array 的相對優勢在於

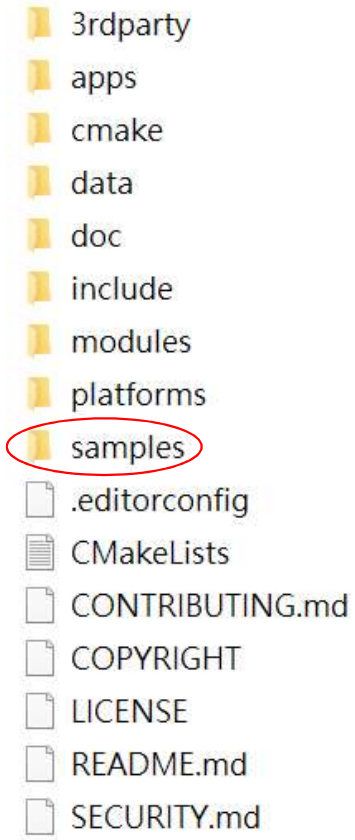
1. 使用更少的儲存空間
2. 有更快的速度
3. 以及有 NumPy 與 SciPy 等 library 各種優化過、更強大的功能的支援

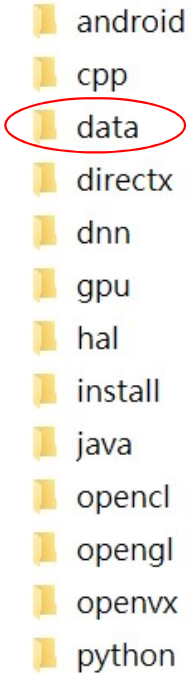
2. How to Install OpenCV for Python on Windows 10 (skip)

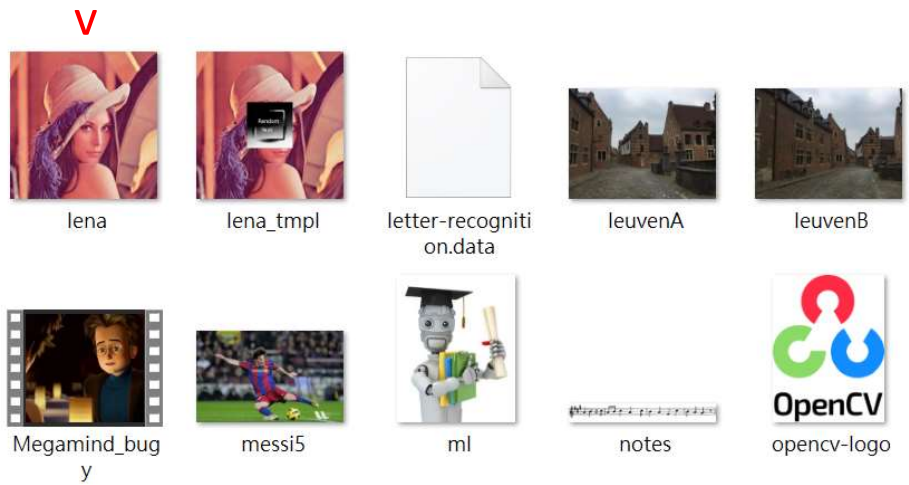
3. How to Read, Write, Show Images in OpenCV

https://github.com/opencv/opencv to download









https://drive.google.com/file/d/14-6CatqE8o0TBytyRZk2y3dsDzGMcynK/view?usp=sharing
(download images, videos here)

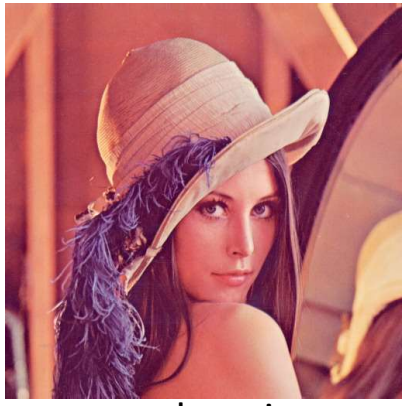
ex1.py

```python
import cv2

img = cv2.imread('lena.jpg',0)  # grayscale
print(img.shape)
# (512,512)
"""
array([[163, 162, 161, ..., 170, 154, 130],
       [162, 162, 162, ..., 173, 155, 126],
       [162, 162, 163, ..., 170, 155, 128],
       ...,
       [ 43,  42,  51, ..., 103, 101,  99],
       [ 41,  42,  55, ..., 103, 105, 106],
       [ 42,  44,  57, ..., 102, 106, 109]], dtype=uint8)
"""

img = cv2.imread('lena.jpg',1)  # color (default)
print(img.shape)
# (512,512,3)

img = cv2.imread('lena.jpg',-1) # as is
print(img.shape)
# (512,512,3)

cv2.imshow('lena image',img)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows()

cv2.imwrite('lena_copy.png',img)
```

Read an image

cv2.imread() Second argument is a flag which specifies the way image should be read.

flag	integer value	description
cv2.IMREAD_COLOR	1	Loads a color image.
cv2.IMREAD_GRAYSCALE	0	Loads image in grayscale mode
cv2.IMREAD_UNCHANGED	-1	Loads image as such including alpha channel



lena.jpg

* Array:
(512,512)
(512,512,3)

Syntax: `cv2.imread(/path/to/image, flag)`

Syntax: `cv2.imshow(window_name, image)`

ex2.py

```python
import cv2

def test():
    img = cv2.imread('lena.jpg')
    while True:
        cv2.imshow('lena image', img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print('I am done')
            cv2.destroyAllWindows()
            break

if __name__ == '__main__':
    test()
```

1. indent 縮排
2. bitwise AND (&) 0&0=0; 0&1=0 1&0=0; 1&1=1

cv2.waitKey([delay]) → retval

Parameters:

delay – Delay in milliseconds. 0 is the special value that means “forever”.

It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

ord(' ')可以將字符轉化為對應的整數(ASCII碼)

`cv2.waitKey(1) & 0xFF` &

cv2.waitKey(1) 與 0xFF（1111 1111）相 ‘與’ 是因為cv2.waitKey(1) 的返回值不止8位(32)，但是只有後8位實際有效，為避免產干擾，通過 ‘與’ 操作將其餘位置設0。

cv2.waitKey() returns a 32 Bit integer value (might be dependent on the platform). The key input is in ASCII which is an 8 Bit integer value. So you only care about these 8 bits and want all other bits to be 0.

if __name__ == '__main__'的意思是：

當.py檔案被直接執行時，if __name__ == '__main__'之下的程式碼塊將被執行；當.py檔案以模組形式被匯入時，if __name__ == '__main__'之下的程式碼塊不被執行。

通俗的理解__name__ == '__main__'：假如你叫小明.py，在朋友眼中，你是小明(__name__ == '小明')；在你自己眼中，你是你自己(__name__ == '__main__')。

ex3.py

```python
import cv2

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
# cap = cv2.VideoCapture('vtest.avi')
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS)) # = 0 if cam 0 is used
codec = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', codec, 20.0, (width, height))

print(cap.isOpened())  # True
while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        out.write(frame) # save color video

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('gray video', frame) # show gray video

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    else:
       break

cap.release()
out.release()
cv2.destroyAllWindows()
```

4. How to Read, Write, Show Videos from Camera in OpenCV

output_format: 'XVID',
codec used in VideoWriter
when saving video to file.

參數cv2.CAP_DSHOW是微軟特有的用於防止釋放攝像頭時的[WARN:0] terminating async callback警告

The cv2.CAP_DSHOW is a flag passed as part of the open call, there are many others you can pass, and this CAP_DSHOW is Microsoft specific.

ex4.py

5. Draw geometric shapes on images

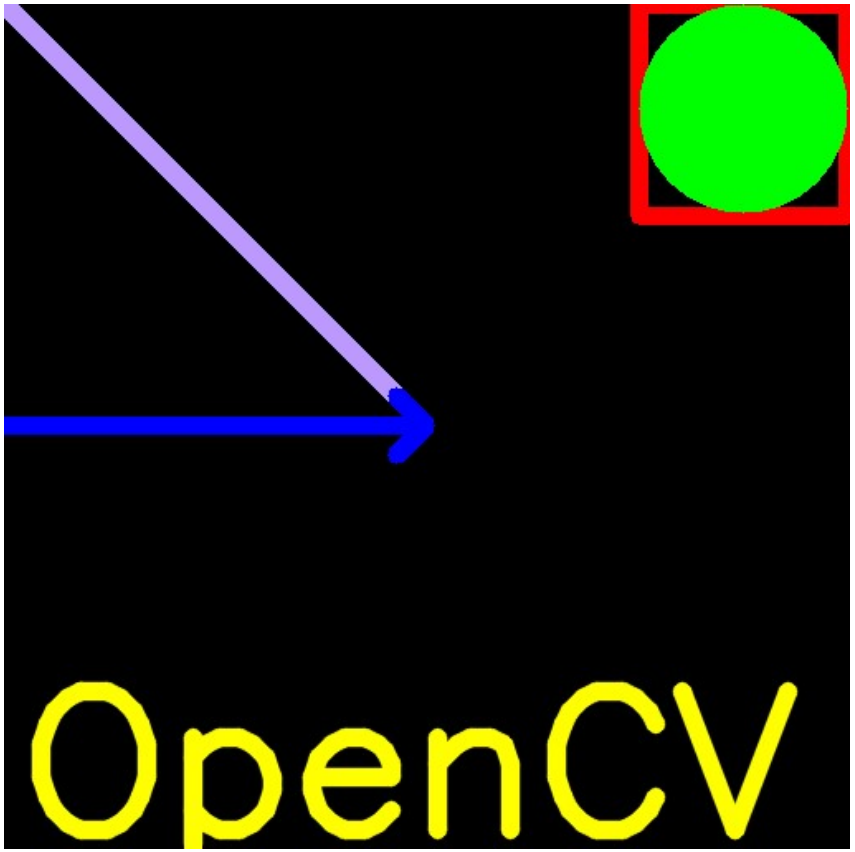
```
import numpy as np
import cv2

img = cv2.imread('lena.jpg',1)   # color (default)
# img = np.zeros([512, 512, 3], np.uint8)   # black image

img = cv2.line(img, (0, 0), (255, 255), (255, 153, 187), 10) # color=(b,g,r)
img = cv2.arrowedLine(img, (0, 255), (255, 255), (255, 0, 0), 10)

img = cv2.rectangle(img, (384, 0), (510, 128), (0, 0, 255), 10)
img = cv2.circle(img, (447, 63), 63, (0, 255, 0), -1)   # fill in
font = cv2.FONT_HERSHEY_SIMPLEX
img = cv2.putText(img, 'OpenCV', (10, 500), font, 4, (0, 255, 255), 10, cv2.LINE_AA)

cv2.imshow('geomatric image',img)
cv2.waitKey(0)   # waits until a key is pressed
cv2.destroyAllWindows()
```

color (b,g,r)

bottom-left corner of the text

cv2.putText(image, 'OpenCV', org, font,
fontScale, color, thickness, cv2.LINE_AA)
(line types)


OpenCV


OpenCV

6. Setting Camera Parameters

cap.get(propId) : 取得影片的一些資訊，其中 propId 是屬性的 ID，可用的值是從 0 到 18，請參考 OpenCV 的文件。

https://docs.opencv.org/3.0-beta/modules/videoio/doc/reading_and_writing_video.html#videocapture-get

cap.set(propId, value) : Sets a property in the VideoCapture.

3 **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
4 **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.

```
cap.set(3, 1280)
cap.set(4, 720)     # HD
```

VGA = 640 x 480
HD = 1280 x 720
FHD = 1920 x 1080
4K = 3840 x 2160

0 , 1, ~ 18

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds or video capture timestamp.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by `retrieve()` .
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).
- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).
- **CV_CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
- **CV_CAP_PROP_WHITE_BALANCE** Currently not supported
- **CV_CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

ex5.py

```
import cv2
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))  # 640
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)) # 480

cap.set(3, 1280) # highest resolution
cap.set(4, 720)   # highest resolution
print(cap.get(3))
print(cap.get(4))

while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('gray video', frame) # show gray video

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
       break

cap.release()
cv2.destroyAllWindows()
```

7. Show Date and Time on Videos

ex6.py

```
import cv2
import datetime

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))  # 640
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)) # 480

while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:

        font = cv2.FONT_HERSHEY_SIMPLEX
        text = str(datetime.datetime.now())
        #text = 'Width:' +str(cap.get(3))+ ' Height:'+ str(cap.get(4))
        frame = cv2.putText(frame, text, (10, 50), font, 1, (0, 255, 255), 2,
                            cv2.LINE_AA)
        cv2.imshow('date time', frame) # show date & time

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    else:
       break

cap.release()
cv2.destroyAllWindows()
```

```
import datetime
text = str(datetime.datetime.now())
print(text)
```

2021-03-03 10:40:35.268917

8. Handle Mouse Events in OpenCV

ex7.py

```
import cv2
events = [i for i in dir(cv2) if 'EVENT' in i]
print(events)
print(len(events))
```

18 events

['EVENT_FLAG_ALTKEY', 'EVENT_FLAG_CTRLKEY', 'EVENT_FLAG_LBUTTON', 'EVENT_FLAG_MBUTTON', 'EVENT_FLAG_RBUTTON', 'EVENT_FLAG_SHIFTKEY', 'EVENT_LBUTTONDBLCLK', 'EVENT_LBUTTONDOWN', 'EVENT_LBUTTONUP', 'EVENT_MBUTTONDBLCLK', 'EVENT_MBUTTONDOWN', 'EVENT_MBUTTONUP', 'EVENT_MOUSEHWHEEL', 'EVENT_MOUSEMOVE', 'EVENT_MOUSEWHEEL', 'EVENT_RBUTTONDBLCLK', 'EVENT_RBUTTONDOWN', 'EVENT_RBUTTONUP']

cv2.setMouseCallback(windowName, onMouse[, param]) 滑鼠響應

: sets mouse handler for the specified window

onMouse: callback function for mouse events.

ex7.py

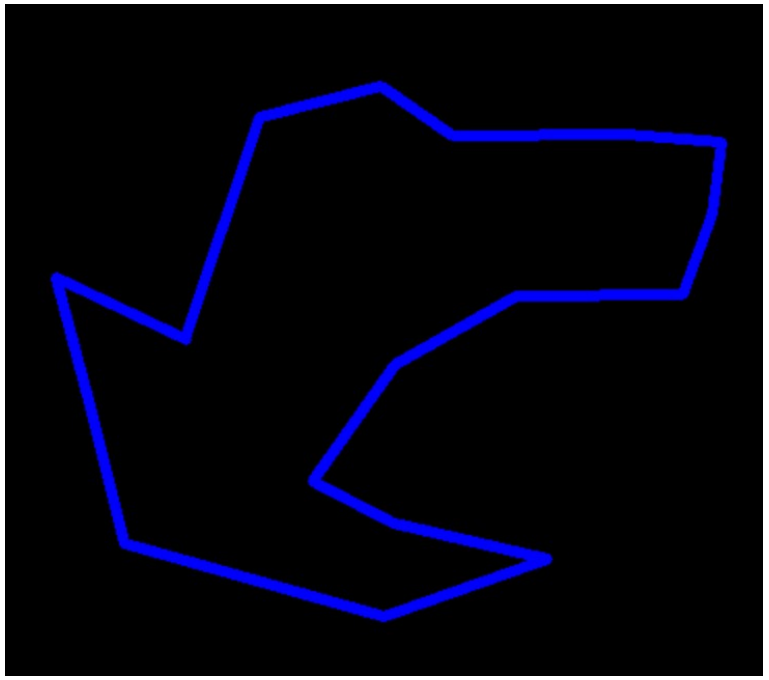
```python
import cv2

# events = [i for i in dir(cv2) if 'EVENT' in i]
# print(events)

def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x, ',', y)
        font = cv2.FONT_HERSHEY_SIMPLEX
        text = str(x)+','+str(y)
        cv2.putText(img, text, (x, y), font, 0.5, (255, 255, 0), 2,
                    cv2.LINE_AA)
        cv2.imshow('image',img)
    if event == cv2.EVENT_RBUTTONDOWN:
        blue = img[y, x, 0]
        green = img[y, x, 1]
        red = img[y, x, 2]
        font = cv2.FONT_HERSHEY_SIMPLEX
        text = str(blue)+','+str(green)+','+str(red)
        cv2.putText(img, text, (x, y), font, 0.5, (0, 255, 255), 2,
                    cv2.LINE_AA)
        cv2.imshow('image',img)

img = cv2.imread('lena.jpg')  # 512x512x3
cv2.imshow('image', img)

cv2.setMouseCallback('image', click_event)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

512x512x3

9. More Mouse Event Examples

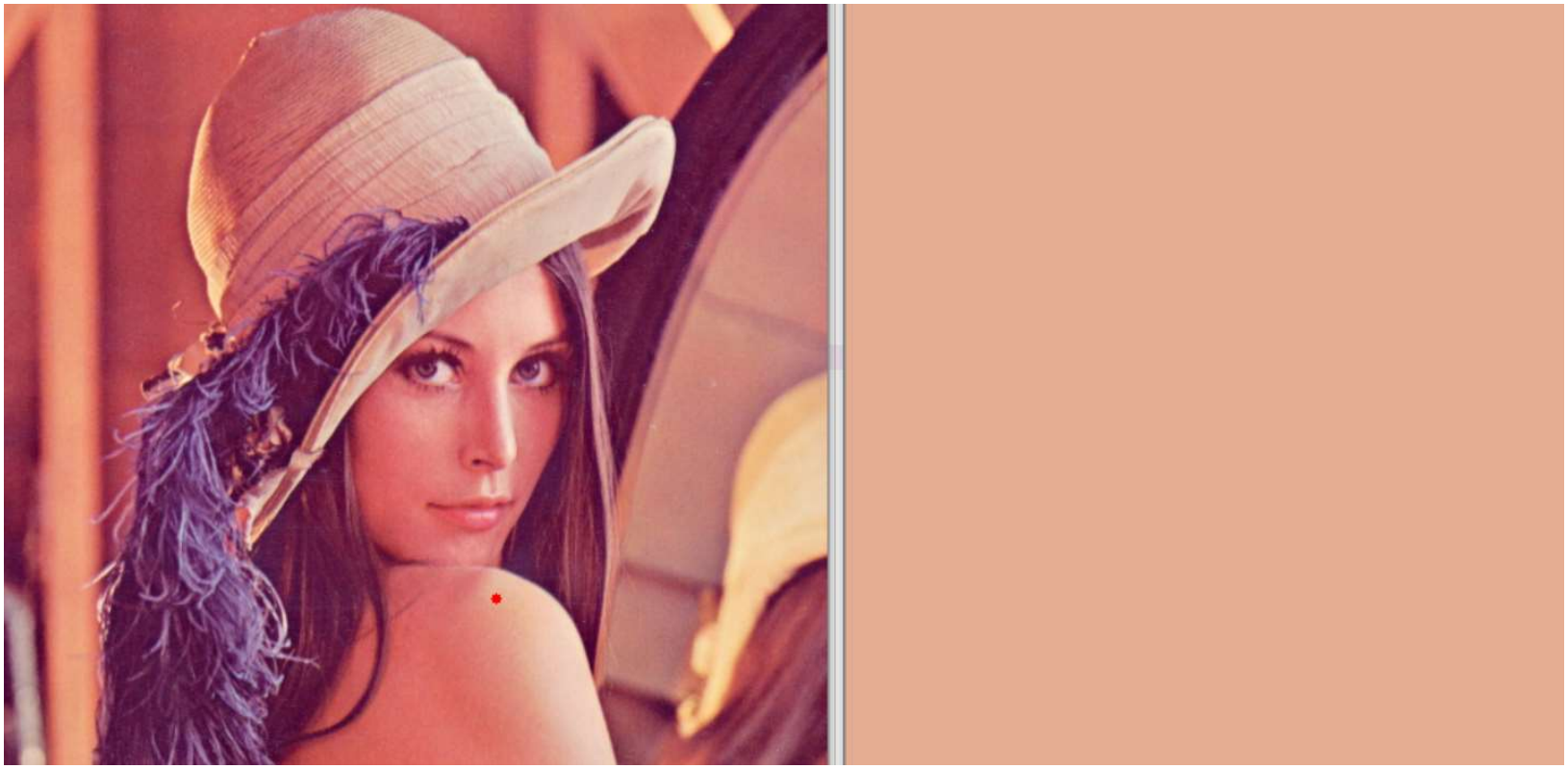
ex8.py : connecting points with line using mouse

```
import cv2
import numpy as np

def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(img, (x, y), 3, (0, 0, 255), -1)
        points.append((x, y))
        if len(points) >= 2:
            cv2.line(img, points[-2], points[-1], (255, 0, 0), 5)
        cv2.imshow('image',img)

img = np.zeros((512, 512, 3), np.uint8)  # 512x512x3
cv2.imshow('image', img)
points = []
cv2.setMouseCallback('image', click_event)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



ex9.py : extract color information and show it in another window

```
import cv2
import numpy as np

def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        blue = img[y, x, 0]
        green = img[y, x, 1]
        red = img[y, x, 2]
        cv2.circle(img, (x, y), 3, (0, 0, 255), -1)
        cv2.imshow('image', img)
        mycolorImage = np.zeros((512, 512, 3), np.uint8)  # black image
        mycolorImage[:] = [blue, green, red]
        cv2.imshow('color',mycolorImage)

img = cv2.imread('lena.jpg')
cv2.imshow('image', img)
cv2.setMouseCallback('image', click_event)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



10. cv.split, cv.merge, cv.resize, cv.add, cv.addWeighted, ROI

ex10.py

```
import cv2

img = cv2.imread('messi5.jpg') # 548x342
print(img.shape)  # (row, column, channel)
print(img.size)     # total no. of pixels
print(img.dtype)  # image dadatype

b,g,r = cv2.split(img)
img = cv2.merge((b,g,r))   # try (r,g,b)

cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
(342, 548, 3)
562248
uint8
```

import cv2 ex11.py ROI : region of interest

```
img = cv2.imread('messi5.jpg') # WxH: 548x342
print(img.shape)  # (row, column, channel)
print(img.size)   # total no. of pixels
print(img.dtype)  # image dadatype
```

```
(342, 548, 3)
562248
uint8
```

```
b,g,r = cv2.split(img)
img = cv2.merge((b,g,r))

ball = img[280:340, 330:390]   # ROI, see also
img[273:333, 100:160] =ball    cv2.selectROI

cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

ex12.py

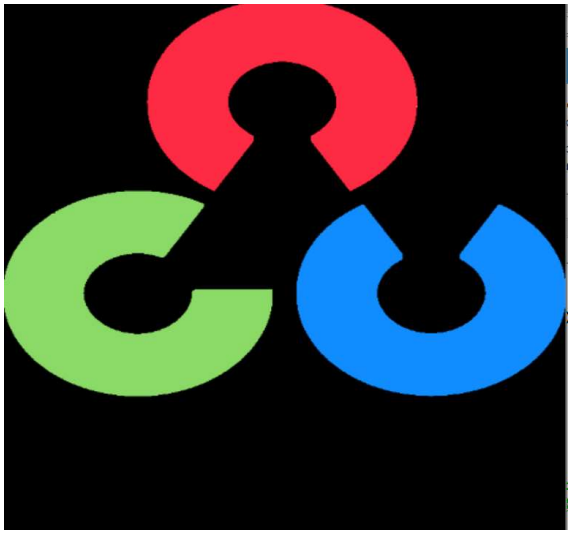
resize, add, addWeighted

```
import cv2

img = cv2.imread('messi5.jpg')  # 548x342
img2 = cv2.imread('opencv-logo.png')  # 600x794

img = cv2.resize(img, (512, 512))
img2 = cv2.resize(img2, (512, 512))

#dst = cv2.add(img, img2);
dst = cv2.addWeighted(img, 0.2, img2, 0.8, 0)  # image blending 圖像融合

cv2.imshow('image', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

addWeighted(src1, alpha, src2, beta, gamma, dst=None, dtype=None)

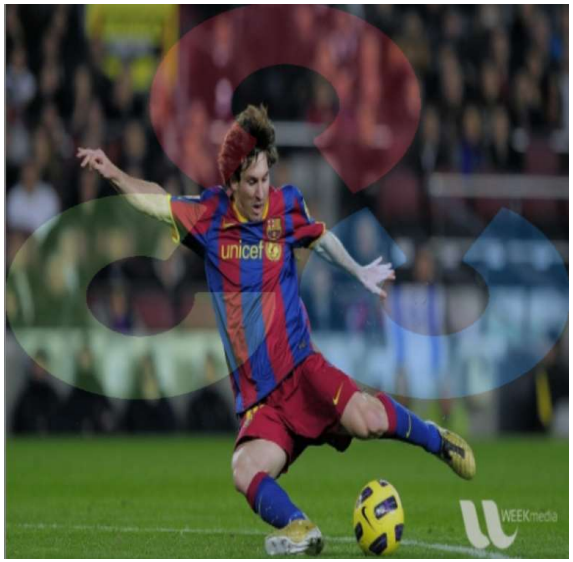
dst(I)=saturate(src1(I)∗alpha+src2(I)∗beta+gamma)

+



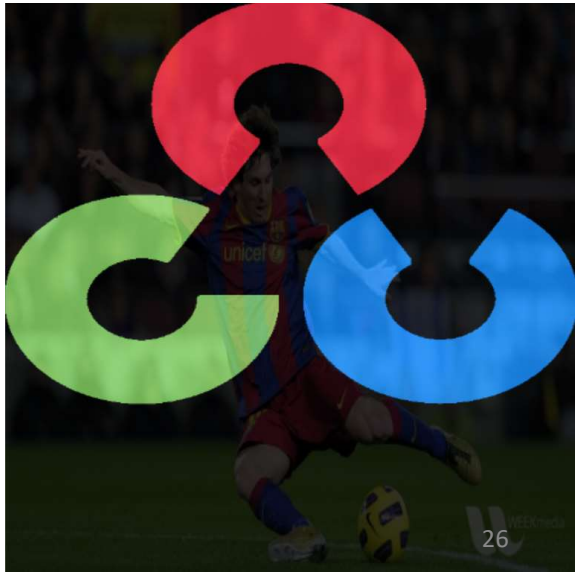
= ?

0.8
0.2
0



0.5
0.5
0

0.2
0.8
0



Python OpenCV cv2 Resize Image

https://pythonexamples.org/python-opencv-cv2-resize-image/

cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])

Original image

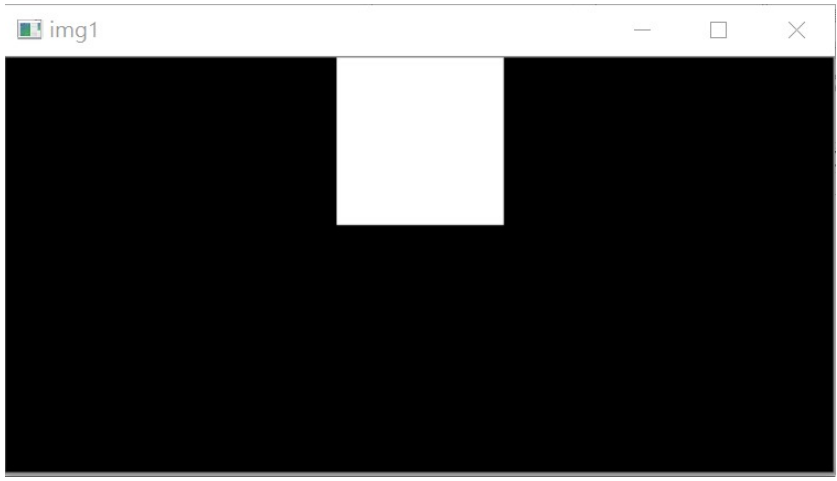
11. Bitwise Operations (bitwise AND, OR, NOT and XOR)

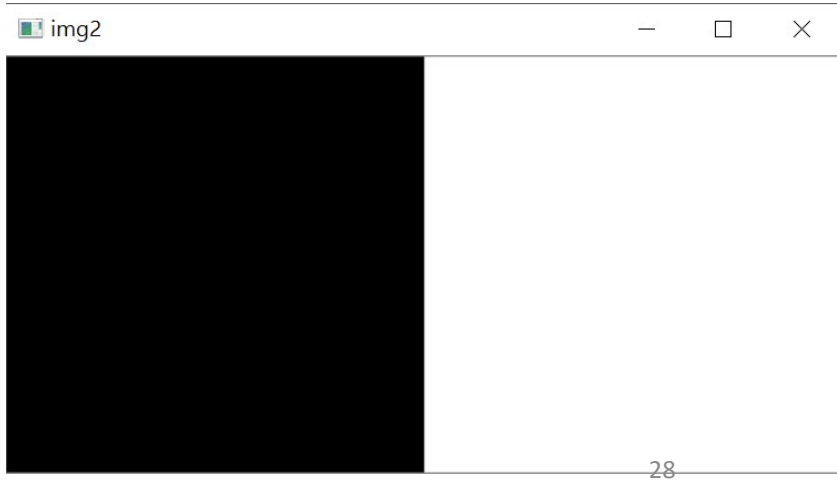
XOR: 互斥或閘 Exclusive-OR gate

Truth Table for the logic XOR

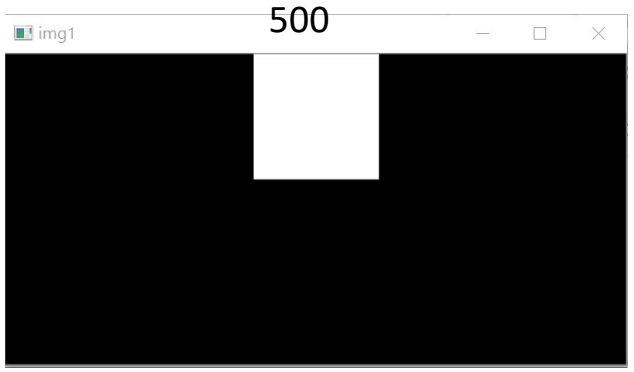
B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

Read as A **OR** B but not **BOTH** gives Q (odd)

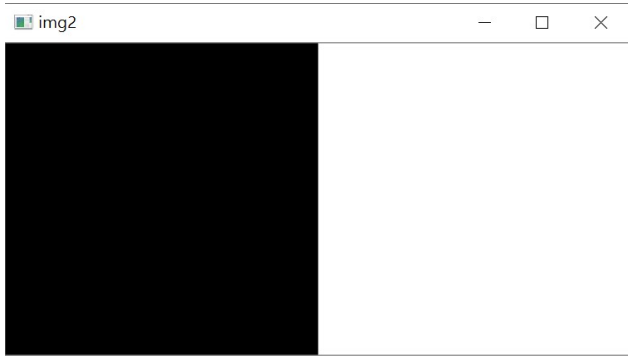




ex13.py



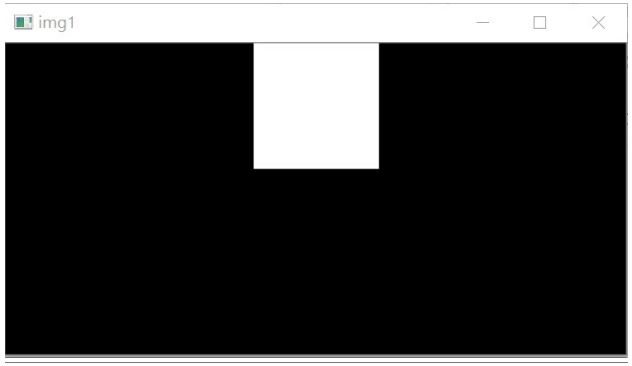

```
import cv2
import numpy as np

img1 = np.zeros((250, 500, 3), np.uint8)
img1 = cv2.rectangle(img1, (200, 0), (300, 100),(255, 255, 255), -1)
img2 = np.full((250, 500, 3), 255, np.uint8)
img2 = cv2.rectangle(img2, (0, 0), (250, 250), (0, 0, 0), -1)

bitAnd = cv2.bitwise_and(img2, img1)
bitOr = cv2.bitwise_or(img2, img1)
bitXor = cv2.bitwise_xor(img2, img1)
bitNot1 = cv2.bitwise_not(img1)
bitNot2 = cv2.bitwise_not(img2)
```

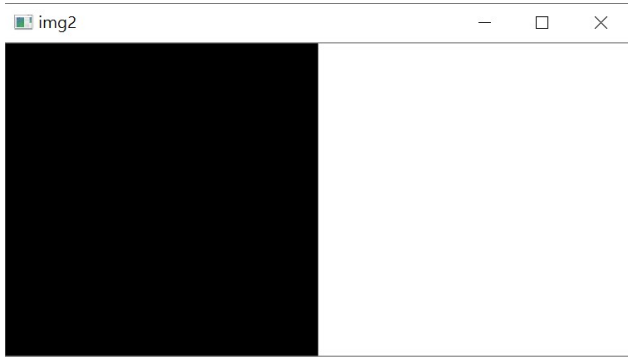


np. zeros, np. ones

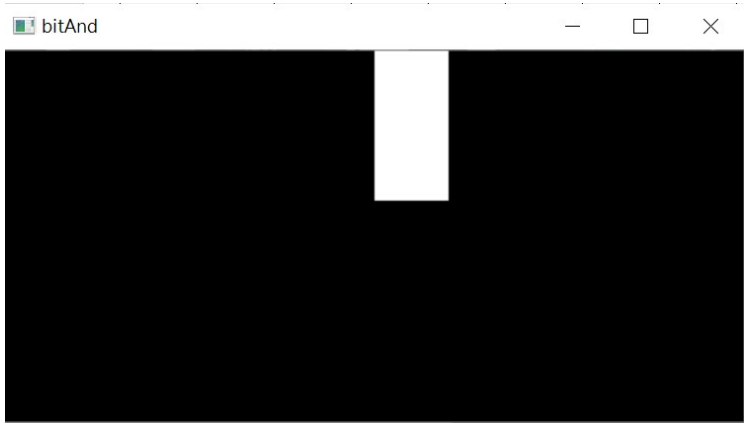
除了建立元素皆為 0 和 1 的陣列外，我們也可以用 np.full() 來指定特定值

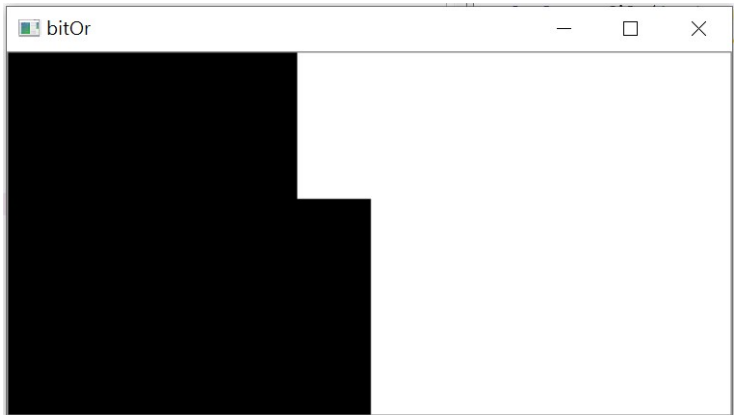
```
cv2.imshow('img1',img1)
cv2.imshow('img2',img2)
cv2.imshow('bitAnd',bitAnd)
cv2.imshow('bitOr',bitOr)
cv2.imshow('bitXor',bitXor)
cv2.imshow('bitNot1',bitNot1)
cv2.imshow('bitNot2',bitNot2)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

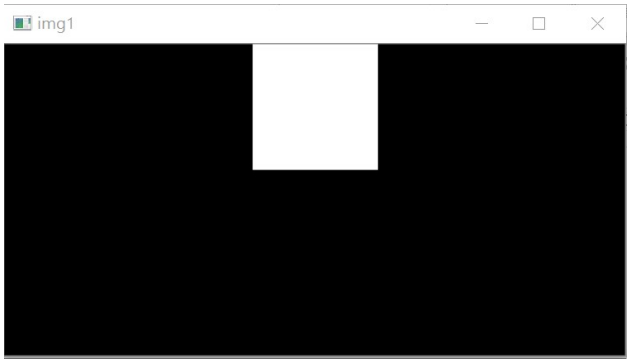


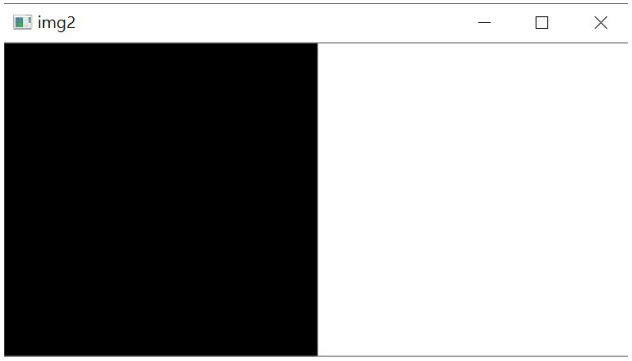


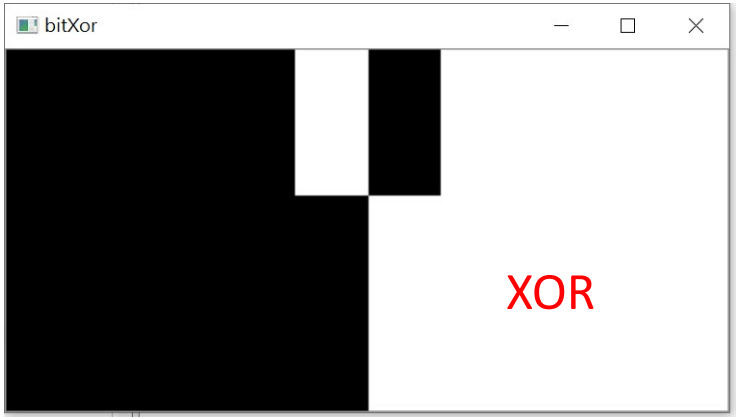
ex13.py

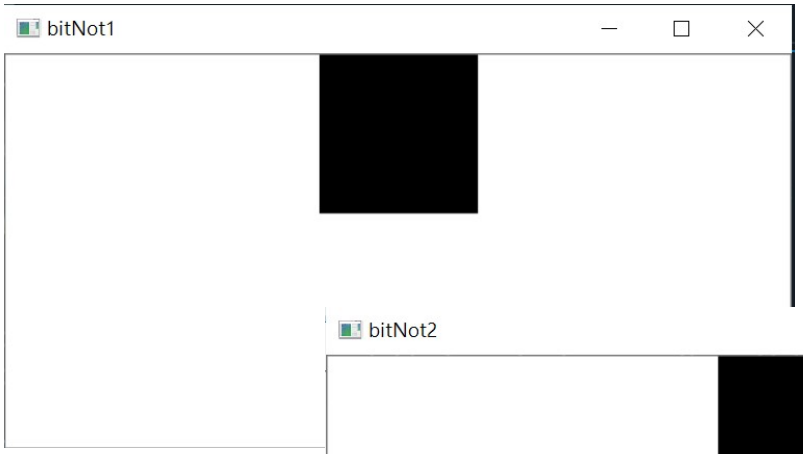


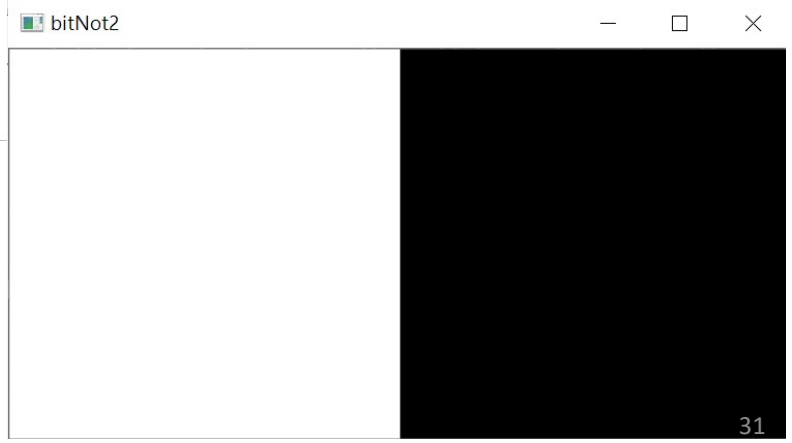












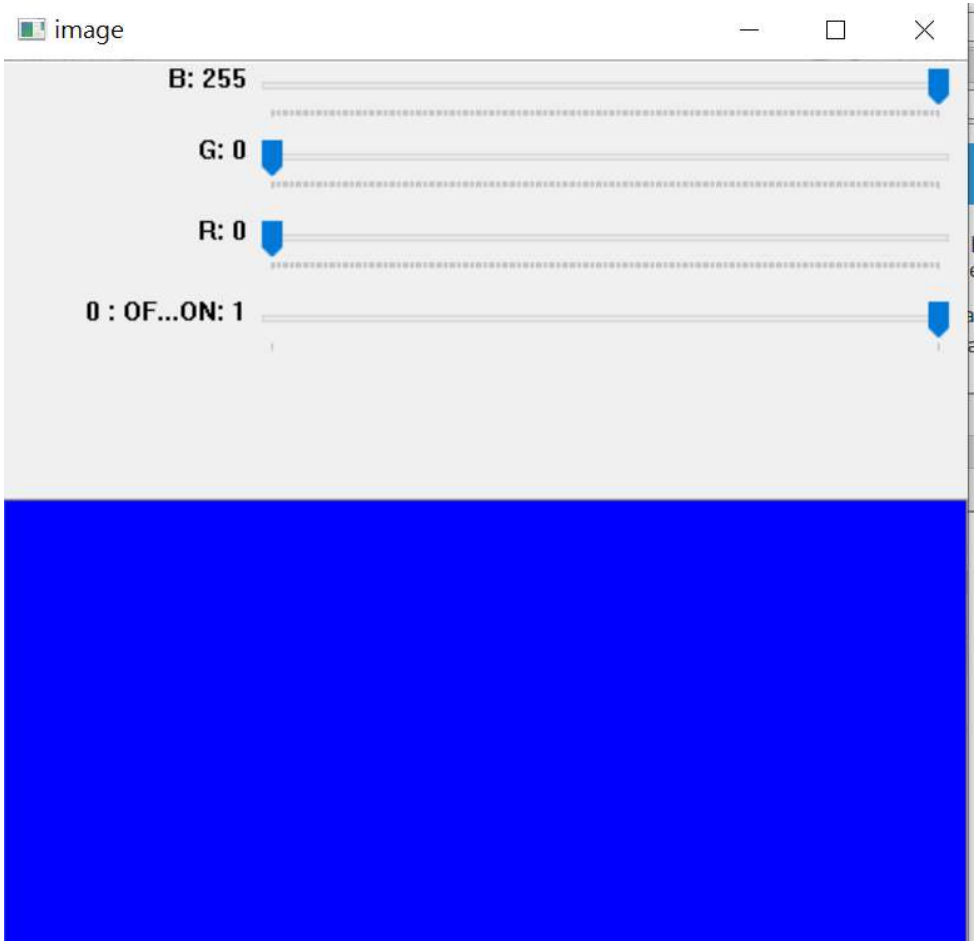
ex13.py

12. How to Bind Trackbar To OpenCV Windows

createTrackbar() takes five parameters:

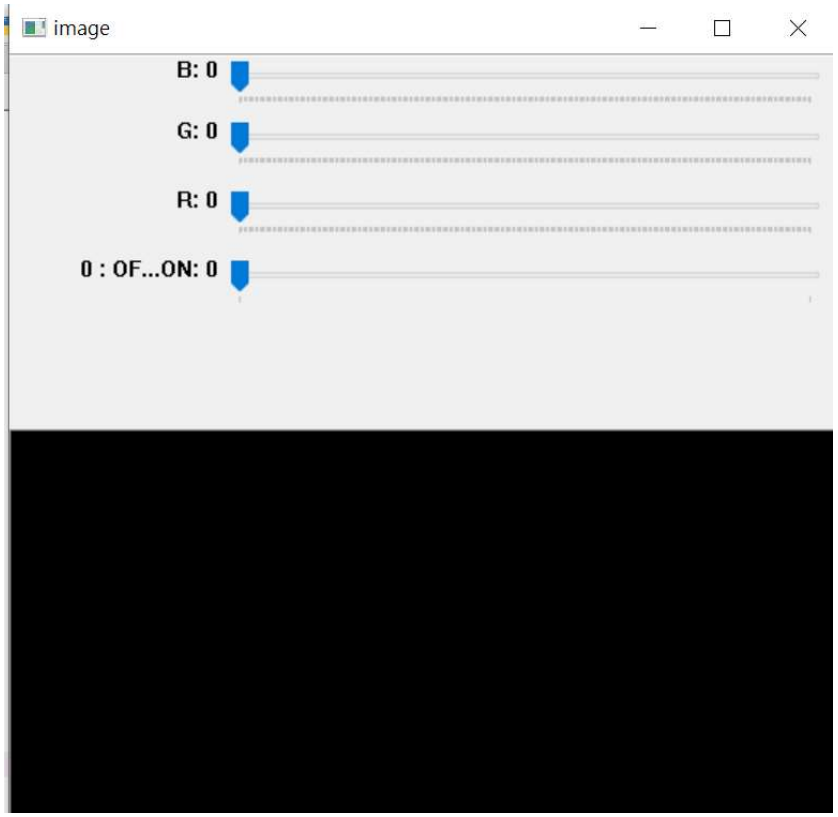
1 Name of the trackbar, which will also be used to retrieve data
2 Named window on which it will reside
3 Initial (default) value of the trackbar
4 The highest value of the trackbar
5 A callback function that will do nothing.
which is executed everytime trackbar value changes.
The callback function always has a default argument which is the trackbar position. In our case, function does nothing, so we simply pass.

getTrackbarPos() takes two values:
1 Name of the trackbar and,
2 Name of the Window where it resides
This returns the trackbar position between low to high value i.e 0 to 255.



ex14.py

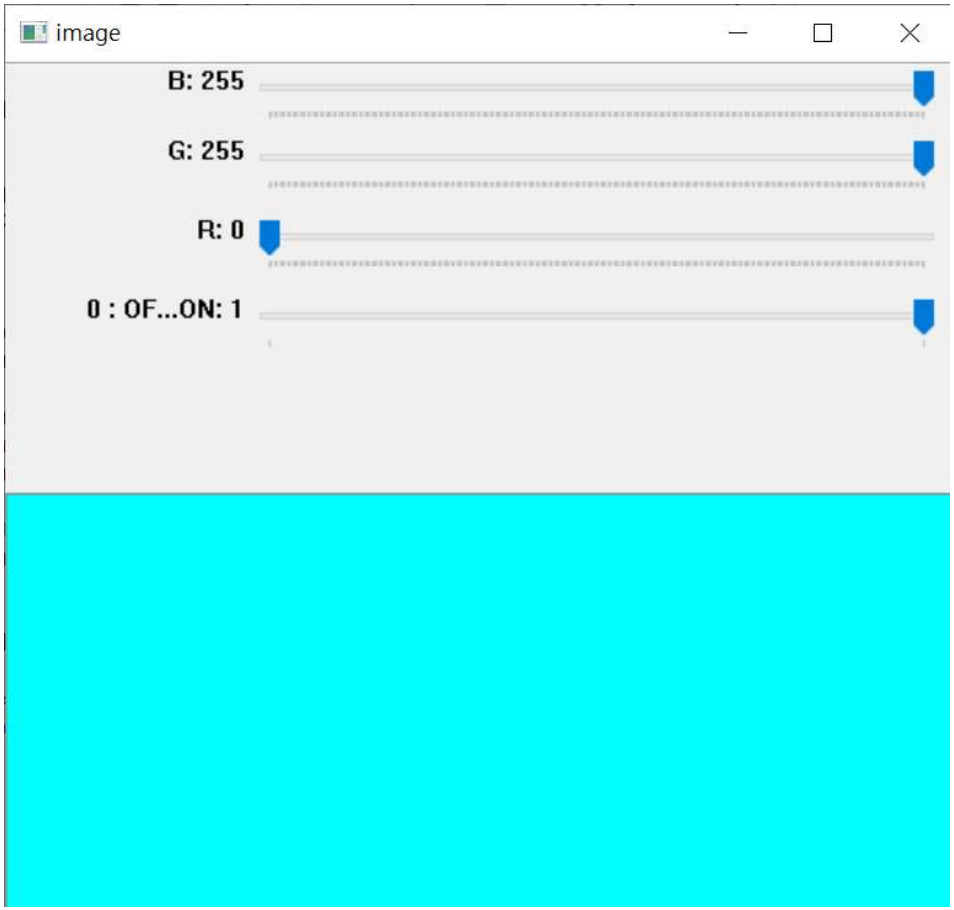
```
import numpy as np
import cv2 as cv

def nothing(x):
    print(x)
    #pass

# Create a black image, a window
img = np.zeros((300,512,3), np.uint8)
cv.namedWindow('image')

cv.createTrackbar('B', 'image', 0, 255, nothing)
cv.createTrackbar('G', 'image', 0, 255, nothing)
cv.createTrackbar('R', 'image', 0, 255, nothing)

switch = '0 : OFF\n1 : ON'
cv.createTrackbar(switch, 'image', 0, 1, nothing)
```



Another important application of trackbar is to use it as a button or switch.

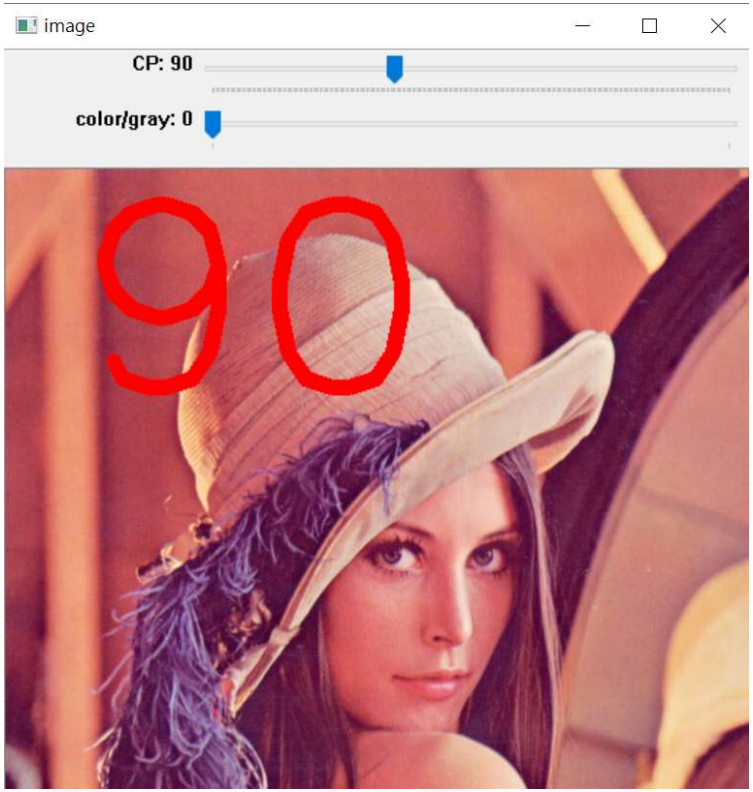
ex14.py

```
while(1):
        cv.imshow('image',img)
        k = cv.waitKey(1) & 0xFF
        if k == 27:    # esc key
            break

        b = cv.getTrackbarPos('B', 'image')
        g = cv.getTrackbarPos('G', 'image')
        r = cv.getTrackbarPos('R', 'image')
        s = cv.getTrackbarPos(switch, 'image')

        if s == 0:
          img[:] = 0
        else:
          img[:] = [b, g, r]

cv.destroyAllWindows()
```



ex15.py

```python
import cv2 as cv

def nothing(x):
    print(x)

# Create a window
cv.namedWindow('image')

cv.createTrackbar('CP', 'image', 0, 255, nothing) # Current Position

switch = 'color/gray'
cv.createTrackbar(switch, 'image', 0, 1, nothing)

while(1):
    img = cv.imread('lena.jpg')
    pos = cv.getTrackbarPos('CP', 'image')
    font = cv.FONT_HERSHEY_SIMPLEX
    cv.putText(img, str(pos), (50, 150), font, 6, (0, 0, 255), 10)

    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break

    s = cv.getTrackbarPos(switch, 'image')

    if s == 0:
       pass
    else:
       img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    img = cv.imshow('image',img)

cv.destroyAllWindows()
```



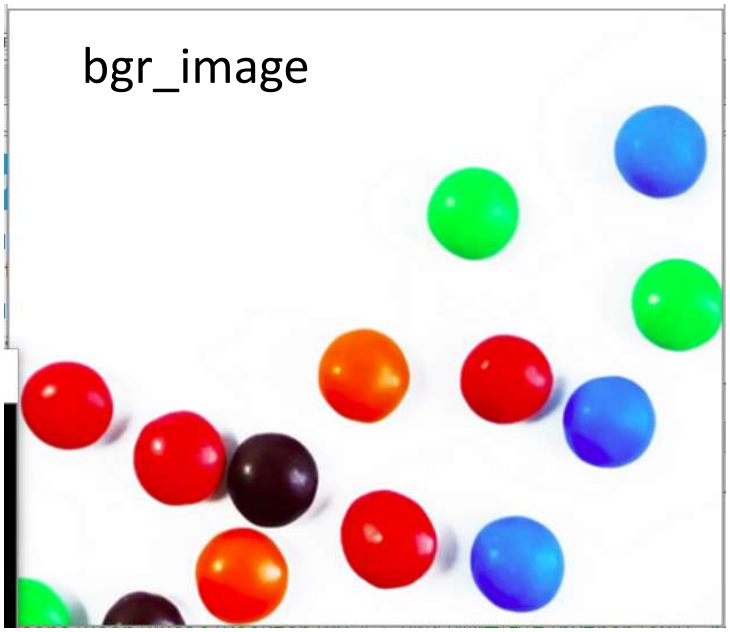
13. Object Detection and Object Tracking Using HSV Color Space

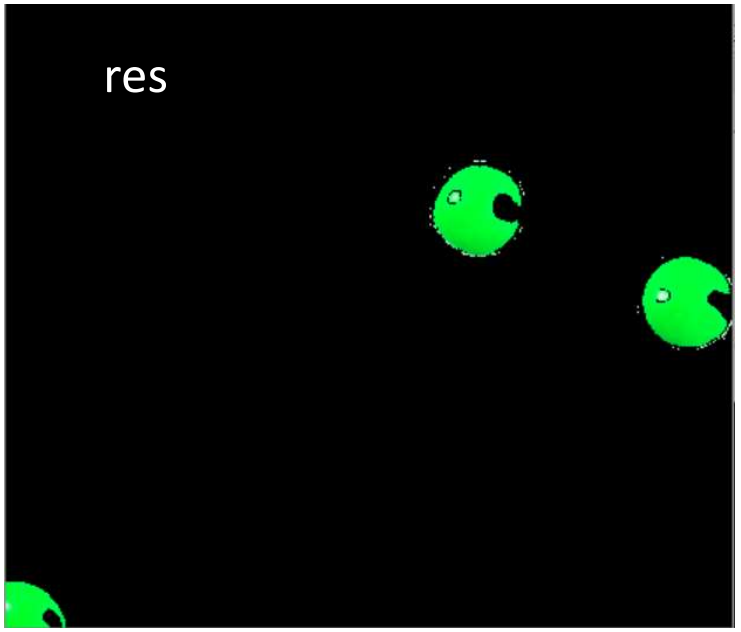
ex16.py ex17.py

```
hsv = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, lower_b, upper_b)
res = cv2.bitwise_and(bgr_image, bgr_image, mask=mask)
```

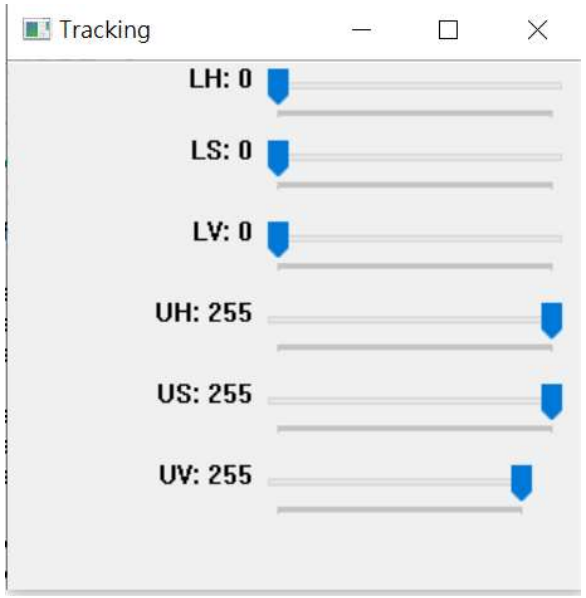
→ binary image

color thresholding





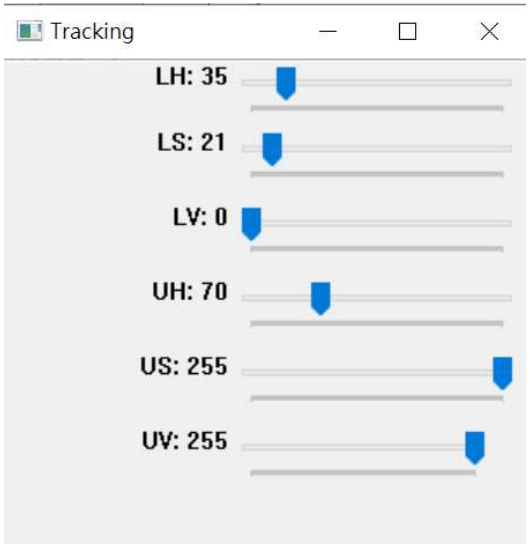
ex16.py



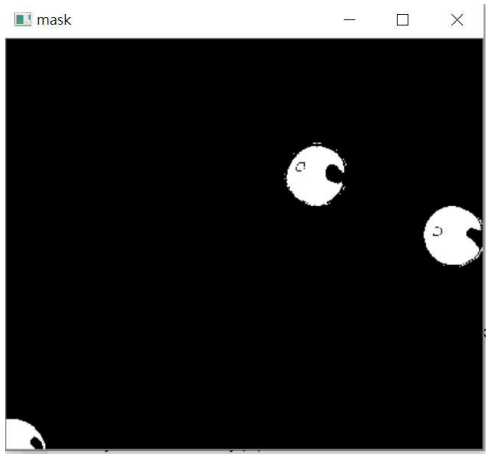

```
 8 import cv2
 9 import numpy as np
10
11 def nothing(x):
12     pass
13
14 cv2.namedWindow("Tracking")
15 cv2.createTrackbar("LH", "Tracking", 0, 255, nothing)
16 cv2.createTrackbar("LS", "Tracking", 0, 255, nothing)
17 cv2.createTrackbar("LV", "Tracking", 0, 255, nothing)
18 cv2.createTrackbar("UH", "Tracking", 255, 255, nothing)
19 cv2.createTrackbar("US", "Tracking", 255, 255, nothing)
20 cv2.createTrackbar("UV", "Tracking", 255, 255, nothing)
```

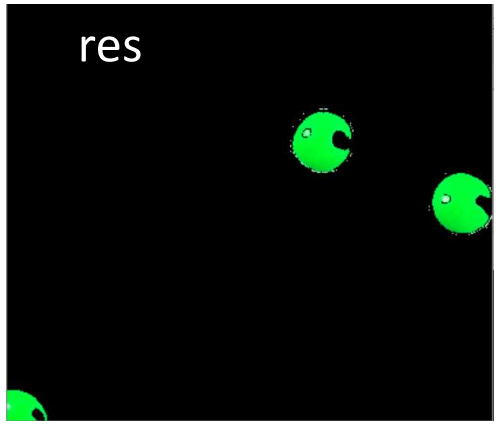
```python
while True:
    frame = cv2.imread('smarties.png')

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("LH", "Tracking")
    l_s = cv2.getTrackbarPos("LS", "Tracking")
    l_v = cv2.getTrackbarPos("LV", "Tracking")

    u_h = cv2.getTrackbarPos("UH", "Tracking")
    u_s = cv2.getTrackbarPos("US", "Tracking")
    u_v = cv2.getTrackbarPos("UV", "Tracking")

    l_b = np.array([l_h, l_s, l_v])
    u_b = np.array([u_h, u_s, u_v])

    mask = cv2.inRange(hsv, l_b, u_b)

    res = cv2.bitwise_and(frame, frame, mask=mask)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    cv2.imshow("res", res)

    key = cv2.waitKey(1)
    if key == 27:
        break

cv2.destroyAllWindows()
```

ex16.py



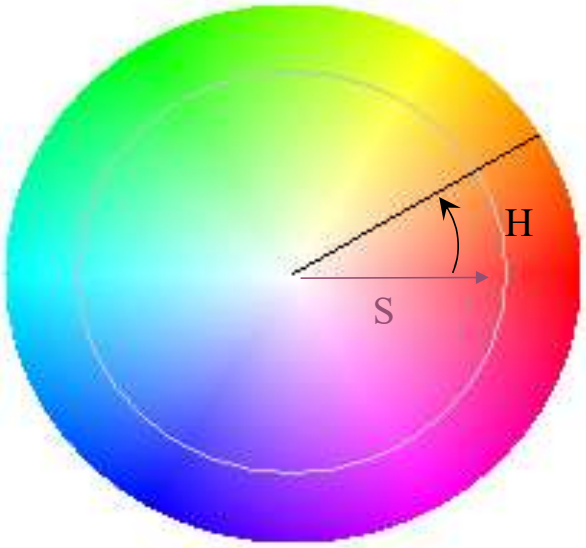
8-bit single channel array





!Note:

For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different softwares use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges.



!Note:
Thresholding Operations using inRange

dst=cv.inRange(src, lowerb, upperb[, dst])

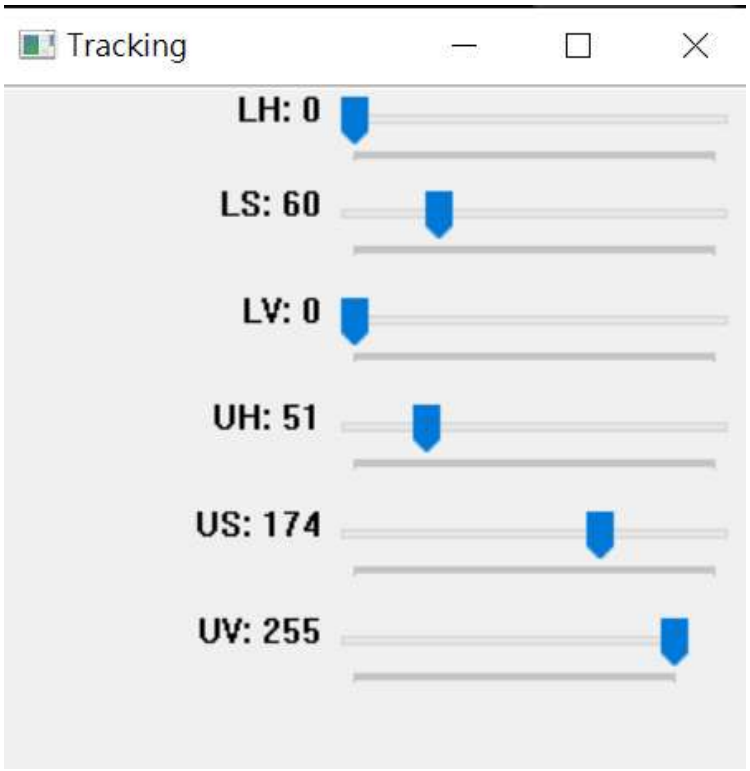
dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

ex17.py for object tracking

```python
import cv2
import numpy as np

def nothing(x):
    pass

cap = cv2.VideoCapture(0);

cv2.namedWindow("Tracking")
cv2.createTrackbar("LH", "Tracking", 0, 255, nothing)
cv2.createTrackbar("LS", "Tracking", 0, 255, nothing)
cv2.createTrackbar("LV", "Tracking", 0, 255, nothing)
cv2.createTrackbar("UH", "Tracking", 255, 255, nothing)
cv2.createTrackbar("US", "Tracking", 255, 255, nothing)
cv2.createTrackbar("UV", "Tracking", 255, 255, nothing)
```

```python
while True:
    #frame = cv2.imread('smarties.png')
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("LH", "Tracking")
    l_s = cv2.getTrackbarPos("LS", "Tracking")
    l_v = cv2.getTrackbarPos("LV", "Tracking")

    u_h = cv2.getTrackbarPos("UH", "Tracking")
    u_s = cv2.getTrackbarPos("US", "Tracking")
    u_v = cv2.getTrackbarPos("UV", "Tracking")

    l_b = np.array([l_h, l_s, l_v])
    u_b = np.array([u_h, u_s, u_v])

    mask = cv2.inRange(hsv, l_b, u_b)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    cv2.imshow("res", res)

    key = cv2.waitKey(1)
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

The skin in channel H is characterized by values between 0 and 50, in the channel S from 0.23 (59) to 0.68 (174) for Asian and Caucasian ethnics.



?

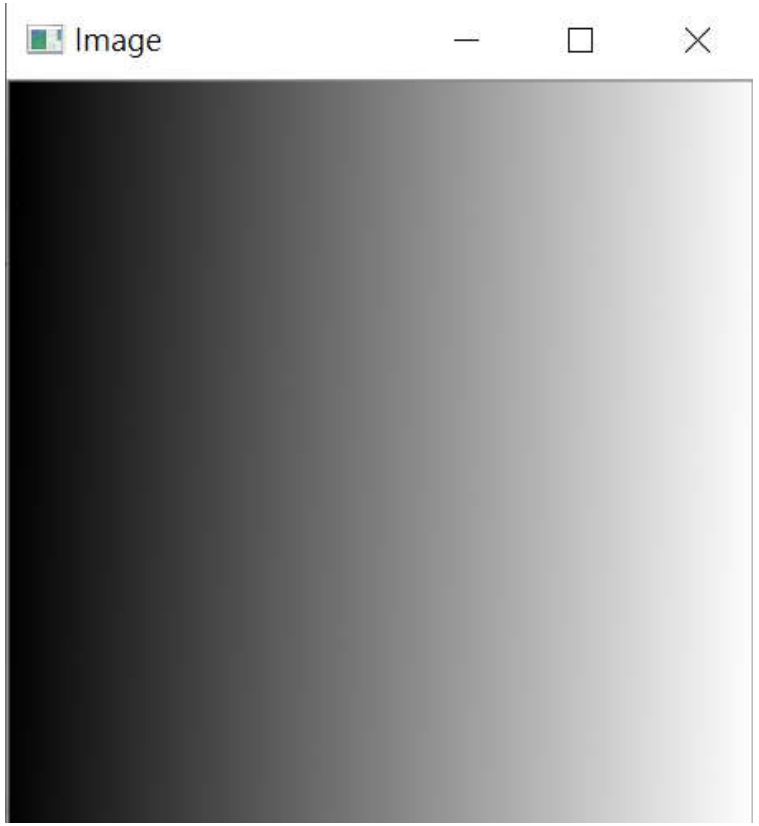
14. Simple Image Thresholding

ret, out = cv2.threshold(src image, thresh, maxVal, method)

First argument is the source image, which should be a grayscale image.
Second argument is the threshold value which is used to classify the pixel values.
Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.
Fourth parameter provides 5 different styles of thresholding:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

gradient.png
300x300x3



- THRESH_BINARY

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- THRESH_BINARY_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- THRESH_TRUNC

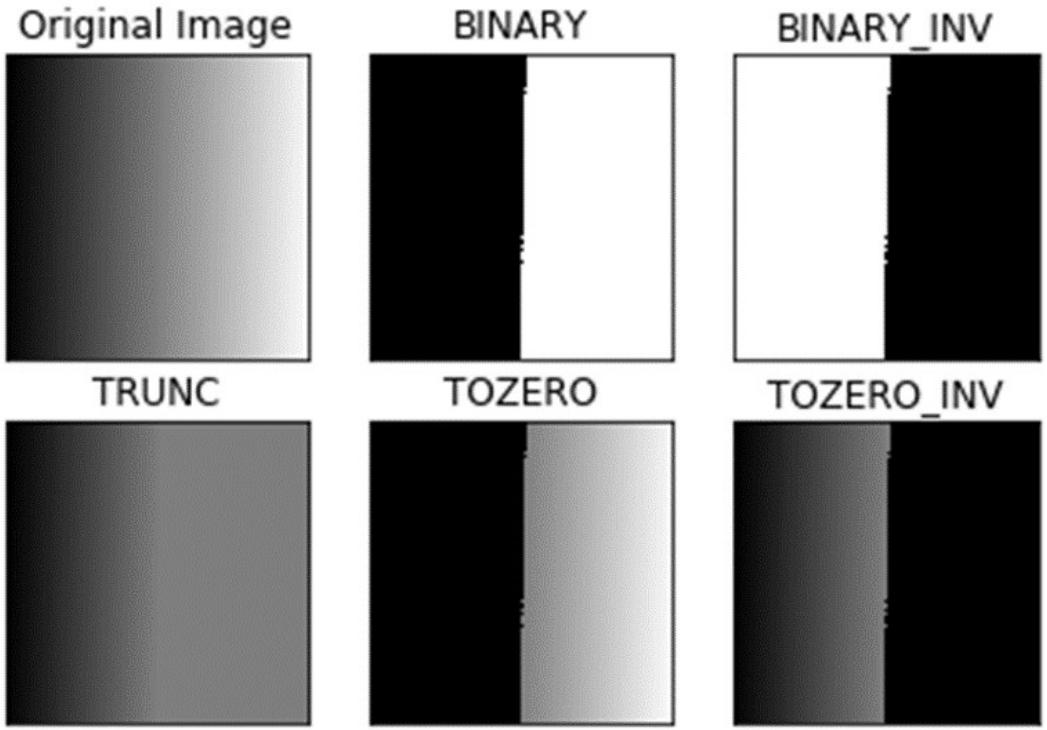
$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- THRESH_TOZERO

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

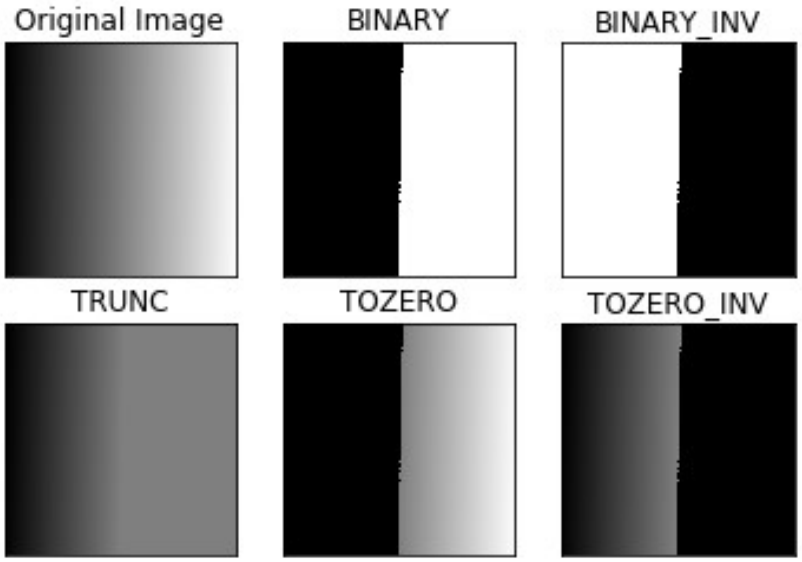
- THRESH_TOZERO_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$



ex18.py

```python
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('gradient.png',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



15. Adaptive Thresholding

In 14. simple image thresholding, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculate the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

sudoku.png
558x563x3

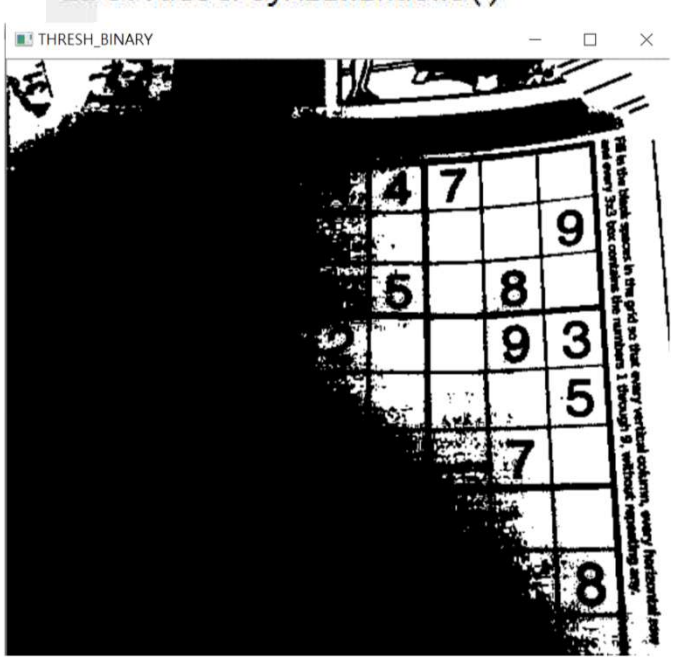
Adaptive Method - It decides how thresholding value is calculated.
cv2.ADAPTIVE_THRESH_MEAN_C : threshold value is the mean of neighbourhood area.
cv2.ADAPTIVE_THRESH_GAUSSIAN_C : threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.
Block Size - It decides the size of neighbourhood area.
C - It is just a constant which is subtracted from the mean or weighted mean calculated.

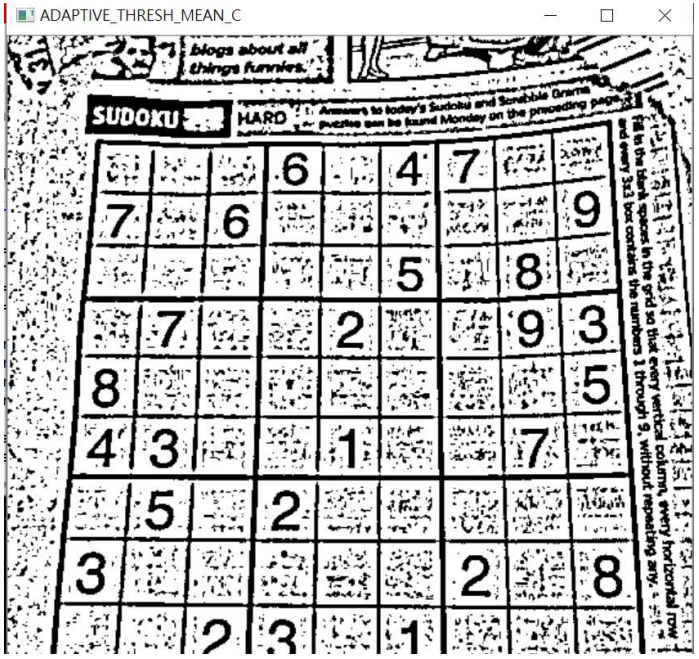
```
cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 2);
cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 2)
```

src, maxVal

ex19.py

```python
import cv2 as cv

img = cv.imread('sudoku.png',0) # grayscale
_, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
th2 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 2)
th3 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 2)

cv.imshow("Image", img)
cv.imshow("THRESH_BINARY", th1)
cv.imshow("ADAPTIVE_THRESH_MEAN_C", th2)
cv.imshow("ADAPTIVE_THRESH_GAUSSIAN_C", th3)

cv.waitKey(0)
cv.destroyAllWindows()
```





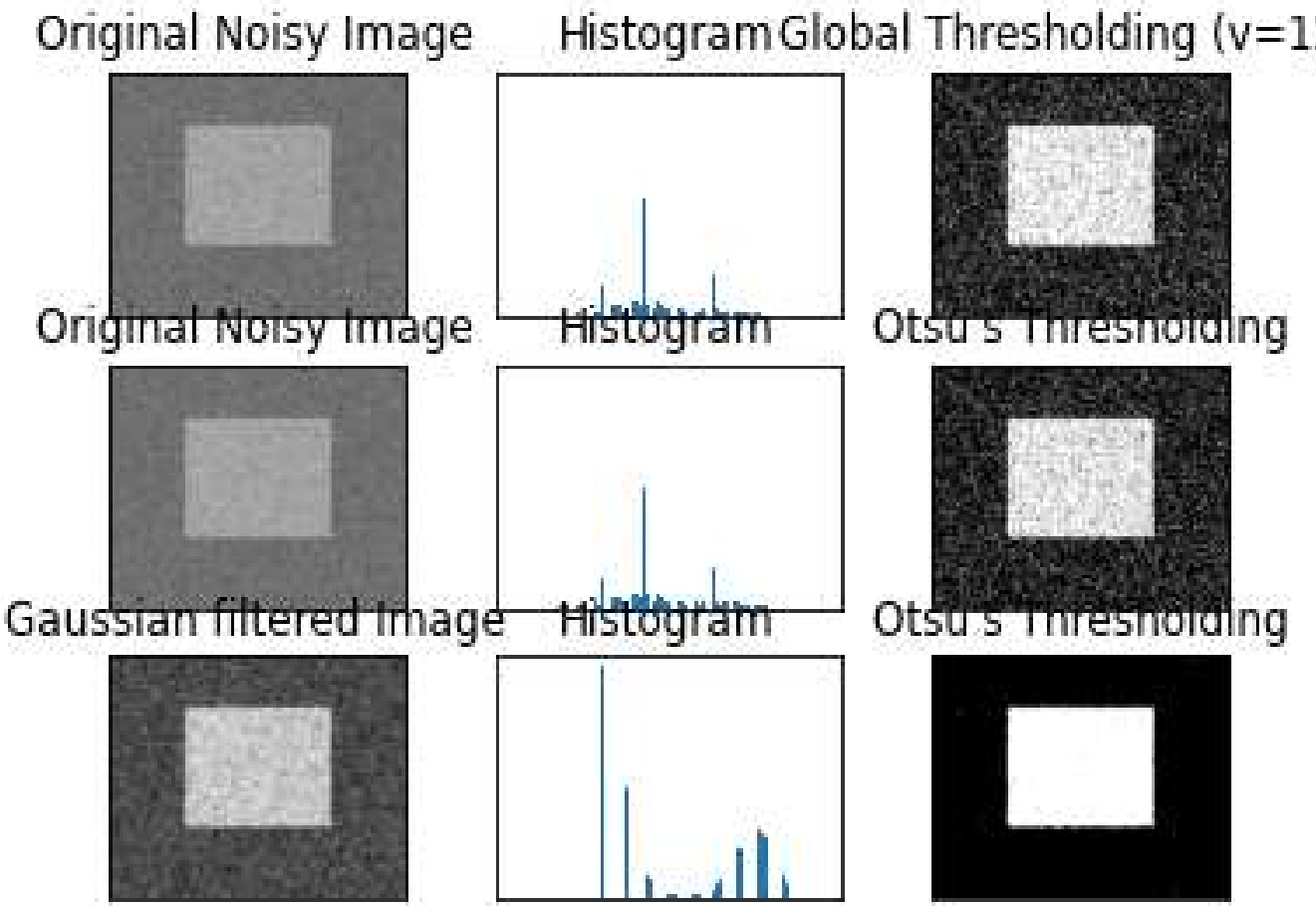
Otsu's Binarization

In global thresholding, we used an arbitrary chosen value as a threshold. In contrast, Otsu's method avoids having to choose a value and determines it automatically.

Consider an image with only two distinct image values (bimodal image), where the histogram would only consist of two peaks. A good threshold would be in the middle of those two values. Similarly, Otsu's method determines an optimal global threshold value from the image histogram.

In order to do so, the cv.threshold() function is used, where cv.THRESH_OTSU is passed as an extra flag. The threshold value can be chosen arbitrary (typically 0). The algorithm then finds the optimal threshold value which is returned as the first output.

Check out the example below. The input image is a noisy image. In the first case, global thresholding with a value of 127 is applied. In the second case, Otsu's thresholding is applied directly. In the third case, the image is first filtered with a 5x5 gaussian kernel to remove the noise, then Otsu thresholding is applied. See how noise filtering improves the result.



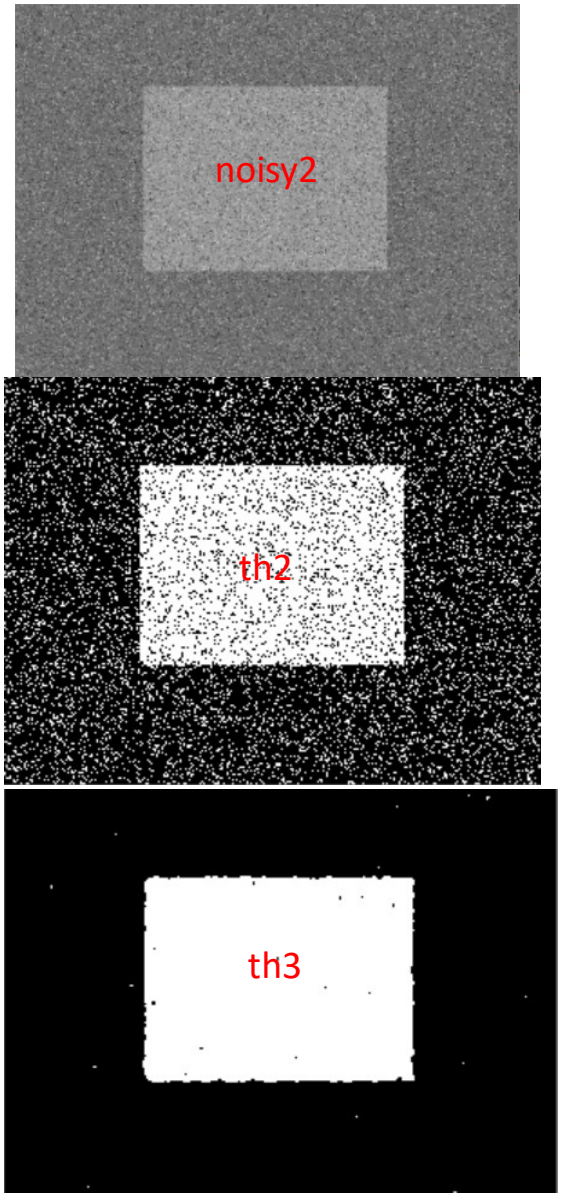
ret1: 127.0

ret2: 130.0

ret3: 132.0

otsu.py

```
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('noisy2.png',0)  # grayscale
# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)   # see 18 blurring image
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1, img, 0, th2, blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
        'Original Noisy Image','Histogram',"Otsu's Thresholding",
        'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```



16. matplotlib with OpenCV

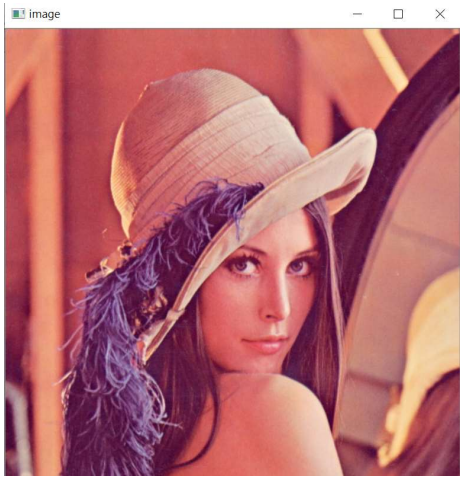
See ex18.py

ex20.py

```
import cv2
Import matplotlib.pyplot as plt

img1 = cv2.imread('lena.jpg', -1)
cv2.imshow('image', img1)
img2 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

plt.imshow(img2)
plt.xticks([]), plt.yticks([])
plt.show()

cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2: BGR

matplotlib: RGB



plt.imshow(img1)

17. Morphological Transformations

morphological operations like Erosion, Dilation, Opening, Closing etc. We will see different functions like : cv.erode(), cv.dilate(), cv.morphologyEx()

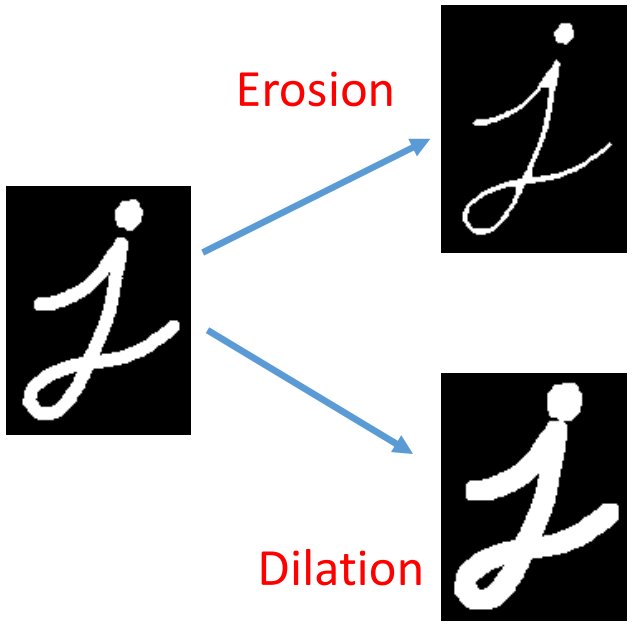
Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation.

kernel

1	**1**	**1**
1	1	1
1	1	1

Erosion: A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

Dilation: a pixel element is '1' if at least one pixel under the kernel is '1'.



useful for removing small white noises, detach two connected objects etc.

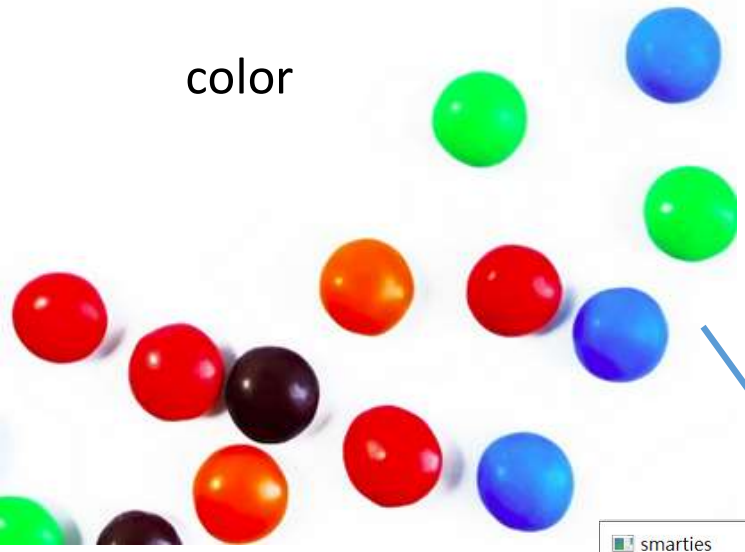
Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

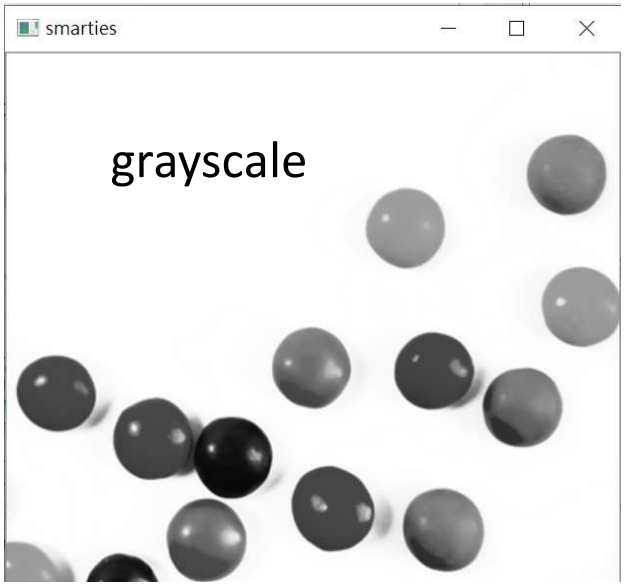
kernel: In image processing, a kernel, convolution matrix, or a mask is a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more.

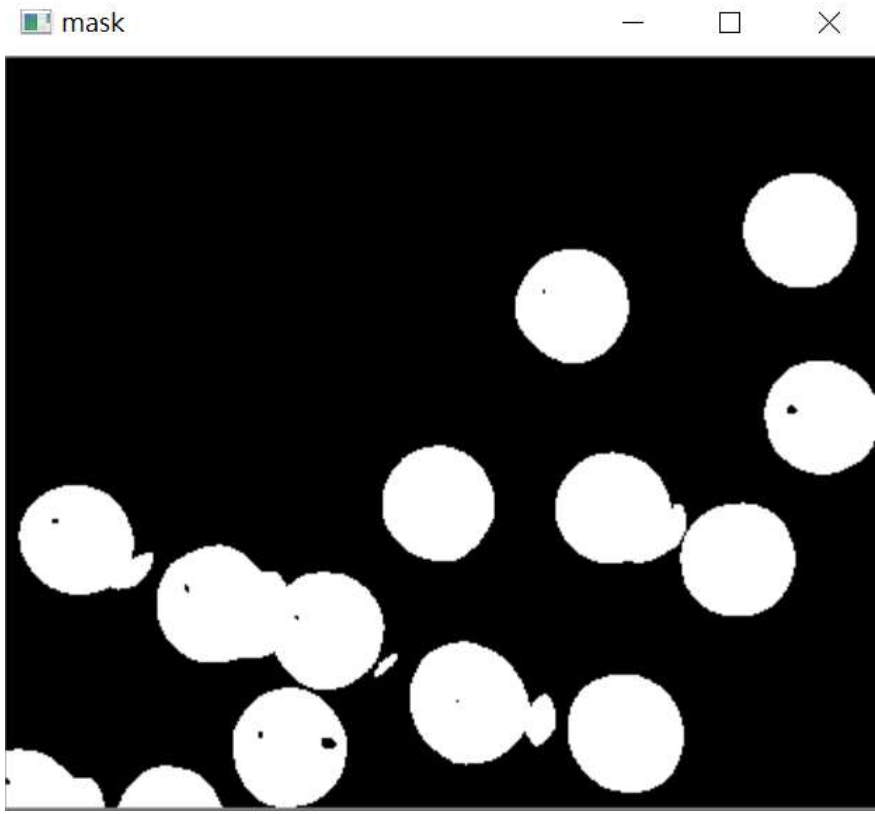
Opening: erosion followed by dilation, useful in removing noise.

Closing: dilation followed by erosion, useful in closing small holes inside the foreground objects, or small black points on the object.

color





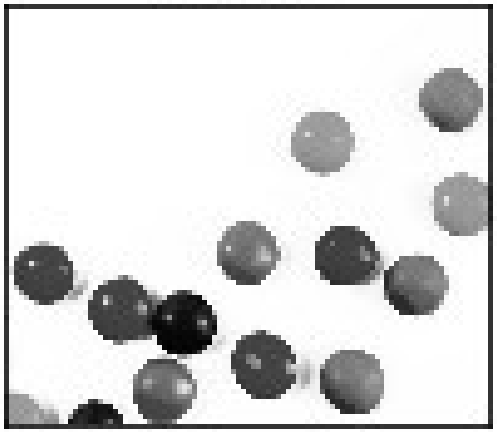


Simple thresholding

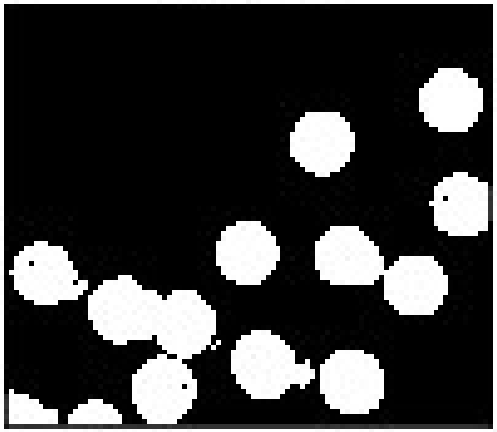
ex21.py

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

img = cv2.imread('smarties.png', cv2.IMREAD_GRAYSCALE)  # 0
_, mask = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY_INV)

kernel = np.ones((5,5), np.uint8) # 5x5
dilation = cv2.dilate(mask, kernel, iterations=2)
erosion = cv2.erode(mask, kernel, iterations=1)
opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
titles = ['image', 'mask', 'dilation', 'erosion', 'opening', 'closing']
images = [img, mask, dilation, erosion, opening, closing]

for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

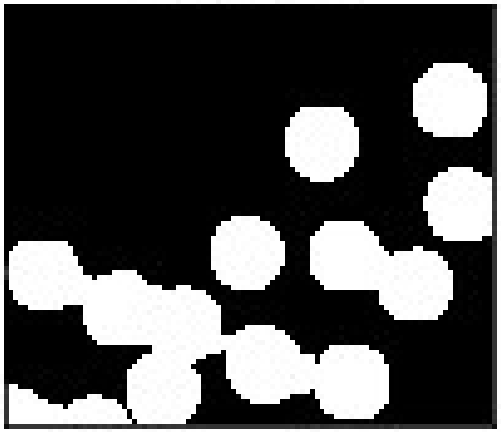
image



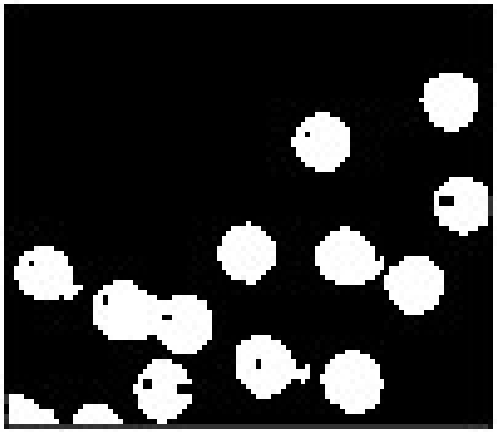
mask



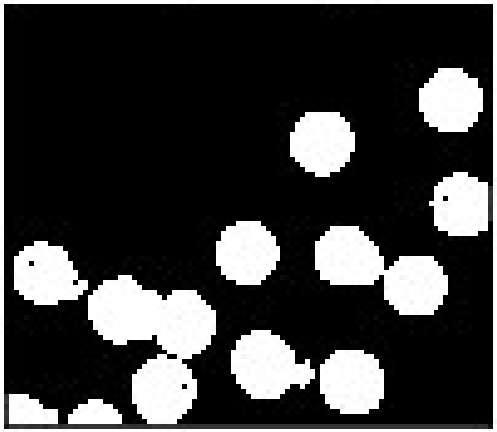
dilation



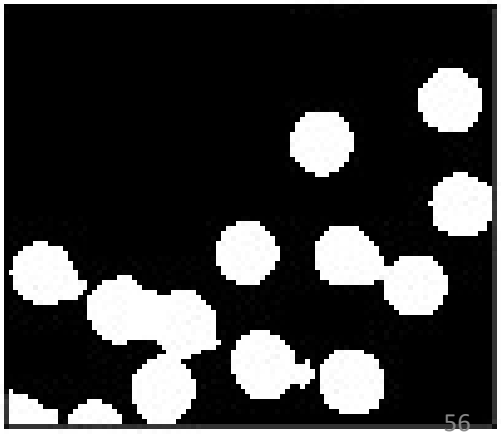
erosion



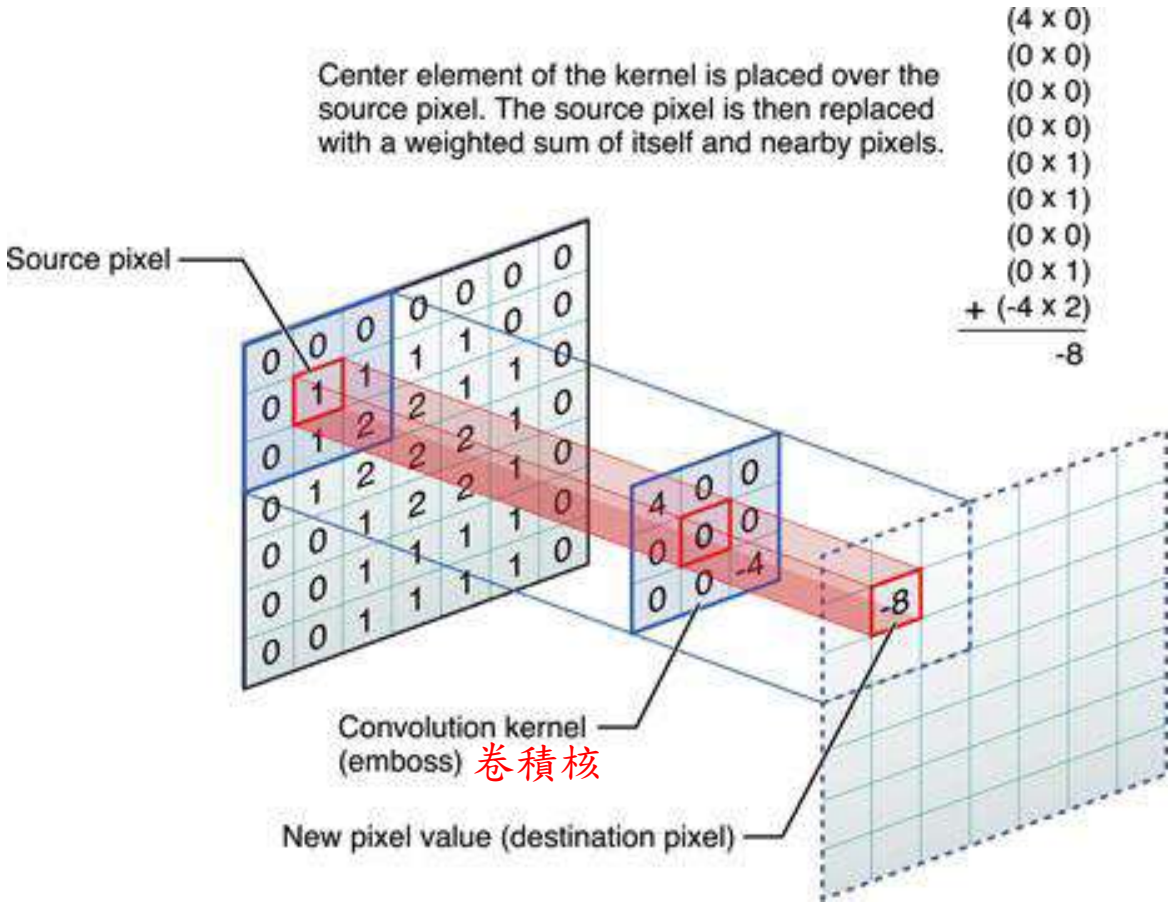
opening



dosing



18. Smoothing Images | Blurring Images



18. Smoothing Images | Blurring Images

We will learn different morphological operations like 2D Convolution (Image Filtering) and Image Blurring (Image Smoothing) using Averaging, Gaussian Blurring, Median Blurring, Bilateral Filtering etc. We will see different functions like : cv.filter2D(), cv.blur(), cv.GaussianBlur(), cv.medianBlur(), cv.bilateralFilter() etc.

dst = cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])

desired depth of the destination image 目標圖像的所需深度

ddepth = -1, the output image will have the same depth as the source.

dst = cv2.blur(src, ksize): averaging

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

ksize=(3,3)

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

ksize=(5,5)

dst = cv2.GaussianBlur(src，ksize，sigmaX [，DST [，sigmaY [，borderType]]] ）

σ（空間的標準差）:0 程式自動計算

If only sigmaX is specified, sigmaY is taken as equal to sigmaX.

If both are given as zeros, they are calculated from the kernel size.

Gaussian filtering is highly effective in removing Gaussian noise from the image.

Gaussian Filter的模糊化效果比Averaging明顯，更為自然。

dst = cv2.GaussianBlur(img, (5, 5), 0)

59

18. Smoothing Images | Blurring Images

dst = cv2.medianBlur(src, d): ksize=(d, d)

good for salt-and-pepper noise

cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value.

The central element is always replaced by some pixel value in the image. This reduces the noise effectively.

dst = cv2.medianBlur(img, 5)

dst = cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace):（顏色空間的標準差）

能保持邊界清晰且有效的消除噪聲

但比較慢。

highly effective at noise removal while preserving edges.

color σ即顏色空間的標準差，愈大代表在計算時需要考慮更多的顏色。space σ即坐標空間的標準差，這個參數與Guassian filter使用的相同，數值越大，代表越遠的像素有較大的權值。

簡單起見，可以令2個sigma的值近似

如果他們很小（小於10），那麼濾波器幾乎沒有什麼效果

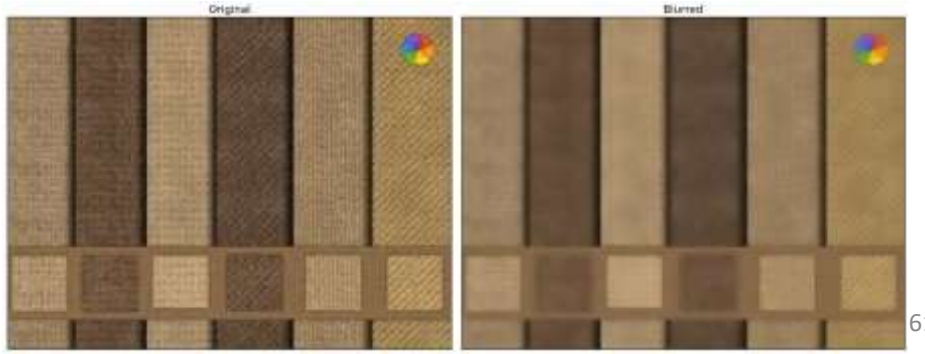
如果他們很大（大於150），那麼濾波器的效果會很強，使圖像清晰非常卡通化

cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace):

The bilateral filter also uses a Gaussian filter in the space domain, but it also uses one more (multiplicative) Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are 'spatial neighbors' are considered for filtering, while the Gaussian component applied in the intensity domain (a Gaussian function of intensity differences) ensures that only those pixels with intensities similar to that of the central pixel ('intensity neighbors') are included to compute the blurred intensity value.
As a result, this method preserves edges, since for pixels lying near edges, neighboring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring.

cv2.bilateralFilter(img,9,75,75)

The texture on the surface is gone, but edges are still preserved



ex22.py

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

#img = cv2.imread('lena.jpg')
#img = cv2.imread('opencv-logo.png')
#img = cv2.imread('water.png')
img = cv2.imread('Halftone_Gaussian_Blur.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

kernel = np.ones((5, 5), np.float32)/25
dst = cv2.filter2D(img, -1, kernel)
blur = cv2.blur(img, (5, 5))
gblur = cv2.GaussianBlur(img, (5, 5), 0)
median = cv2.medianBlur(img, 5)
bilateralFilter = cv2.bilateralFilter(img, 9, 75, 75)

titles = ['image', '2D Convolution', 'blur', 'GaussianBlur', 'median', 'bilateralFilter']
images = [img, dst, blur, gblur, median, bilateralFilter]

for i in range(6):
    plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
```

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

dst = blur

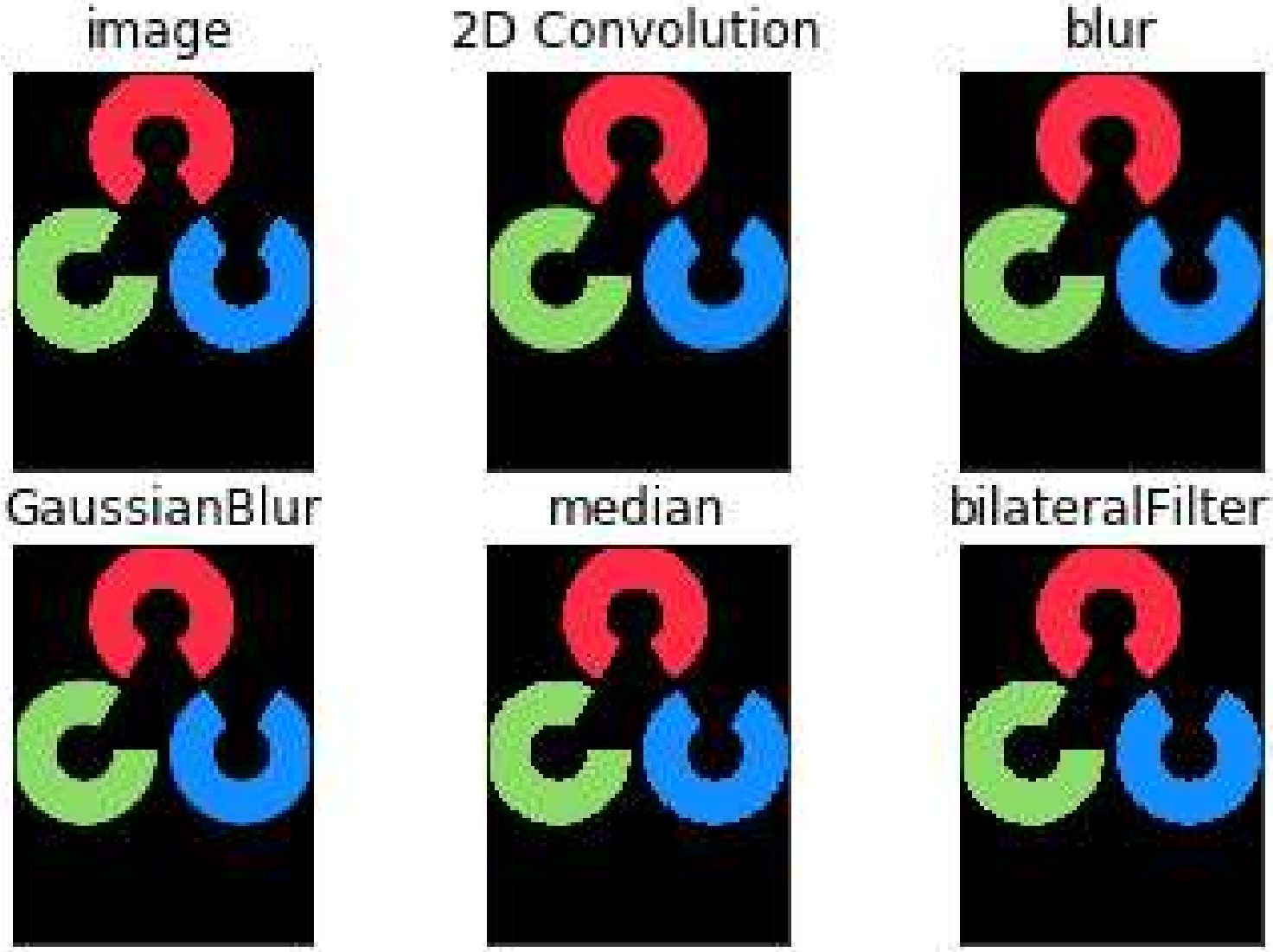


image
2D Convolution
blur
GaussianBlur
median
bilateralFilter


image
2D Convolution
blur
GaussianBlur
median
bilateralFilter

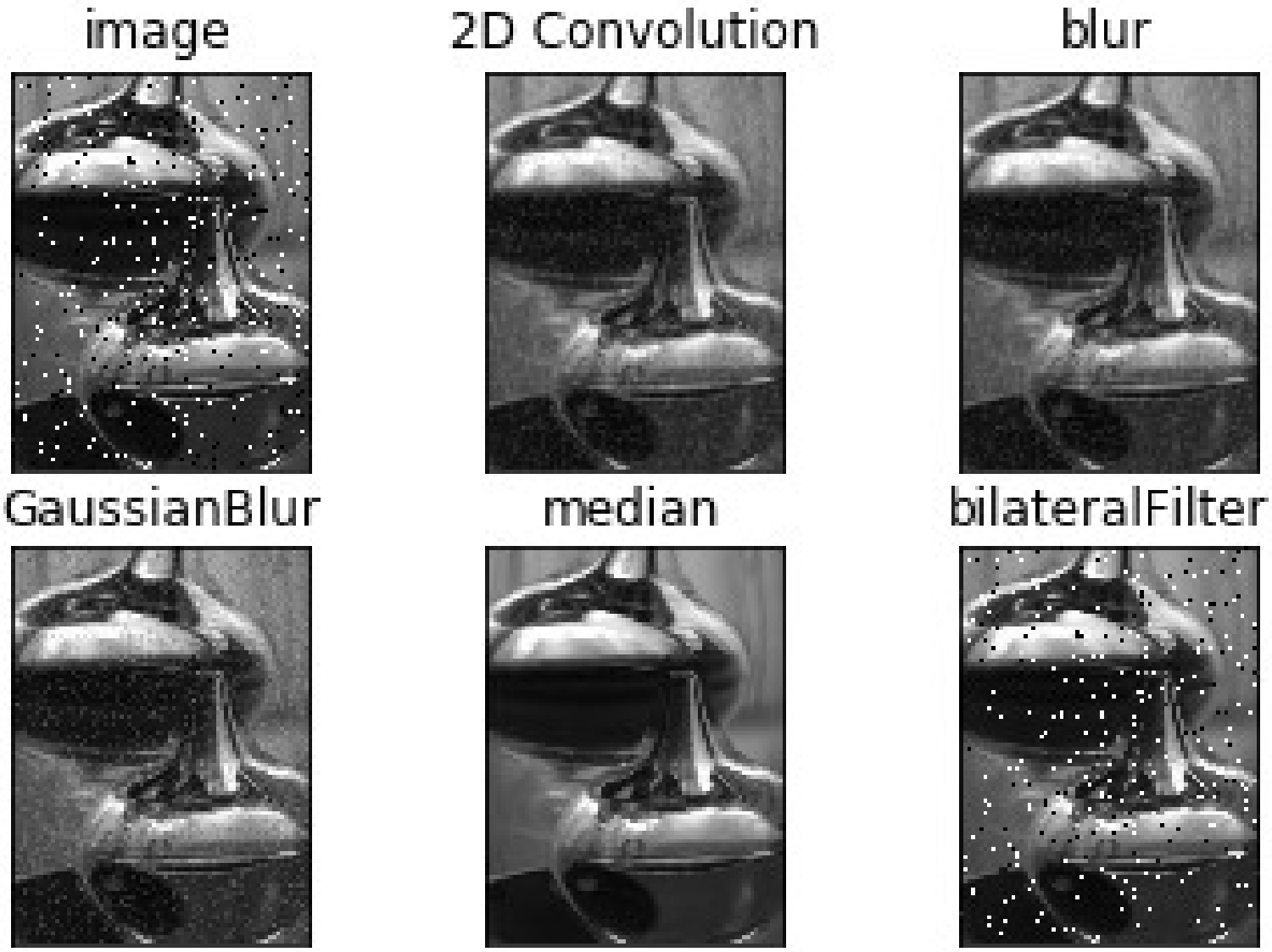
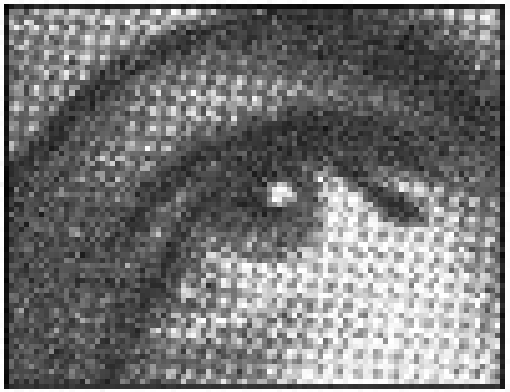


image
2D Convolution
blur
GaussianBlur
median
bilateralFilter

image



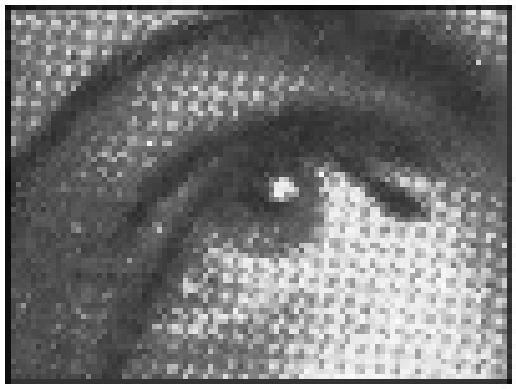
2D Convolution

blur

GaussianBlur

median

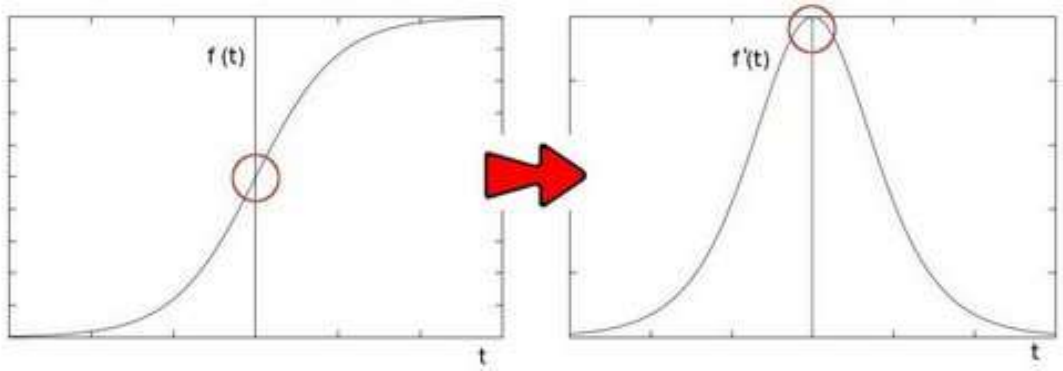
bilateralFilter



19. Image Gradients and Edge Detection

OpenCV provides three types of gradient filters or High-pass filters:
Sobel, Scharr and Laplacian (input image in grayscale).

Sobel: detect changes in the first derivative of intensity.
Getting the first derivative of the intensity, we observed that an edge is characterized by a maximum.



dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

前四個是必須的引數：

第一個引數是需要處理的灰階影象；
第二個引數是影象的深度，-1表示採用的是與原影象相同的深度。目標影象的深度必須大於等於原影象的深度，可使用cv2.CV_16S, cv2.CV_64F；
第三、四個引數dx和dy表示求導的階數，0表示這個方向上沒有求導，一般為0、1、2。
其後是可選的引數：

dst: Destination (output) image；
ksize是Sobel運算元的大小，必須為1、3、5、7。
scale是縮放導數的比例常數，預設情況下沒有伸縮係數；1
delta是一個可選的增量，將會加到最終的dst中，同樣，預設情況下沒有額外的值加到dst中；0
borderType是判斷影象邊界的模式。這個引數預設值為v2.BORDER_DEFAULT。

Sobel Operator ksize = 3

Assuming that the image to be operated is I:

1. We calculate two derivatives:

   a. **Horizontal changes**: This is computed by convolving I with a kernel G_x with odd size. For example for a kernel size of 3, G_x would be computed as:

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

   a. **Vertical changes**: This is computed by convolving I with a kernel G_y with odd size. For example for a kernel size of 3, G_y would be computed as:

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

Note:
When the size of the kernel is 3, the Sobel kernel shown above may produce noticeable inaccuracies (after all, Sobel is only an approximation of the derivative). OpenCV addresses this inaccuracy for kernels of size 3 by using the Scharr() function. This is as fast but more accurate than the standard Sobel function. It implements the following kernels:

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

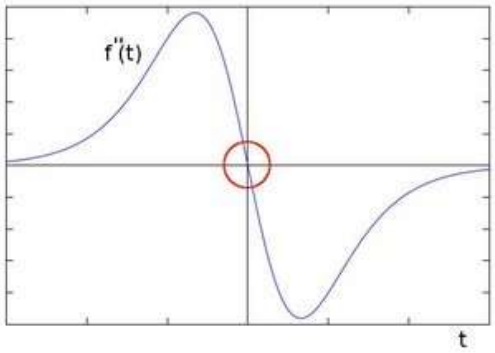
$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

dst=cv2.Scharr(src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]]])

dst = cv2.Scharr(src, ddepth, 1, 0)

Laplacian: detect zero crossings of the second derivative on intensity changes

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$



You can observe that the second derivative is zero! So, we can also use this criterion to attempt to detect edges in an image. However, note that zeros will not only appear in edges (they can actually appear in other meaningless locations); this can be solved by applying filtering where needed.

dst = cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])

dst = cv2.Laplacian(src_gray, ddepth, ksize=kernel_size)

- The arguments are:
- src_gray: The input image.
- dst: Destination (output) image
- ddepth: Depth of the destination image. Since our input is CV_8U we define ddepth = CV_64F to avoid overflow
- kernel_size: The kernel size of the Sobel operator to be applied internally. We use 3 in this example.
- scale, delta and BORDER_DEFAULT: We leave them as default values.

ex23.py

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

img = cv2.imread("messi5.jpg", cv2.IMREAD_GRAYSCALE)
#img = cv2.imread("sudoku.png", cv2.IMREAD_GRAYSCALE)
lap = cv2.Laplacian(img, cv2.CV_64F, ksize=3)
lap = np.uint8(np.absolute(lap))
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1)

sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))

sobelCombined = cv2.bitwise_or(sobelX, sobelY)
```

sobelCombined = cv2.addWeighted(sobelX, 0.5, sobelY, 0.5, 0)

bitwise_or: 類似 union 聯集
ex:
33 bitwise_or 21 = 53

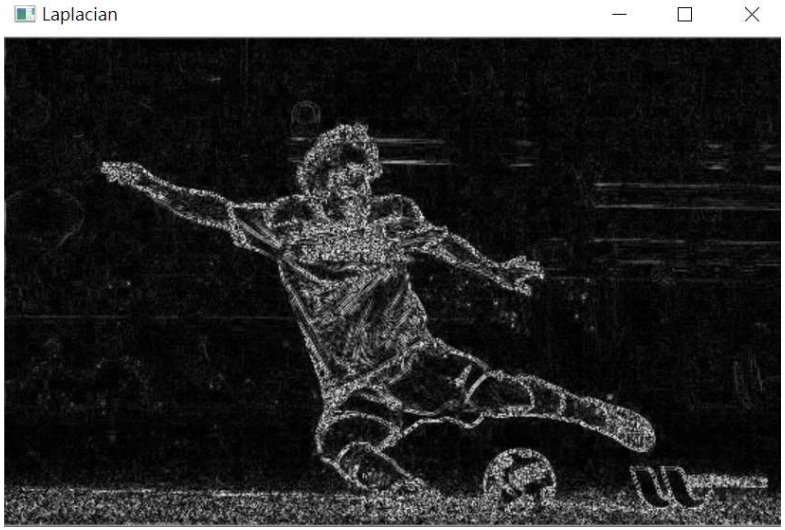
00100001 (33)
00010101 (21)

00110101 (53)

40 bitwise_or 44 = 44

```python

titles = ['image', 'Laplacian', 'sobelX', 'sobelY', 'sobelCombined']
images = [img, lap, sobelX, sobelY, sobelCombined]
for i in range(5):
    plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()

cv2.imshow('image',img)
cv2.imshow('Laplacian',lap)
cv2.imshow('sobelX',sobelX)
cv2.imshow('sobelY',sobelY)
cv2.imshow('sobelCombined',sobelCombined)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

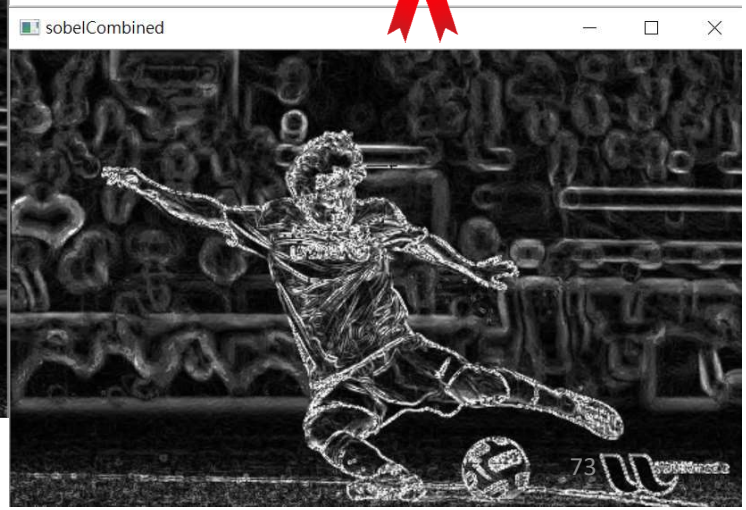

image
unicef
WEEKmedia


Laplacian

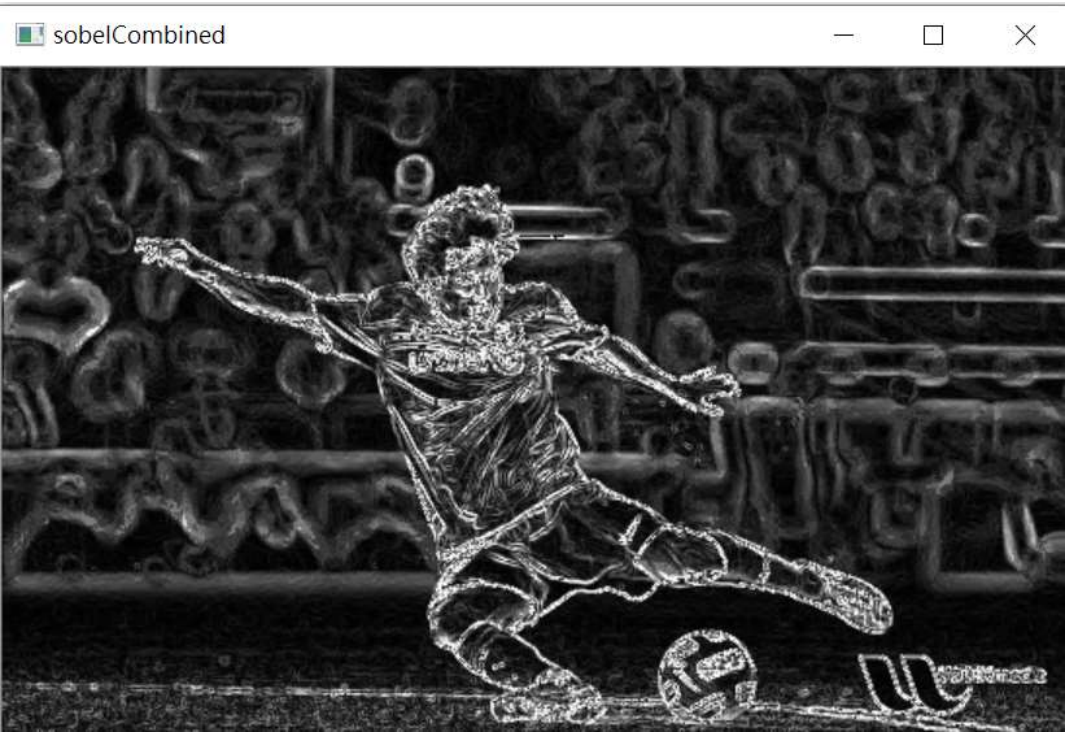
edge: 亮


sobelX


sobelY


sobelCombined
73

cv2.bitwise_or(sobelX, sobelY)



cv2.addWeighted(sobelX, 0.5, sobelY, 0.5, 0)

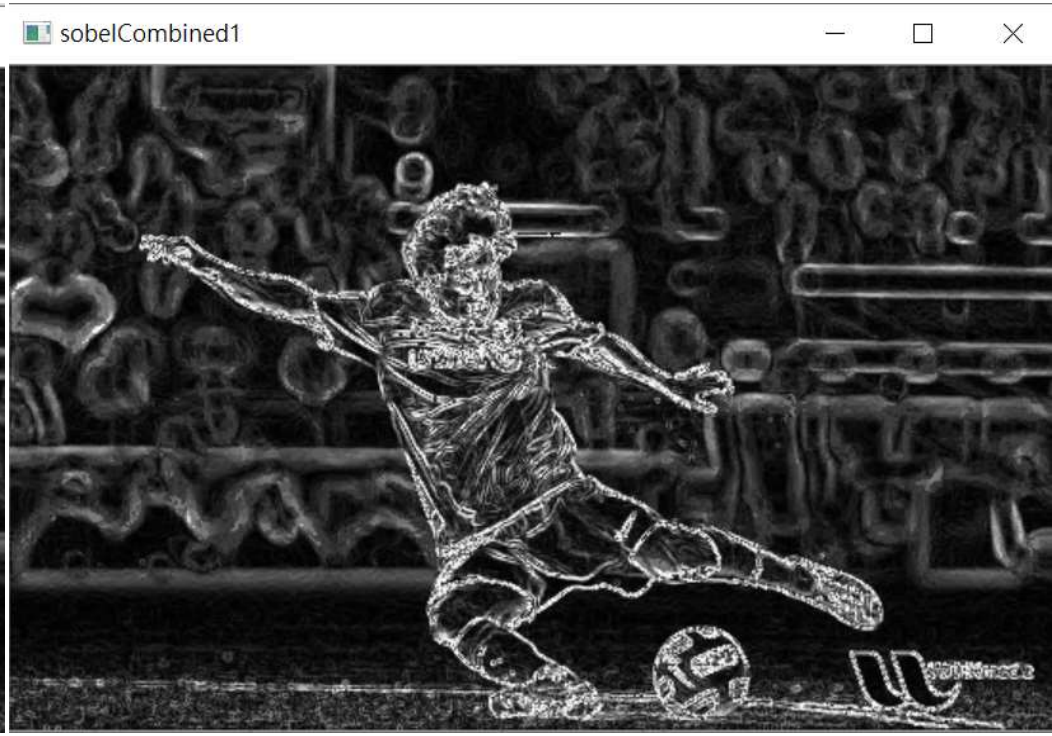
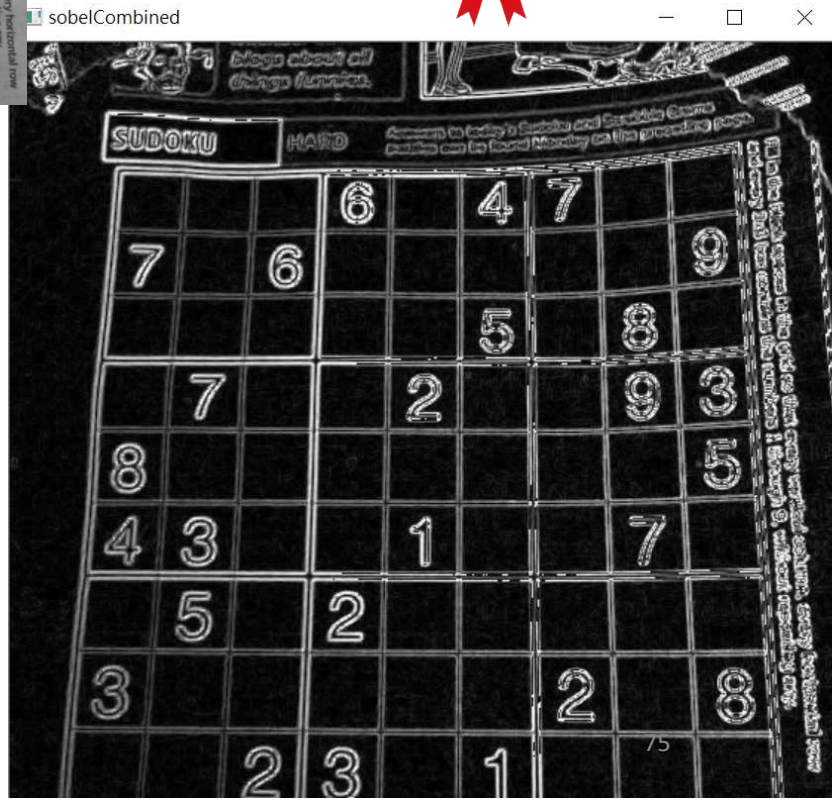



image
blogs about all things funnies.
SUDOKU
HARD
Answers to today's Sudoku and Scrabble Grams puzzles can be found Monday on the preceding page.
Fill in the blank spaces in the grid so that every vertical column, every horizontal row and every 3x3 box contains the numbers 1 through 9, without repeating any.

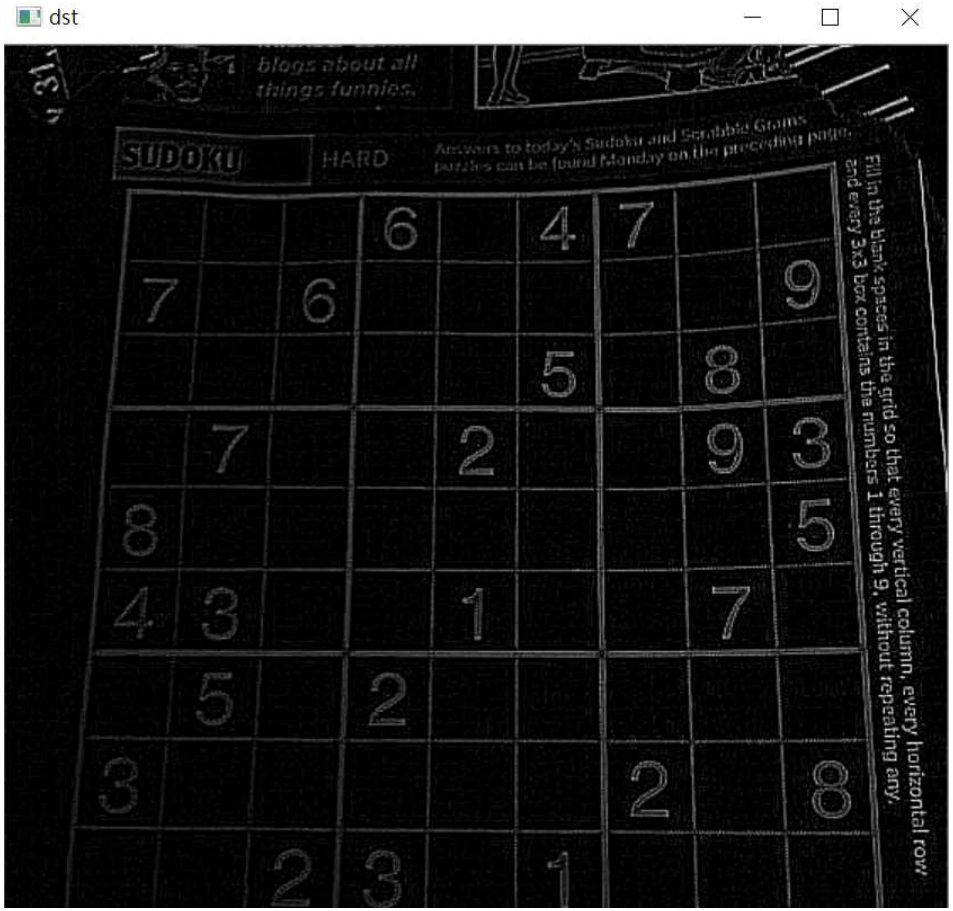

Laplacian
SUDOKU
HARD


sobelCombined
blogs about all things funnies.
SUDOKU
HARD

More on Laplacian: two commonly used Laplacian operators

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
import cv2
import numpy as np
#img = cv2.imread("messi5.jpg", cv2.IMREAD_GRAYSCALE)
img = cv2.imread("sudoku.png", cv2.IMREAD_GRAYSCALE)
kernel = np.ones((3,3), np.float32)
kernel[1][1]=-8
dst = cv2.filter2D(img,-1, kernel)
cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



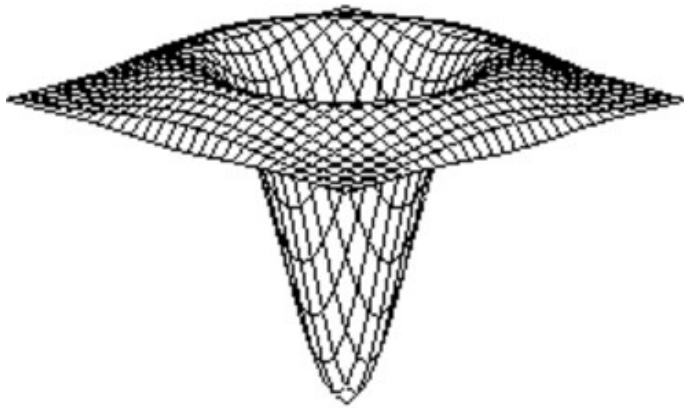
LoG (Laplacian of Gaussian) :

Because these Laplacian kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

LoG: 9x9 kernel

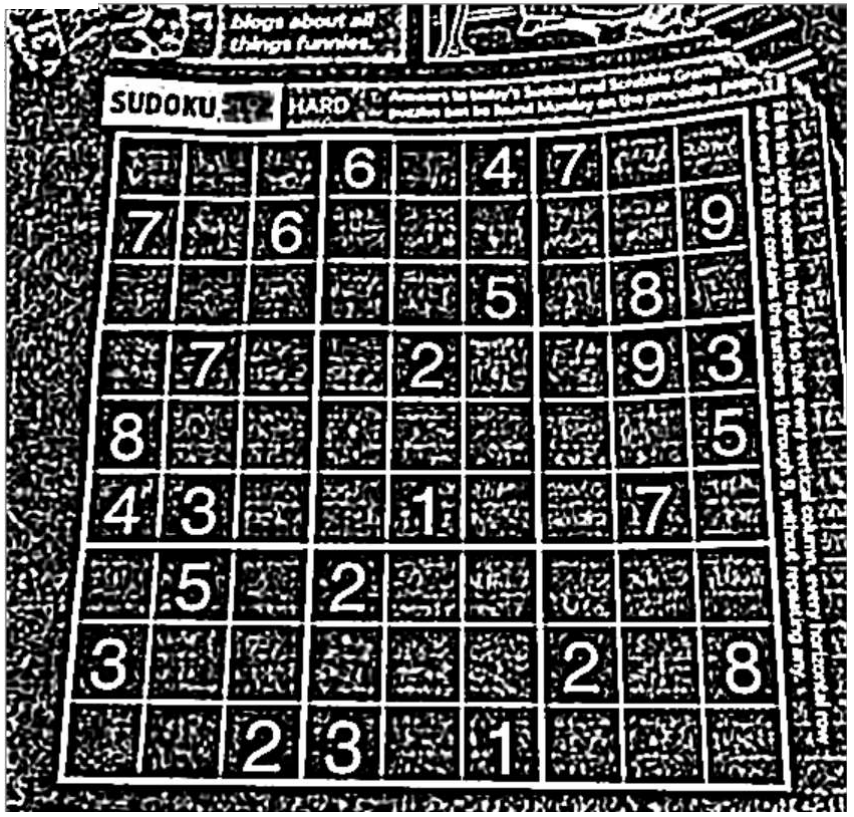
Gaussian (sigma = 1.4)

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

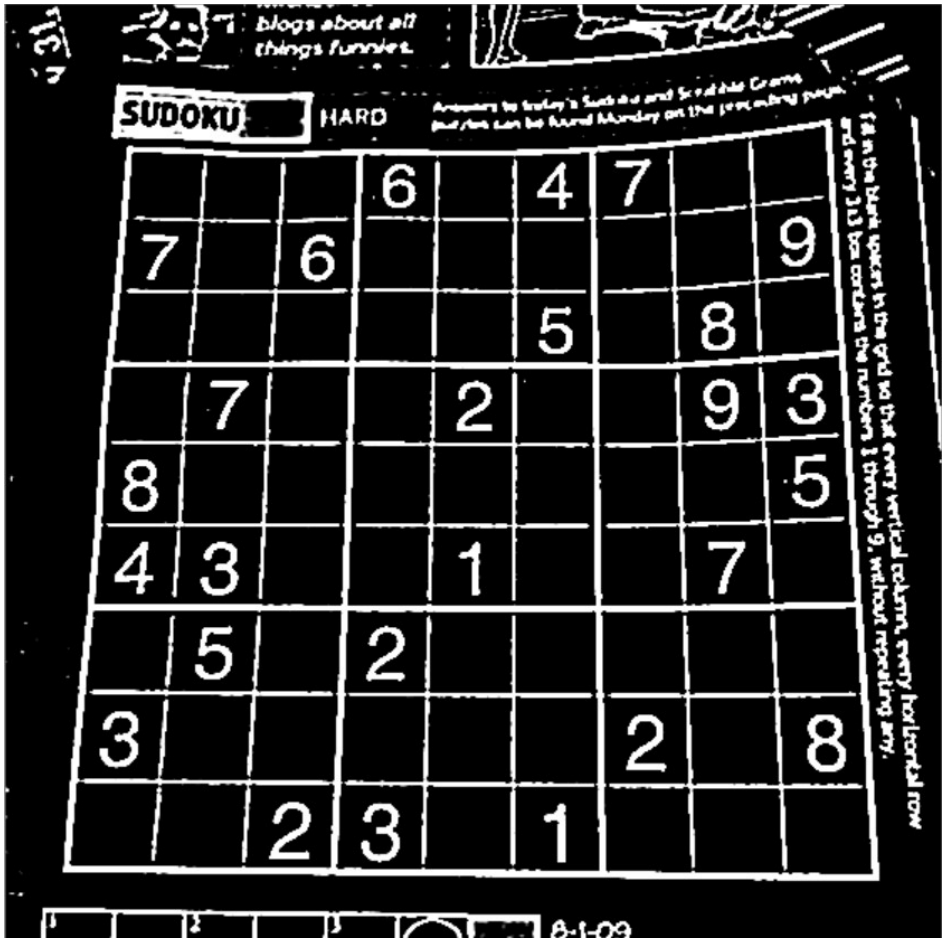


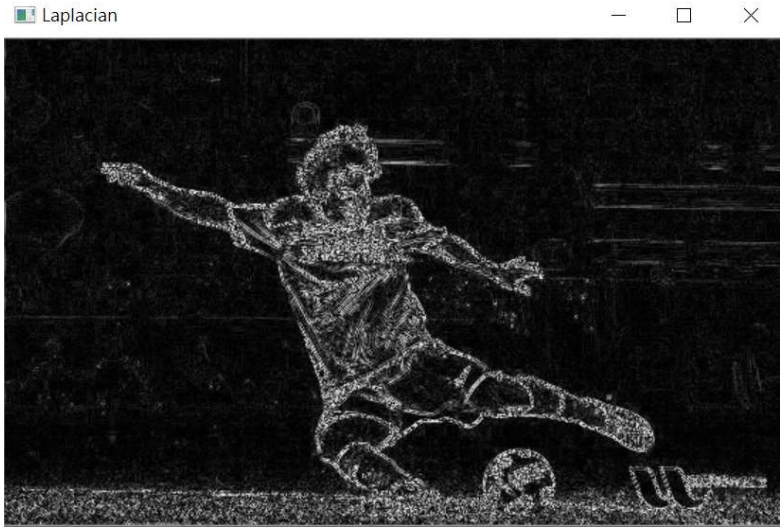
```
import cv2
import numpy as np

#img = cv2.imread("messi5.jpg", cv2.IMREAD_GRAYSCALE)
img = cv2.imread("sudoku.png", cv2.IMREAD_GRAYSCALE)
kernel = np.array([[0, 1, 1, 2, 2, 2, 1, 1, 0],[1, 2, 4, 5, 5, 5, 4, 2, 1],
                   [1, 4, 5, 3, 0, 3, 5, 4, 1],[2, 5, 3, -12, -24, -12, 3, 5, 2],
                   [2, 5, 0, -24, -40, -24, 0, 5, 2],[2, 5, 3, -12, -24, -12, 3, 5, 2],
                   [1, 4, 5, 3, 0, 3, 5, 4, 1],[1, 2, 4, 5, 5, 5, 4, 2, 1],
                   [0, 1, 1, 2, 2, 2, 1, 1, 0]], dtype = np.float32)
dst = cv2.filter2D(img,-1, kernel)

cv2.imshow('LoG',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

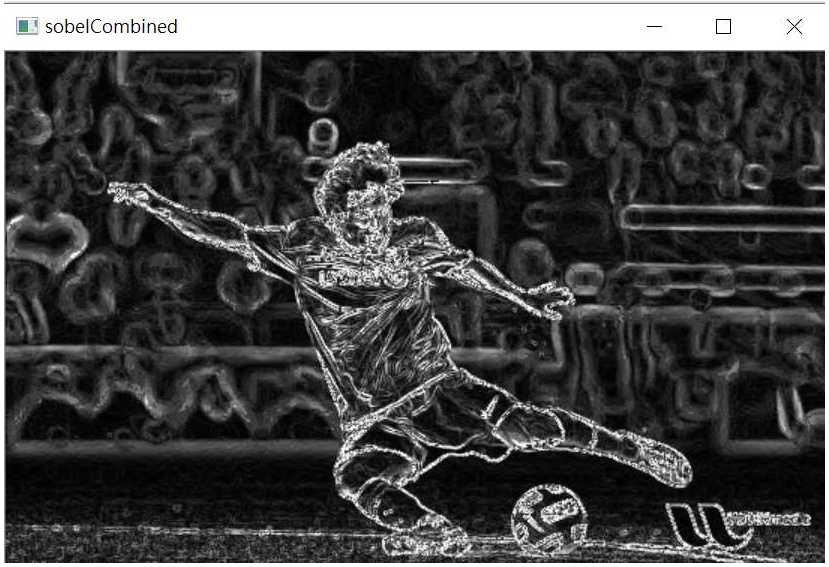
LoG



```
import cv2
import numpy as np

#img = cv2.imread("messi5.jpg", cv2.IMREAD_GRAYSCALE)
img = cv2.imread("sudoku.png", cv2.IMREAD_GRAYSCALE)
kernel = np.array([[0, 1, 1, 2, 2, 1, 0, 0, 0],[1, 2, 4, 5, 5, 5, 4, 2, 1],
        [1, 4, 5, 3, 0, 3, 5, 4, 1],[2, 5, 3, -12, -24, -12, 3, 5, 2],
        [2, 5, 0, -24, -40, -24, 0, 5, 2],[5, 1, 1, -12, -24, -12, 3, 5, 2],
        [1, 4, 5, 3, 0, 3, 5, 4, 1],[1, 2, 4, 5, 5, 5, 4, 2, 1],
        [0, 1, 1, 2, 0, 0, 0, 1, 0]], dtype = np.float32)
dst = cv2.filter2D(img,-1, kernel)

cv2.imshow('LoG',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




Laplacian


sobelCombined


LoG

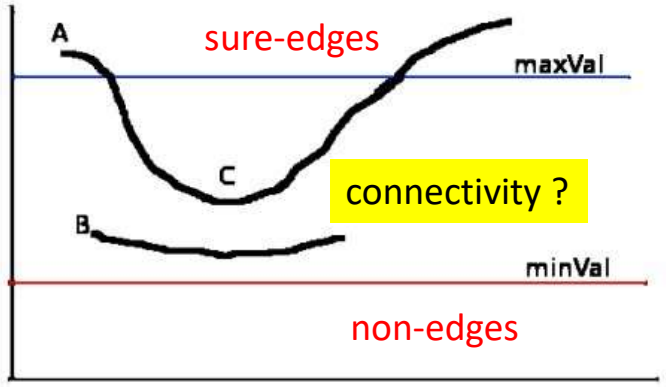

dst

20. Canny Edge Detection

It was developed by John F. Canny in 1986. It is a multi-stage algorithm
The Canny edge detection algorithm is broken down to 5 steps:

1 Noise reduction: eg., to remove the noise in the image with a 5x5 Gaussian filter.
2 Gradient calculation: with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (Gx) and vertical direction (Gy).
3 Non-maximum suppression: At every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).
4 Double threshold: two threshold values, minVal and maxVal.
5 Edge Tracking by Hysteresis:

Output = binary image [0, 255]



maxVal/minVal = 2~3

cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])

(dst)

canny = cv2.Canny(img, 100, 200)

apertureSize: 就是Sobel運算子的大小。

L2gradient: Boolean 參數

True: 使用精確的L2範數進行計算（即兩個方向的導數的平方和再開根號）;

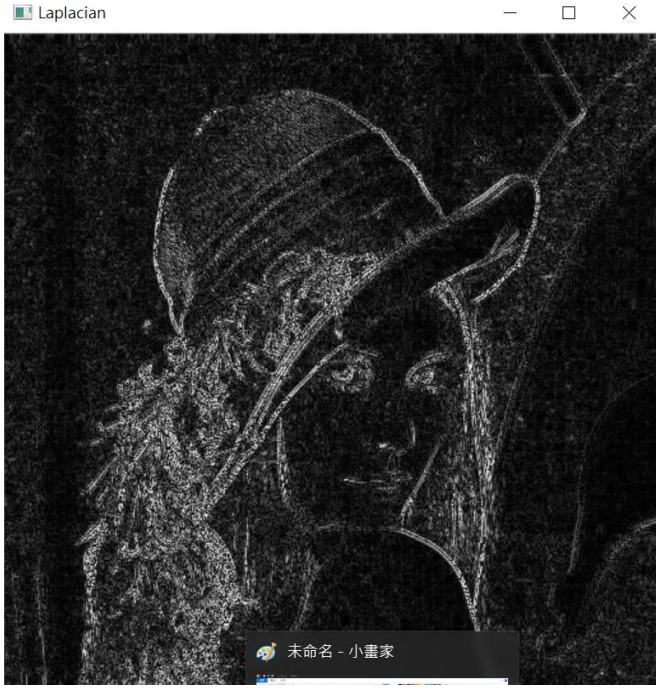
$$G = \sqrt{G_x^2 + G_y^2}$$

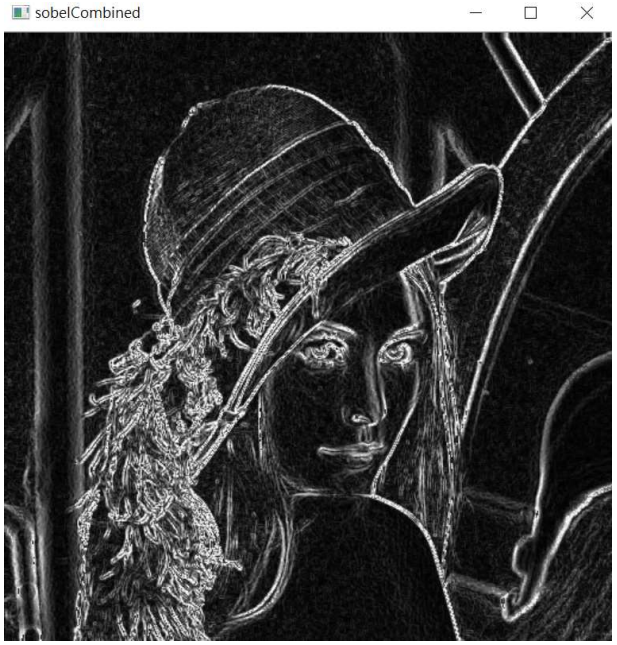
False 使用L1範數（直接將兩個方向導數的絕對值相加）。

$$G = |G_x| + |G_y|$$

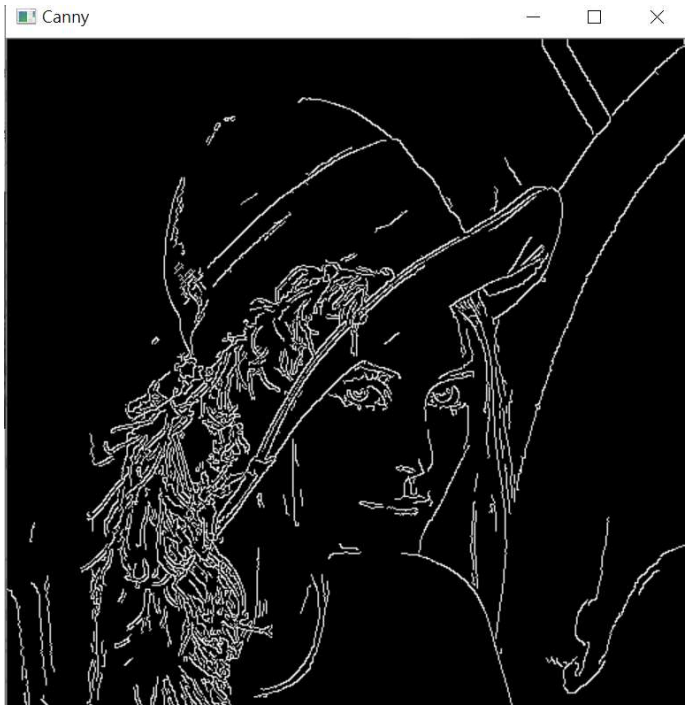
ex24.py

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

img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
lap = cv2.Laplacian(img, cv2.CV_64F, ksize=3)
lap = np.uint8(np.absolute(lap))
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1)
canny = cv2.Canny(img,100,200)

sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))

sobelCombined = cv2.bitwise_or(sobelX, sobelY)

titles = ['image', 'Laplacian', 'sobelX', 'sobelY', 'sobelCombined', 'Canny
images = [img, lap, sobelX, sobelY, sobelCombined, canny]
for i in range(6):
    plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()

cv2.imshow('image',img)
cv2.imshow('Laplacian',lap)
cv2.imshow('sobelX',sobelX)
cv2.imshow('sobelY',sobelY)
cv2.imshow('sobelCombined',sobelCombined)
cv2.imshow('Canny',canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





binary image



細緻

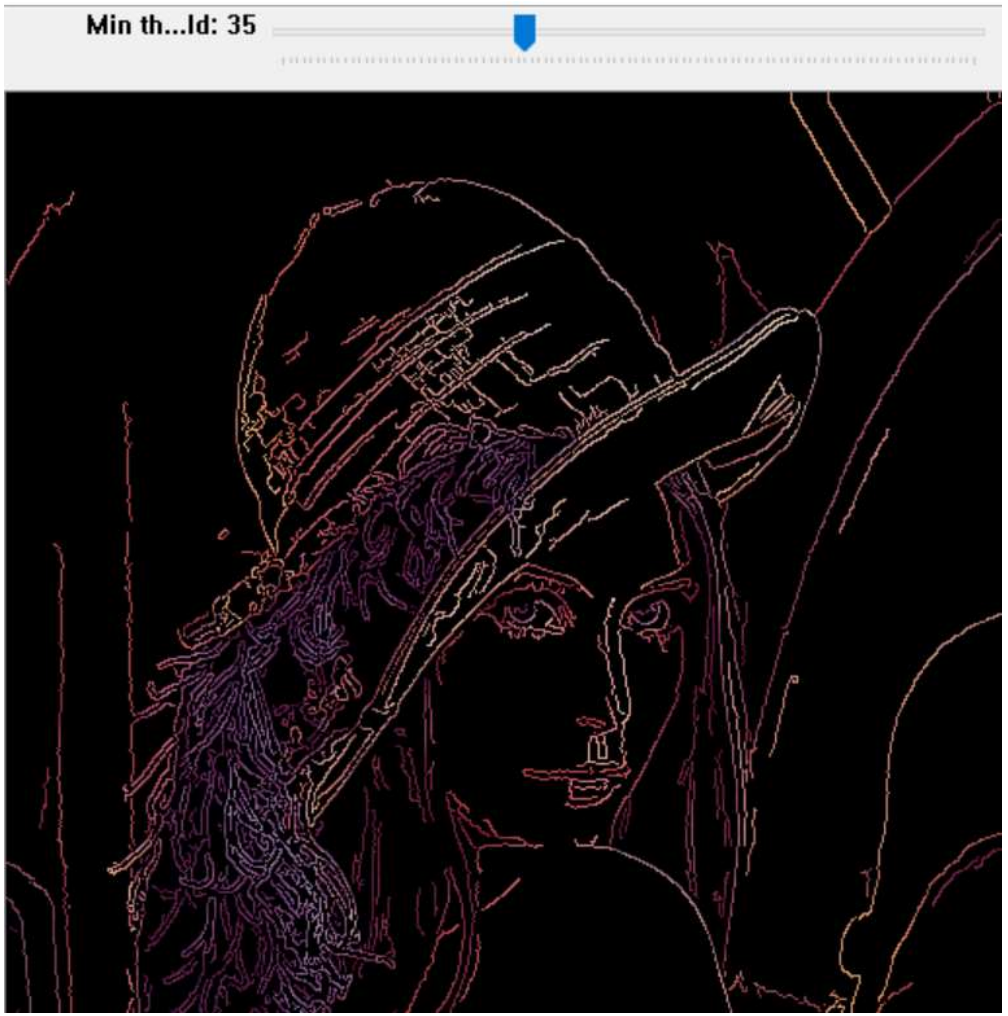
(see array)

Use Trackbar to dynamically changing the thresholds

ex25.py

```
8  import cv2
9
10 def CannyThreshold(lowThreshold):
11     detected_edges = cv2.GaussianBlur(gray,(3,3),0)
12     detected_edges = cv2.Canny(detected_edges,lowThreshold,lowThreshold*ratio,apertureSize = kernel_size)
13     dst = cv2.bitwise_and(img,img,mask = detected_edges) # just add some colours to edges from original image.
14     cv2.imshow('canny demo',dst)
15
16 lowThreshold = 0
17 max_lowThreshold = 100
18 ratio = 3
19 kernel_size = 3
20
21 img = cv2.imread('lena.jpg')
22 gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
23
24 cv2.namedWindow('canny demo')
25 cv2.createTrackbar('Min threshold','canny demo',lowThreshold, max_lowThreshold, CannyThreshold)
26
27 CannyThreshold(0)  # initialization
28 if cv2.waitKey(0) == 27:
29     cv2.destroyAllWindows()
```

detected_edges


Min th...ld: 35