

파이썬과 OpenCV를 이용한 영상처리 원리와 응용



A4 Screen

이 슬라이드에서 사용한 서체 :

- Open Sans(<https://ko.cooltext.com/Download-Font-Open+Sans>)
- KoPubWorld돋움체(<http://www.kopus.org/biz/electronic/font.aspx>)



1장. 영상처리 개요

영상처리 개요에 대해 설명합니다.

- 1절. 이미지 표현
- 2절. OpenCV와 이미지 읽기
- 3절. 디지털 이미지와 컬러

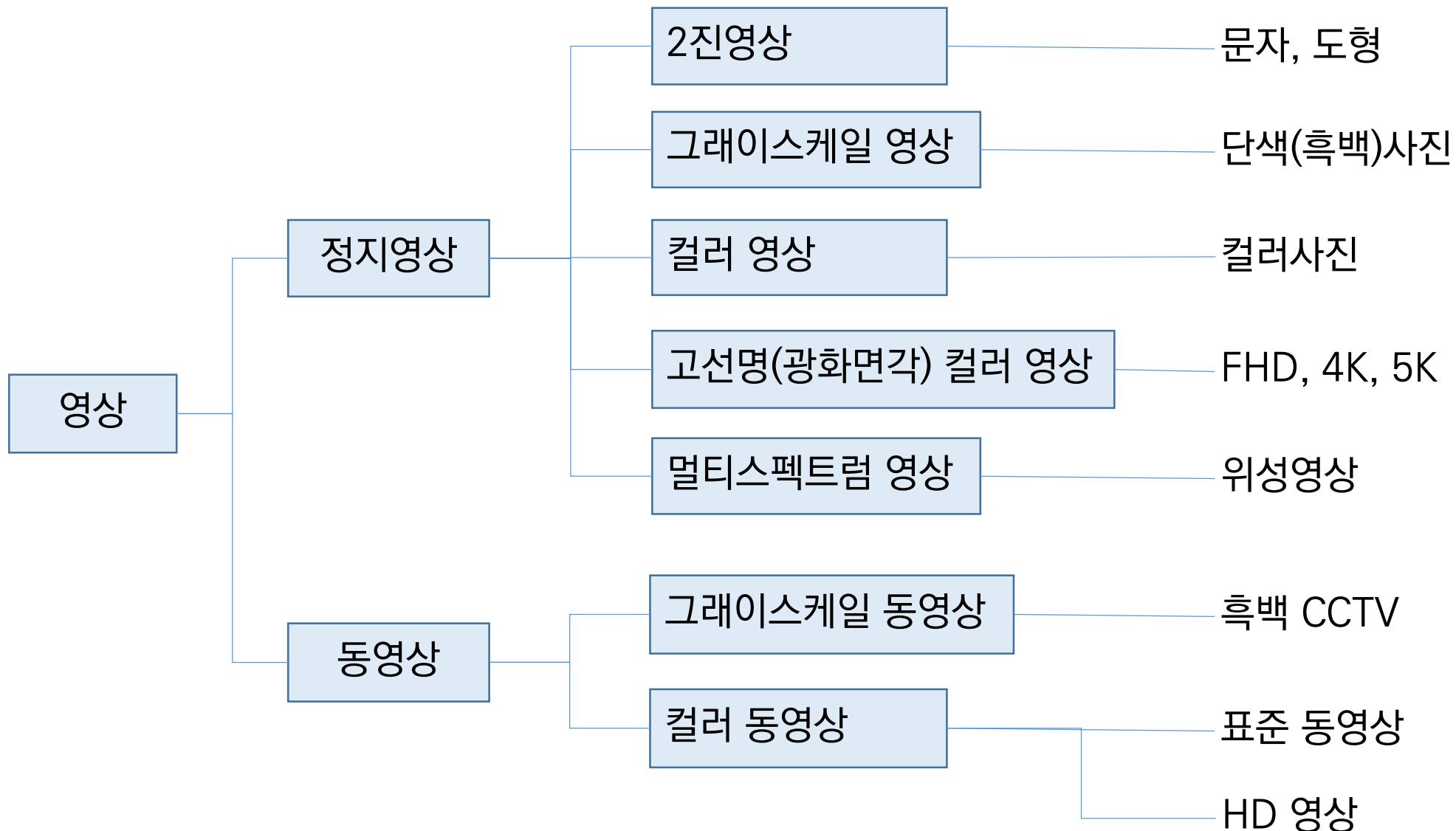
1장. 영상처리 개요



1절. 이미지 표현

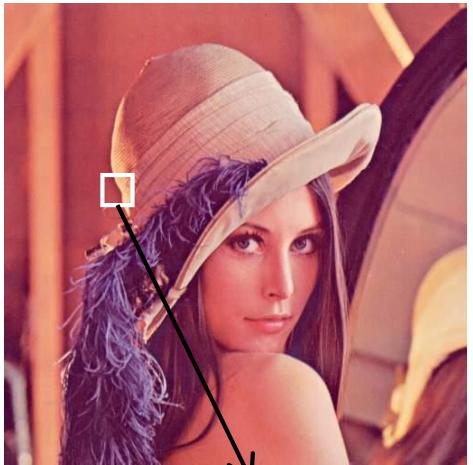
파이썬 OpenCV를 이용한
영상처리

저자 허진경

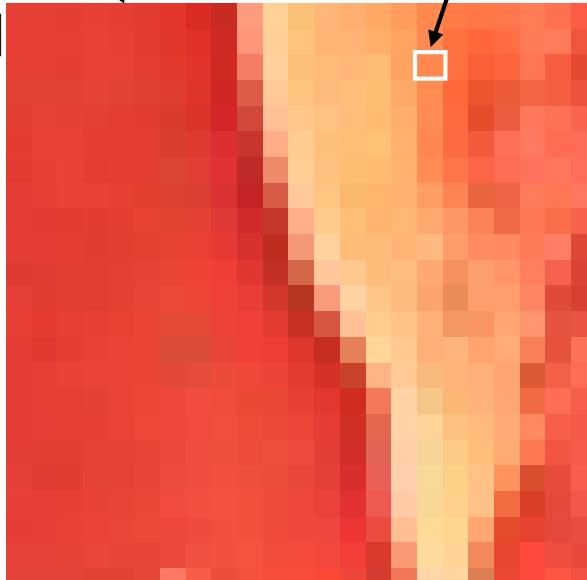


01 디지털 이미지와 좌표 표현 방법

사진, 도형 등 아날로그 이미지

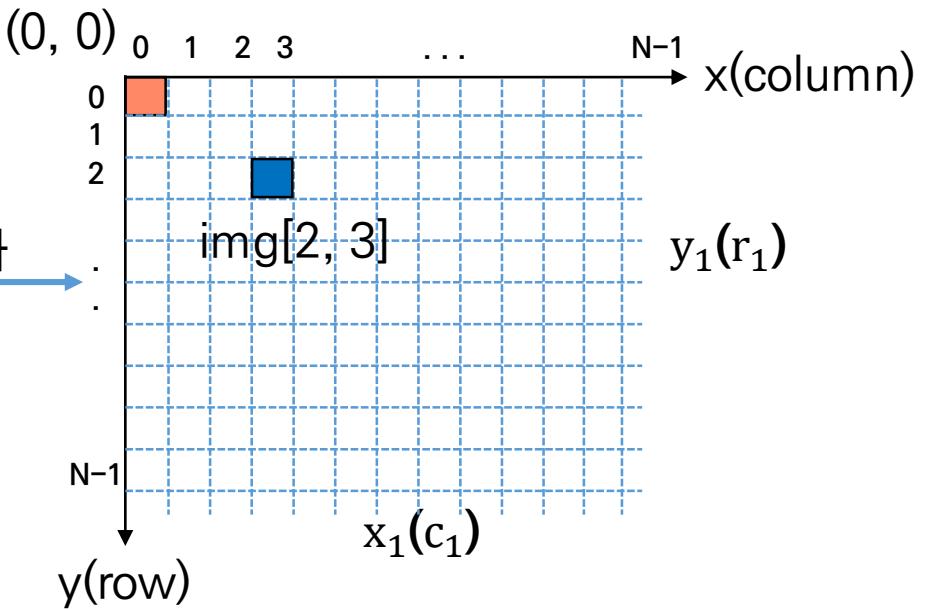


확대



디지털화

Pixel(화소)



표본화 : 공간적으로 연속되는 화상을 이산적인 화소의 집합으로 분할하는 것

같은 이미지라도 가로*세로의 크기가 큰 이미지가 더 선명

양자화 : 이미지의 농담을 이산적인 정수값으로 변환하는 조작

흑(1) 또는 백(0)으로 하는 2진(1비트)로 양자화 하는 것은 2진 화상
화소를 저장하는 비트 수를 증가시킬수록 표현이 풍부해짐
-> 6비트 이상일 경우 사람의 눈으로 거의 구별할 수 없음
-> 컴퓨터에서 다룰 수 있도록 고려하여 256단계(8비트, 1바이트/화소)를 사용
-> 화소의 농담이 0~255의 수치로 표현 됨

01

레나 이미지

레나(Lenna 또는 Lena) 이미지는 플레이보이 잡지 1972년 11월자 센터폴드(잡지 중간에 접어서 들어가 있는 페이지)에 실린 스웨덴의 모델인 레나 포르센(1951년 3월 31일 -)의 사진의 일부분



1장. 영상처리 개요



2절. OpenCV와 이미지 읽기

파이썬 OpenCV를 이용한
영상처리

저자 허진경

01 OpenCV

Open Source Computer Vision Library

C++, Python, Java, MATLAB 인터페이스 제공

<https://opencv.org/>

- 파이썬용 OpenCV 설치
`pip install opencv-python`
- OpenCV 주요 모듈과 함께 추가 모듈도 설치
`pip install opencv-contrib-python`
- OpenCV를 사용하려면 넘파이 패키지가 설치되어 있어야 함
`pip install numpy`

```
1. import cv2  
2. print(cv2.__version__)
```

4.2.1



cv2.imread(fileName, flags)

- ▶ fileName (str) – 이미지파일의 경로
- flag (int) – 이미지 파일을 읽을 때의 Option.
- ▶ Return : numpy.ndarray



cv2.imread()의 flags

- ▶ cv2.IMREAD_COLOR : 이미지 파일을 컬러로 읽어들입니다. 투명한 부분은 무시되며, 기본값입니다.
- ▶ cv2.IMREAD_GRAYSCALE : 이미지를 그레이스케일로 읽어들입니다. 실제 이미지 처리시 중간단계로 많이 사용합니다.
- ▶ cv2.IMREAD_UNCHANGED : 이미지파일을 알파채널까지 포함하여 읽어 들입니다. 알파채널은 불투명도를 표시하며 값이 0이면 완전하게 투명합니다.
- ▶ 3개의 상수 대신 1, 0, -1을 사용해도 됩니다.

IMREAD_UNCHANGED	-1	설정된 경우로드 된 이미지를 그대로 (알파 채널이 있는 경우 자르기) 반환합니다. EXIF 방향을 무시합니다.
IMREAD_GRAYSCALE	0	설정하면 항상 이미지를 단일 채널 회색조 이미지로 변환합니다.(코덱 내부 변환).
IMREAD_COLOR	1	설정된 경우 항상 이미지를 3 채널 BGR 컬러 이미지로 변환합니다.
IMREAD_ANYDEPTH	2	설정된 경우 입력에 해당 깊이가 있으면 16 비트 / 32 비트 이미지를 반환하고, 그렇지 않으면 8 비트로 변환합니다.
IMREAD_ANYCOLOR	4	설정된 경우 이미지를 가능한 모든 색상 형식으로 읽습니다.
IMREAD_LOAD_GDAL	8	설정되어 있으면 gdal 드라이버를 사용하여 이미지를 로드합니다.
IMREAD_REDUCED_GRAYSCALE_2	16	설정하면 항상 이미지를 단일 채널 회색조 이미지로 변환하고 이미지 크기를 1/2로 줄입니다.
IMREAD_REDUCED_COLOR_2	17	설정하면 항상 이미지를 3 채널 BGR 컬러 이미지로 변환하고 이미지 크기를 1/2로 줄입니다.
IMREAD_REDUCED_GRAYSCALE_4	32	설정하면 항상 이미지를 단일 채널 회색조 이미지로 변환하고 이미지 크기를 1/4로 줄입니다.
IMREAD_REDUCED_COLOR_4	33	설정하면 항상 이미지를 3 채널 BGR 컬러 이미지로 변환하고 이미지 크기를 1/4로 줄입니다.
IMREAD_REDUCED_GRAYSCALE_8	64	설정하면 항상 이미지를 단일 채널 회색조 이미지로 변환하고 이미지 크기가 1/8로 줄어 듭니다.
IMREAD_REDUCED_COLOR_8	65	설정하면 항상 이미지를 3 채널 BGR 컬러 이미지로 변환하고 이미지 크기를 1/8로 줄입니다.
IMREAD_IGNORE_ORIENTATION	128	설정된 경우 EXIF의 방향 플래그에 따라 이미지를 회전시키지 않습니다. https://en.wikipedia.org/wiki/Exif

cv2.imshow(title, image)

- ▶ title (str) – 윈도우 창의 Title
image (numpy.ndarray) – cv2.imread() 의 return값

```
1 cv2.imshow("Lena", lena_img)
2 cv2.waitKey()
3 cv2.destroyAllWindows()
```

함께 사용하는 코드

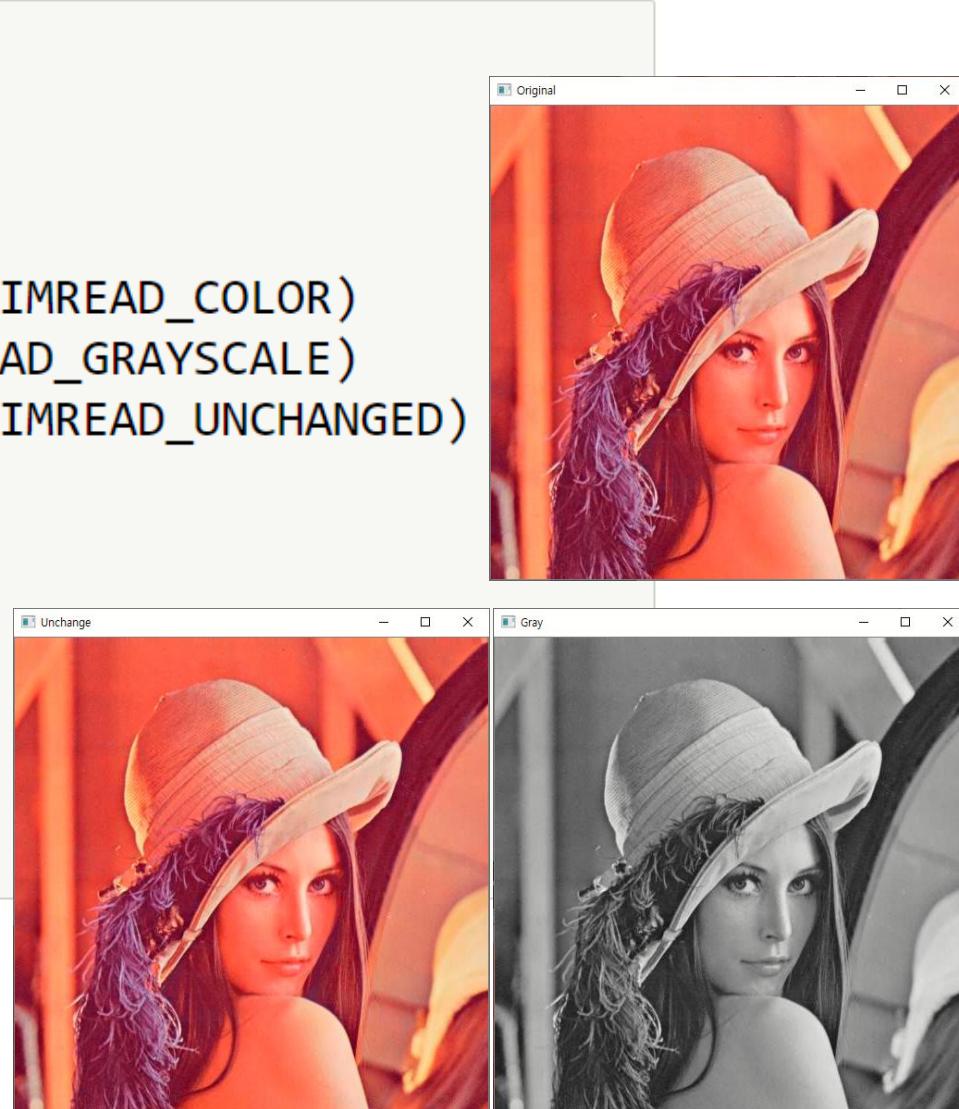


- ▶ cv2.waitKey() : keyboard입력을 대기하는 함수로 0이면 key입력까지 무한대기이며 특정 시간동안 대기하려면 milisecond값을 넣어주면 됩니다.
- ▶ cv2.destroyAllWindows() : 화면에 나타난 윈도우를 종료합니다.
- ▶ 일반적으로 위 3개는 같이 사용됩니다.

01 이미지 보기

```
1 import cv2  
2  
3 fname = 'lena.jpg'  
4  
5 original = cv2.imread(fname, cv2.IMREAD_COLOR)  
6 gray = cv2.imread(fname, cv2.IMREAD_GRAYSCALE)  
7 unchange = cv2.imread(fname, cv2.IMREAD_UNCHANGED)  
8  
9 cv2.imshow('Original', original)  
10 cv2.imshow('Gray', gray)  
11 cv2.imshow('Unchange', unchange)  
12  
13 cv2.waitKey(0)  
14 cv2.destroyAllWindows()
```

jpg 파일일 경우 unchange.shape[2] 는 3 이지만
투명화소 정보가 있는 png 파일일 경우 4임(alpha값 포함)

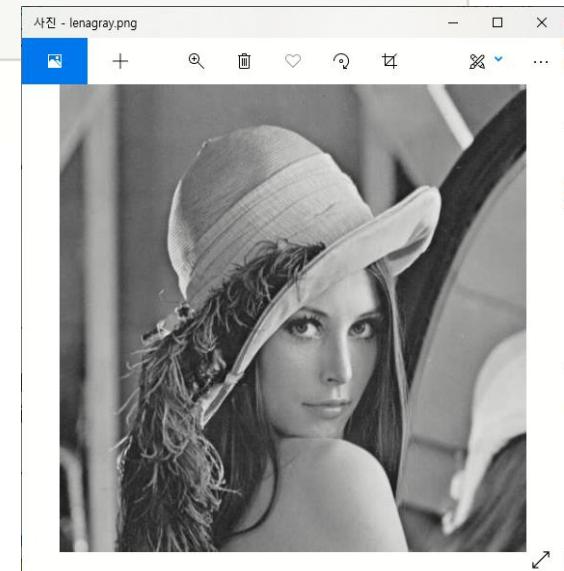


cv2.imwrite(fileName, image) → isSaved

- ▶ fileName (str) – 저장될 파일명
image – 저장할 이미지

```
1 import cv2  
2  
3 img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)  
4 cv2.imwrite('lenagray.png',img)
```

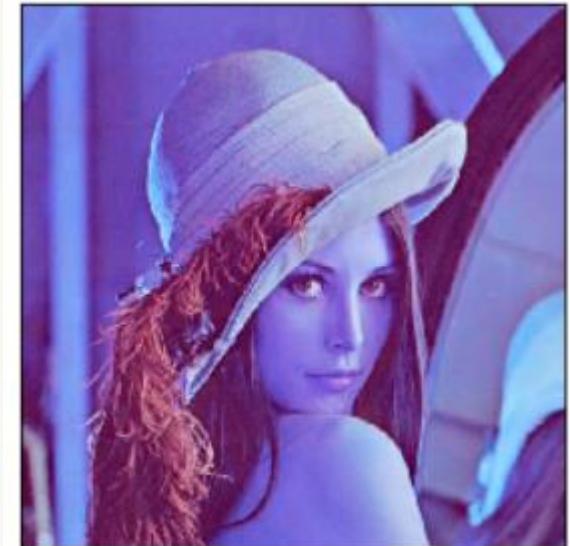
True



plt.imshow(image)

- ▶ Matplotlib는 다양한 plot 기능을 가진 Python Plot Library입니다.
- ▶ 이미지를 zoom하거나 하나의 화면에 여러개의 이미지를 보고자 할 때 유용합니다.

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 from matplotlib import pyplot as plt
4 %matplotlib inline
5
6 img = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
7
8 plt.imshow(img)
9 plt.xticks([]) # x축 눈금
10 plt.yticks([]) # y축 눈금
11 plt.show()
```



원본은 붉은색 계열인데, 결과는 파란색 계열로 나타납니다.

이유는 openCV는 BGR로 사용하지만, Matplotlib는 RGB로 이미지를 보여주기 때문입니다.
즉 결과 값은 3차원 배열의 값 중 첫 번째와 세 번째 배열값을 서로 바꿔 주어야 합니다.

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
6
7 b, g, r = cv2.split(img)    # img파일을 b,g,r로 분리
8 img2 = cv2.merge([r,g,b]) # b, r을 바꿔서 Merge
9
10 plt.imshow(img2)
11 plt.xticks([]) # x축 눈금
12 plt.yticks([]) # y축 눈금
13 plt.show()
```



1장. 영상처리 개요



3절. 디지털 이미지와 컬러

파이썬 OpenCV를 이용한
영상처리

저자 허진경

Binary Image는 pixel당 1bit로 표현하는 영상을 의미합니다.

즉 흰색과 검은색으로만 표현이 되는 영상입니다.

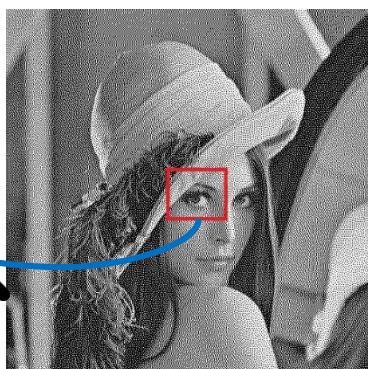
회색조 이미지



2진 이미지



원본 이미지를
thresholding 처리를
하여 binary
image로 변환한 결과
(threshold=120)



dithering : 밀도를
조절하여 밝기를 표현
하는 방법

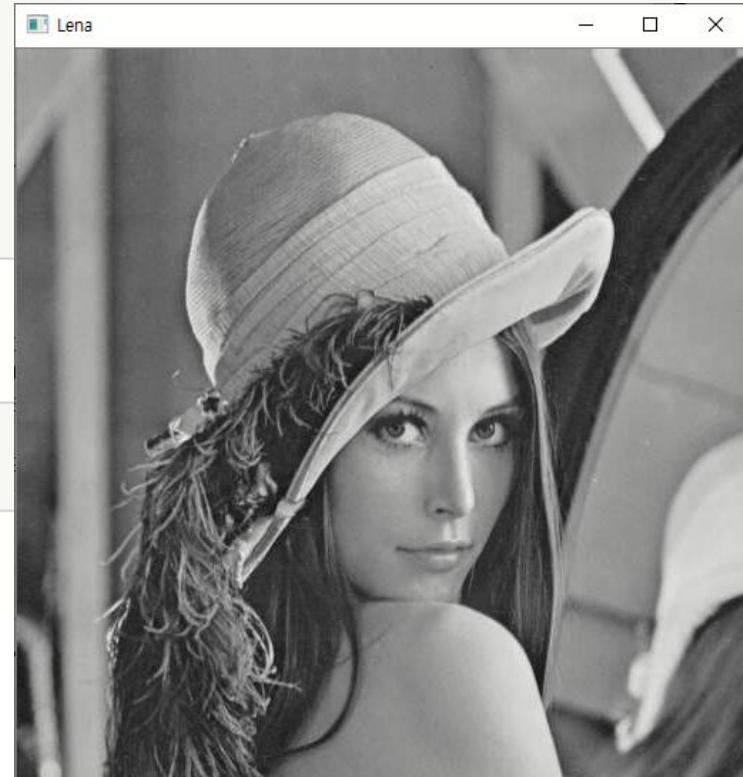
Pixel당 8bit, 즉 256단계의 명암(빛의 세기)을 표현할 수 있는 이미지입니다.

```
1 lena_grey = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2 print(lena_grey.shape)
3 cv2.imshow("Lena", lena_grey)
4 cv2.waitKey()
5 cv2.destroyAllWindows()
```

(512, 512)

```
1 lena_grey[0][0]
```

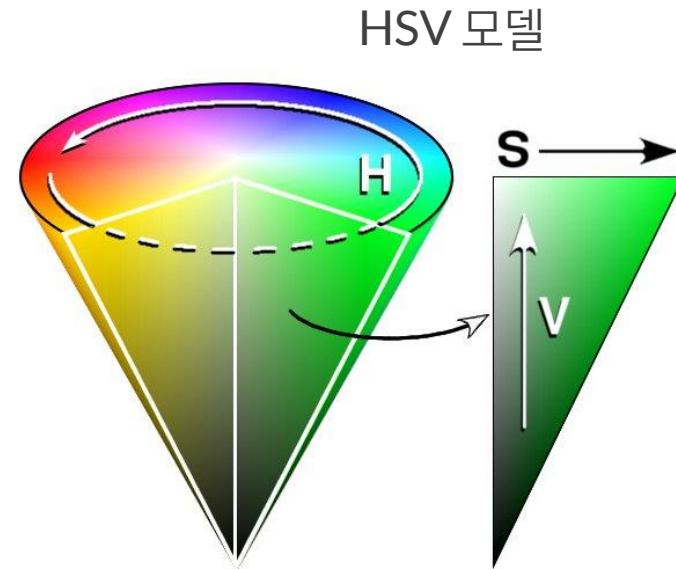
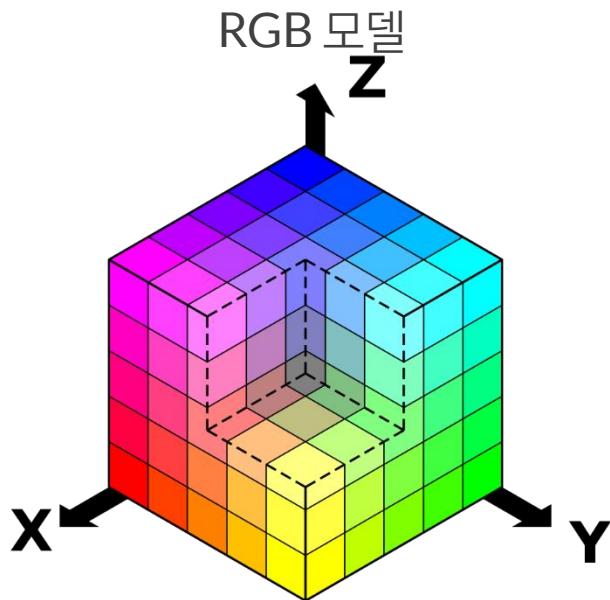
162





Color Image

- ▶ Color 이미지는 pixel의 색을 표현하기 위해서 pixel당 24bit를 사용합니다. 총 16,777,216 가지의 색을 표현할 수 있습니다.
- ▶ 이것을 일반적으로 True color image라고 합니다. pixel은 RGB 각각을 위해서 8bit를 사용하게 됩니다.
- ▶ OpenCV에서는 BGR로 표현을 하기 때문에 Blue→(255,0,0), Green→(0,255,0), Red→(0,0,255), White→(255,255,255), Black→(0,0,0)으로 표현할 수 있습니다.
- ▶ 각 pixel당 3Byte를 사용하기 때문에 용량이 큽니다. 이를 해결하기 위해서 lookup table을 사용하여, 해당 pixel에는 index만 저장하기도 합니다.



HSV의 의미는 다음과 같습니다.

- ▶ Hue : 색상. 일반적인 색을 의미함. 원추모형에서 각도로 표현이 됨.
(0: Red, 120도 : Green, 240: Blue)
- ▶ Saturation : 채도. 색의 순수성을 의미하며 일반적으로 ‘짙다’, ‘흐리다’로 표현이 됨. 중심에서 바깥쪽으로 이동하면 채도가 높음.
- ▶ Value : 명도. 색의 밝고 어두운 정도. 수직축의 깊이로 표현. 어둡다 밝다로 표현이 됨.

By Wapcaplet - From en wiki, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=308191>

cv2.cvtColor(src, code[, dst[, dstCn]])

OpenCV에는 150여 가지 변환 방법이 있습니다. 아래는 변환 방법을 확인하는 코드입니다.

```
flags = [flag for flag in dir(cv2) if flag.startswith('COLOR_')]  
print(flags)
```

```
['COLOR_BAYER_BG2BGR', 'COLOR_BAYER_BG2BGRA', 'COLOR_BAYER_BG2BGR_EA', 'COLOR_BAYER_BG2BGR_VNG', 'COLOR_BAYER_BG2GRAY', 'COLOR_BAYER_BG2RGB', 'COLOR_BAYER_BG2RGBA', 'COLOR_BAYER_BG2RGB_EA', 'COLOR_BAYER_BT2020']
```

밝기 정보(Y)만을 따로 분리해서 전송하고, 색깔 정보(U, V)는 따로 보내는 방법을 사용

컬러 영상을 Gray 영상으로 변환하는 공식은 YPbPr, YCbCr, YIQ 등이 있음

YCrCb : $Y = \text{Red} * 0.2126 + \text{Green} * 0.7152 + \text{Blue} * 0.0722$

YPrPb : Y = Red * 0.299 + Green * 0.587 + Blue * 0.114

HSL, HSV : http://en.wikipedia.org/wiki/HSL_and_HSV

YCrCb : <http://en.wikipedia.org/wiki/YCbCr>

```
1 lena_img = cv2.imread("lena.jpg")
2 B, G, R = lena_img[0][0]
3 print(lena_img[0][0])
4 print(0.114*B + 0.587*G + 0.299*R)      YPrPb
5 print(0.0722*B + 0.7152*G + 0.2116*R)    YCrCb
6 lena_grey = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
7 print(lena_grey[0][0])
```

[127 136 226]

161.884

154.2582

162

1장. 영상처리 개요



4절. 동영상 처리하기

파이썬 OpenCV를 이용한
영상처리

저자 허진경

 cv2.VideoCapture 클래스와, read() 메서드로 프레임 처리

```
cap = cv2.VideoCapture("Puppies-HD.mp4")

if cap.isOpened():
    delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
    while True:
        ret, img = cap.read()
        if ret:
            cv2.imshow("Movie", img)
            if cv2.waitKey(delay) & 0xFF == 27 :
                print("ESC Key pressed")
                break
        else:
            print(ret, img)
            break
    else:
        print("File not opened")
```

cap = cv2.VideoCapture(0)

숫자 0은 0번 USB 카메라를 연결하겠다는 의미입니다.

```
1 import cv2
2
3 cap = cv2.VideoCapture('0')
4
5 if cap.isOpened():
6     delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
7     while True:
8         ret, img = cap.read()
9         if ret:
10             img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
11             cv2.imshow("Movie", img_gray)
12             if cv2.waitKey(delay) & 0xFF == 27 :
13                 print("ESC Key pressed")
14                 break
15             else:
16                 print(ret, img)
17                 break
18     else:
19         print("File not opened")
20
21 cap.release()
22 cv2.destroyAllWindows()
```



```
cap = cv2.VideoCapture("rtsp://admin:admin12@111.201.13.53:553/profile1")
```



rtsp://사용자아이디:비밀번호@아이피주소:포트번호/프로파일

- ▶ 다음은 코드 예입니다.

```
rtsp_addr = "rtsp://admin:admin12@111.201.13.53:553/profile1"
```

```
cap = cv2.VideoCapture(rtsp_addr)
```

- ▶ admin:admin12는 RTSP 스트리밍에 연결할 사용자 아이디와 비밀번호 예시입니다.

- ▶ 아이피 주소와 포트번호, 그리고 프로파일 이름은 다를 수 있습니다.

- ▶ RTSP 주소는 다음 형식처럼 사용할 수 있습니다.

```
rtsp://192.168.0.100:554/user=admin&password=admin12&channel=1&stream=0.sdp
```

cv2.CAP_PROP_로 시작하는 상수들은 동영상의 속성들에 관한 정보

VideoCapture 클래스의 set(id, value) 메서드와 get(id) 메서드를 이용하면 동영상 속성 정보를 변경하거나 조회

```
1 import cv2
2
3 cap = cv2.VideoCapture(0)
4 print(cap.get(cv2.CAP_PROP_FRAME_WIDTH),
      cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
5 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
6 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
7 print(cap.get(cv2.CAP_PROP_FRAME_WIDTH),
      cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
8
9 if cap.isOpened():
10     delay = int(1000 / cap.get(cv2.CAP_PROP_FPS))
```

cv2.imwrite()를 이용해서 이미지 파일로 저장

```
import cv2
import numpy as np
from datetime import datetime

def mouseHandler(event, x, y, flags, param):
    if event==cv2.EVENT_LBUTTONDOWN:
        print(event, x, y)
        print(datetime.today())
        filename = str(datetime.today().microsecond)+".jpg"
        cv2.imwrite(filename, img)

cv2.namedWindow("Movie")
cv2.setMouseCallback("Movie", mouseHandler)
```

이벤트 처리는 다음 절에서 설명됩니다.



cv2.VideoWriter(outputFile, fourcc, frame, size) → retval

- ▶ outputFile – 저장될 파일명을 지정합니다.
- ▶ fourcc – 비디오의 코덱(Codec)정보를 지정합니다. 코덱 정보는 cv2.VideoWriter_fourcc('M','J','P','G') 또는 cv2.VideoWriter_fourcc(*'MJPG') 와 같이 표현할 수 있습니다. 자주 사용하는 코덱은 DIVX, XVID, FMP4, X264, MJPG이며, OS마다 지원하는 코덱은 다릅니다. 더 많은 코덱 정보는 다음 사이트에서 확인할 수 있습니다.
* <https://www.fourcc.org/codecs.php>
- ▶ frame – 초당 저장될 프레임의 수를 지정합니다.
- ▶ size – 저장될 비디오의 크기를 (width, height) 형식으로 지정합니다.
- ▶ 반환값 retval – VideoWriter 객체입니다. 이 객체의 write()함수를 이용해서 비디오를 저장합니다.

```
5 if cap.isOpened():
6     fourcc = cv2.VideoWriter_fourcc(*'DIVX')
7     width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
8     height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
9     size = (int(width), int(height))
10    fps = cap.get(cv2.CAP_PROP_FPS)
11    out = cv2.VideoWriter("video.avi", fourcc, fps, size)
12
13    delay = int(1000/cap.get(cv2.CAP_PROP_FPS))
14    while True:
15        ret, img = cap.read()
16        if ret:
17            gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
18            cv2.imshow("Puppy", gray)
19            out.write(cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR))
```

1장. 영상처리 개요



5절. 이벤트 처리하기

파이썬 OpenCV를 이용한
영상처리

저자 허진경

windowName

이벤트 소스
(윈도우)

def callback():

이벤트 핸들러
(콜백 함수)

이벤트 소스와 이벤트 핸들러 연결

cv2.setMouseCallback(windowName, callback)



cv2.setMouseCallback(windowName, callback, param=None)

- ▶ windowName – 윈도우의 이름(Title)을 지정합니다.
- ▶ callback – 이벤트를 처리할 함수(콜백함수에는 (event, x, y, flags, param)이 전달됩니다.
- ▶ param – 콜백함수에 전달되는 데이터입니다.

***callback(event, x, y, flags, param)***

- ▶ event – cv2.EVENT_로 시작하는 마우스 이벤트 속성이며 [표 16. 마우스 이벤트 속성]을 참고하세요.
- ▶ x,y – 마우스 이벤트가 발생된 위치입니다.
- ▶ flags – 이벤트가 발생할 때 키보드나 마우스의 추가적인 상태를 알려 줍니다. cv2.EVENT_FLAG_로 시작하는 상수 6가지가 있으며 이들의 조합에 따른 결과값이 전달됩니다. 만일 컨트롤키와 쉬프트키를 누른 상태에서 마우스 오른쪽 버튼을 클릭한다면 $8+16+1=25$ 가 전달됩니다.

이름	값	의미
EVENT_FLAG_LBUTTON	1	마우스 왼쪽 버튼 누름
EVENT_FLAG_RBUTTON	2	마우스 오른쪽 버튼 누름
EVENT_FLAG_MBUTTON	4	마우스 가운데 버튼 누름
EVENT_FLAG_CTRLKEY	8	컨트롤(Ctrl) 키 누름
EVENT_FLAG_SHIFTKEY	16	쉬프트(Shift) 키 누름
EVENT_FLAG_ALTHOOK	32	알트(Alt) 키 누름

cv2.waitKey(delay=None) → retval

- ▶ delay – 밀리초(ms)단위 대기시간입니다. 1초=1000밀리초입니다. 만일 delay 값이 기본값(None)이거나 0보다 작거나 같으면(delay<=0) 무한 대기합니다.
- ▶ retval – 이 함수가 반환하는 값은 키보드로 입력한 키의 아스키코드³⁾입니다. 만일 입력 대기시간동안 아무 키도 눌리지 않으면 -1을 반환합니다. 아스키코드 중에서 10진수 27(16진수 1B)은 ESC키를 의미합니다. 0은 48, A는 65, a는 97에 해당하는 아스키코드값을 가지고 있습니다.

```
1 import cv2
2 import numpy as np
3 image = np.full((100, 100, 3), 255, np.uint8)
4
5 while(1):
6     cv2.imshow("Key test", image)
7     key = cv2.waitKey(0)
8     print(f'Code: {key}, Char: {chr(key)}')
9     if key & 0xFF == 27:
10         break
11
12 cv2.destroyAllWindows()
```

**cv2.createTrackbar(trackbarName, windowName, value, count, onChange)**

- ▶ trackbarName – 트랙바 이름입니다.
- ▶ windowName – 트랙바가 등록될 윈도우 이름입니다.
- ▶ value – int 타입이며, 트랙바가 생성될 때 초기값입니다.
- ▶ count – 트랙바의 최댓값입니다. 최소값은 항상 0입니다.
- ▶ onChange – 트랙바의 슬라이드가 변경될 때 호출되는 콜백 함수입니다. 전달되는 파라미터는 트랙바의 위치입니다.

**cv2.getTrackbarPos(trackbarName, windowName) → retval**

- ▶ trackbarName – 트랙바 이름입니다.
- ▶ windowName – 트랙바가 등록된 윈도우 이름입니다.
- ▶ 반환값 retval – 트랙바의 위치를 반환합니다.

1장. 영상처리 개요



6절. 도형 그리기

파이썬 OpenCV를 이용한
영상처리

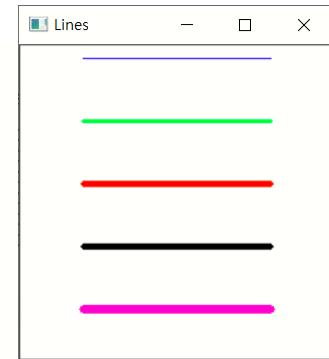
저자 허진경



```
cv2.line(img, start, end, color, thickness=1)
```

- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ start – 선의 시작 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ end – 선의 끝 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ color – BGR 형식의 색상값입니다. (B, G,R) 형식이며 (255,0,0)이면 Blue입니다.
- ▶ thickness – 선의 두께를 픽셀단위 정수로 지정합니다

```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6 cv2.line(img, (50, 10), (200, 10), (255, 0, 0), 1)
7 cv2.line(img, (50, 60), (200, 60), (0, 255, 0), 2)
8 cv2.line(img, (50, 110), (200, 110), (0, 0, 255), 3)
9 cv2.line(img, (50, 160), (200, 160), (0, 0, 0), 4)
10 cv2.line(img, (50, 210), (200, 210), (255, 0, 255), 5)
11
12 cv2.imshow('Lines', img)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```

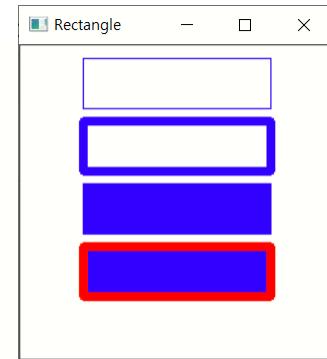




```
cv2.rectangle(img, start, end, color, thickness=1)
```

- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ start – 사각형의 왼쪽 위 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ end – 사각형의 오른쪽 아래 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ color – BGR 형식의 색상값입니다. (B, G, R) 형식이며 (255, 0, 0)이면 Blue입니다.
- ▶ thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 사각형의 안쪽을 채웁니다.

```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6 cv2.rectangle(img, (50, 10), (200, 50), (255, 0, 0))
7 cv2.rectangle(img, (50, 60), (200, 100), (255, 0, 0), 5)
8 cv2.rectangle(img, (50, 110), (200, 150), (255, 0, 0), -1)
9 cv2.rectangle(img, (50, 160), (200, 200), (255, 0, 0), -1)
10 cv2.rectangle(img, (50, 160), (200, 200), (0, 0, 255), 5)
11
12 cv2.imshow('Rectangle',img)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```

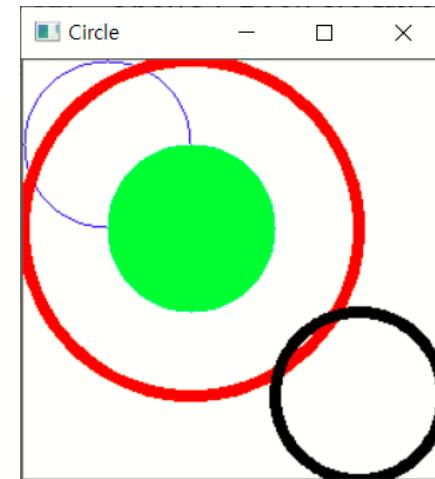




```
cv2.circle(img, center, radius, color, thickness=1)
```

- ▶ img - 도형을 그릴 이미지 객체입니다.
- ▶ center - 원의 중심 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ radius - 반지름을 픽셀단위 정수로 지정합니다.
- ▶ color - BGR 형식의 색상값입니다. (B, G,R) 형식이며 (255,0,0)이면 Blue입니다.
- ▶ thickness - 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 원의 안쪽을 채웁니다.

```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6 cv2.circle(img, (50, 50), 50, (255, 0, 0))
7 cv2.circle(img, (100, 100), 50, (0, 255, 0), -1)
8 cv2.circle(img, (100, 100), 100, (0, 0, 255), 5)
9 cv2.circle(img, (200, 200), 50, (0, 0, 0), 5)
10
11 cv2.imshow('Circle',img)
12 cv2.waitKey(0)
13 cv2.destroyAllWindows()
```



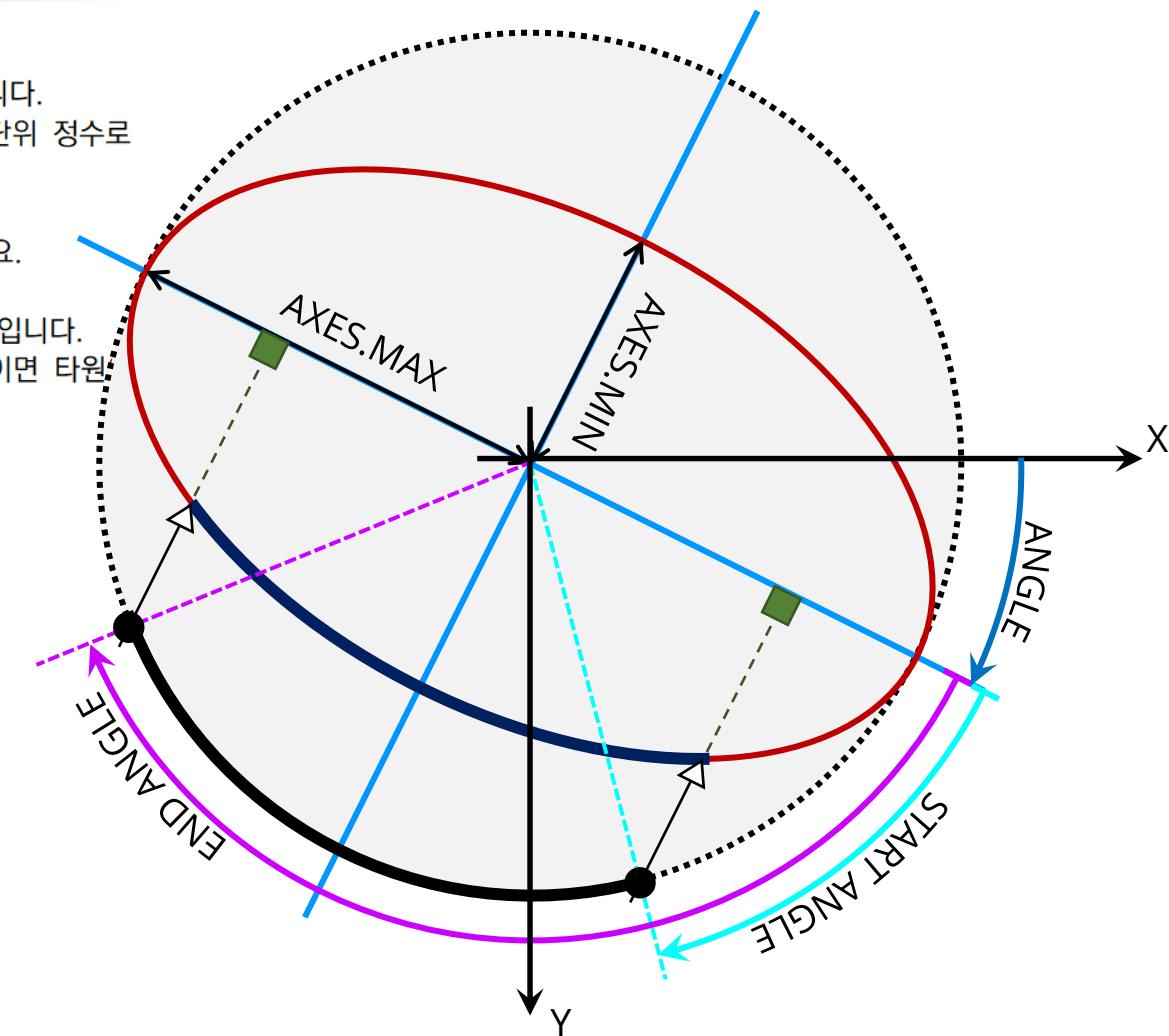


```
cv2.ellipse(img, center, axes, angle, startAngle, endAngle,  
color, thickness=1)
```

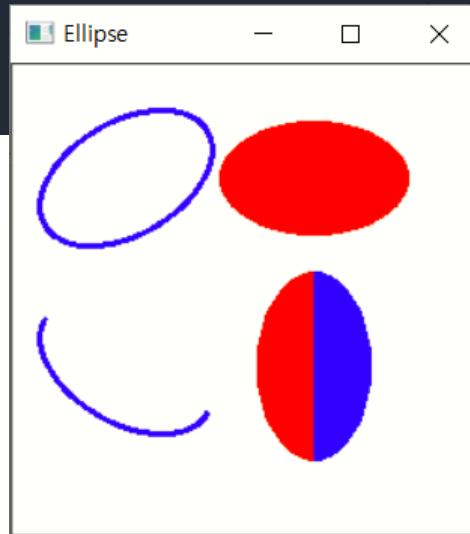
- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ center – 타원의 중심 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ axes – 타원의 중심에서 가장 큰 거리와 작은 거리를 픽셀단위 정수로 지정합니다. (max, min) 형식으로 지정합니다.
- ▶ angle – 타원의 기울기 각도입니다.
- ▶ startAngle – 호의 시작 각도입니다. 아래 그림을 참고하세요.
- ▶ endAngle – 호의 끝 각도입니다. 아래 그림을 참고하세요.
- ▶ color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- ▶ thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 타원의 안쪽을 채웁니다.

cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color, thickness=1)

- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ center – 타원의 중심 좌표입니다. (x, y) 형식으로 지정합니다.
- ▶ axes – 타원의 중심에서 가장 큰 거리와 작은 거리를 픽셀단위 정수로 지정합니다. (max, min) 형식으로 지정합니다.
- ▶ angle – 타원의 기울기 각도입니다.
- ▶ startAngle – 호의 시작 각도입니다. 아래 그림을 참고하세요.
- ▶ endAngle – 호의 끝 각도입니다. 아래 그림을 참고하세요.
- ▶ color – (B,G,R) 형식의 색상값입니다. (255,0,0)이면 Blue입니다.
- ▶ thickness – 선의 두께를 픽셀단위 정수로 지정합니다. -1이면 타원 안쪽을 채웁니다.



```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6 cv2.ellipse(img, (60,60), (50,30), -30, 0, 360, (255,0,0), 2)
7 cv2.ellipse(img, (160,60), (50,30), 0, 0, 360, (0,0,255), -1)
8 cv2.ellipse(img, (60,160), (50,30), 30, 0, 180, (255,0,0), 2)
9 cv2.ellipse(img, (160,160), (50,30), 90, 0, 180, (0,0,255), -1)
10 cv2.ellipse(img, (160,160), (50,30), 90, 180, 360, (255,0,0), -1)
11
12 cv2.imshow('Ellipse',img)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```

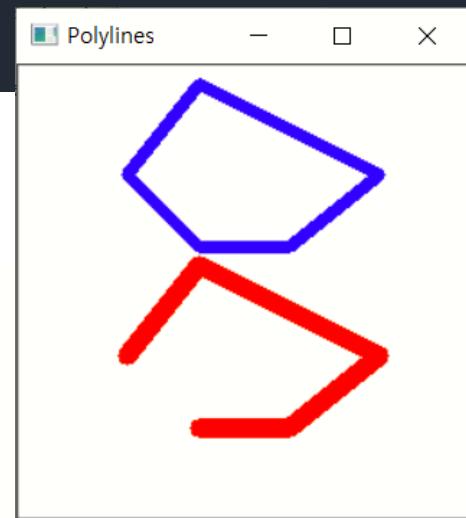




cv2.polyline(img, points, isClosed, color, thickness)

- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ points – 다각형 꼭지점의 좌표들입니다. [[[x1,y1], [x2,y2]....]] 형식으로 지정합니다.
- ▶ isClosed – True이면 닫힌 도형이 됩니다.
- ▶ color – BGR 형식의 색상값입니다. (B, G,R) 형식이며 (255,0,0)이면 Blue입니다.
- ▶ thickness – 선의 두께를 픽셀단위 정수로 지정합니다. 기본값은 1입니다. 음수는 사용할 수 없습니다.

```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5 points = np.array([[[60,60], [100,10], [200,60], [150,100],
6 [100, 100]]], dtype=np.int32)
7
8 cv2.polyline(img, points, True, (255, 0, 0), 5)
9 cv2.polyline(img, points+[0,100], False, (0, 0, 255), 10)
10
11 cv2.imshow('Polyline',img)
12 cv2.waitKey(0)
13 cv2.destroyAllWindows()
```

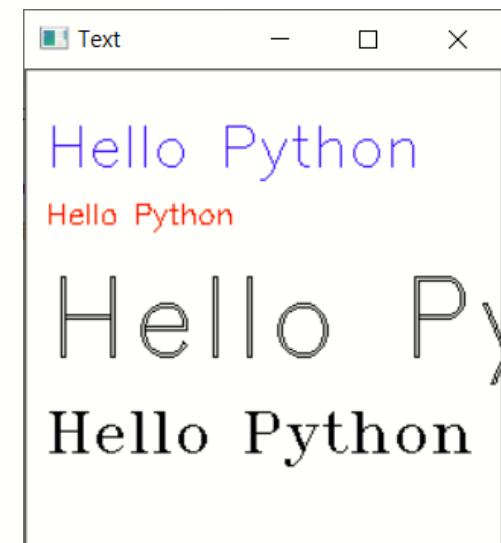




```
cv2.putText(img, text, org, font, fontScale, color)
```

- ▶ img – 도형을 그릴 이미지 객체입니다.
- ▶ text – 표시할 문자열입니다.
- ▶ org – 문자열이 표시될 위치입니다. 문자열의 왼쪽 아래 모서리 지점이 기준입니다.
- ▶ font – 폰트를 지정합니다. cv2.FONT_XXX 형식의 속성으로 지정합니다.
- ▶ fontScale – 폰트의 크기를 지정합니다.
- ▶ color – BGR 형식의 색상값입니다. (B, G,R) 형식이며 (255,0,0)이면 Blue입니다.

```
1 import cv2
2 import numpy as np
3
4 img = np.full((250, 250, 3), 255, dtype=np.uint8)
5
6 cv2.putText(img, "Hello Python", (10,50),
7             cv2.FONT_ITALIC, 1, (255,0,0))
7 cv2.putText(img, "Hello Python", (10,80),
8             cv2.FONT_ITALIC, 0.5, (0,0,255))
8 cv2.putText(img, "Hello Python", (10,150),
9             cv2.FONT_HERSHEY_DUPLEX, 2, (0,0,0))
9 cv2.putText(img, "Hello Python", (10,200),
10             cv2.FONT_HERSHEY_TRIPLEX, 1, (0,0,0))
10
11 cv2.imshow('Text',img)
12 cv2.waitKey(0)
13 cv2.destroyAllWindows()
```





2장. 영상 기본 연산

영상 기본 연산에 대해 설명합니다.

- 1절. 영상 기본 정보
- 2절. 채널과 영상
- 3절. 히스토그램

2장. 기본 연산



1절. 영상 기본 정보

파이썬 OpenCV를 이용한
영상처리

저자 허진경

OpenCV의 imread() 함수를 이용해서 이미지를 읽으면 넘파이 배열로 반환함

영상처리를 잘 하려면 넘파이 패키지를 잘 다뤄야 함

```
1 img = cv2.imread("lena.jpg")
```

```
1 print(img.shape)      512x512 크기 채널 3개인 이미지
```

(512, 512, 3)

```
1 print(img[0,0])      (0, 0)위치의 화소 값(BGR 순서)
```

[127 136 226]

```
1 print(img.size)
```

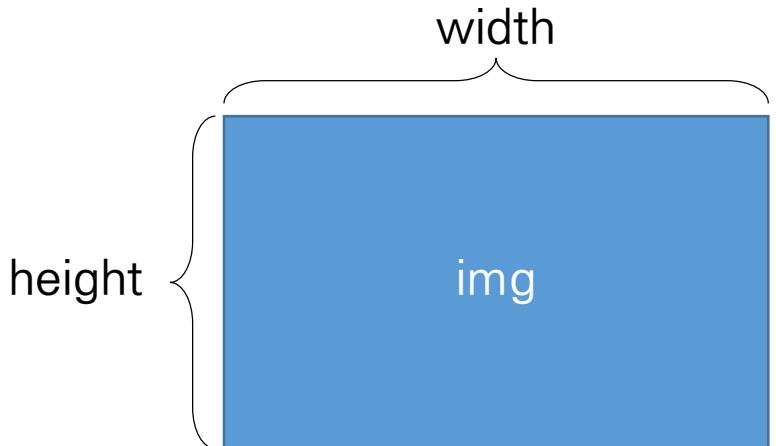
786432 $512 \times 512 \times 3 = 786,432$

```
1 print(img.dtype)
```

uint8

화소의 자료형은 np.uint8(Unsigned int)
부호를 갖지 않은 8비트 정수이므로 0~255표현

height, width = img.shape



반복문 실행 시 화소의 좌표는 img[y, x]

셀 실행시간 확인은 아래 명령으로 주피터 노트북
익스텐션 설치 후 ExecuteTime 선택

pip install jupyter_contrib_nbextensions
jupyter contrib nbextension install

짝수 행을 검은 선으로 처리하는 구문의 예

```
1 height, width = img.shape
```

```
1 for y in range(height) :
2     for x in range(width) :
3         if(y%2==0) :
4             img[y,x] = 0
```

executed in 57ms, finished 09:42:33 2020-06-18

```
1 for y in range(height) :
2     if(y%2==0) :
3         for x in range(width) :
4             img[y,x] = 0
```

executed in 27ms, finished 09:42:33 2020-06-18

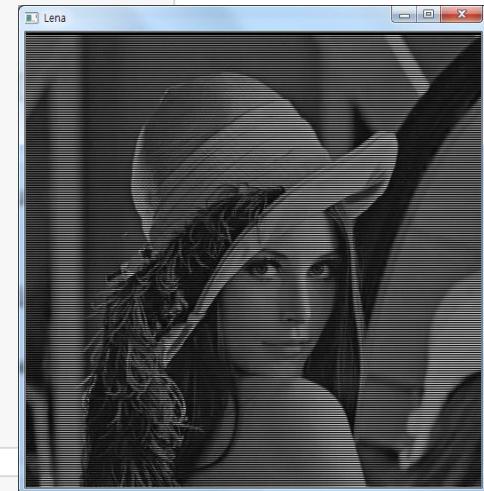
```
1 for y in range(height) :
2     if(y%2==0) :
3         img[y,:] = 0
```

executed in 4ms, finished 09:42:34 2020-06-18

```

1 import numpy as np
2
3
4 def black_lines(img):
5     height, width = img.shape
6     img_ = np.zeros(img.shape, dtype=np.uint8)
7     for y in range(height):
8         if(y % 2 == 0):
9             img_[y, :] = 0
10        else:
11            img_[y, :] = img[y, :]
12    return img_

```



```

1 def img_pro(func, img, *args, show=True, **kwargs):
2     img_ = func(img, *args, **kwargs)
3     if show:
4         cv2.imshow("Image", img_)
5         cv2.waitKey()
6         cv2.destroyAllWindows()
7     else:
8         return img_

```

```

1 img = cv2.imread("./images/lena.jpg", cv2.IMREAD_GRAYSCALE)
2 img_pro(black_lines, img)

1 t = img_pro(black_lines, img, show=False)

1 cv2.imshow("Lena", t)
2 cv2.waitKey()
3 cv2.destroyAllWindows()

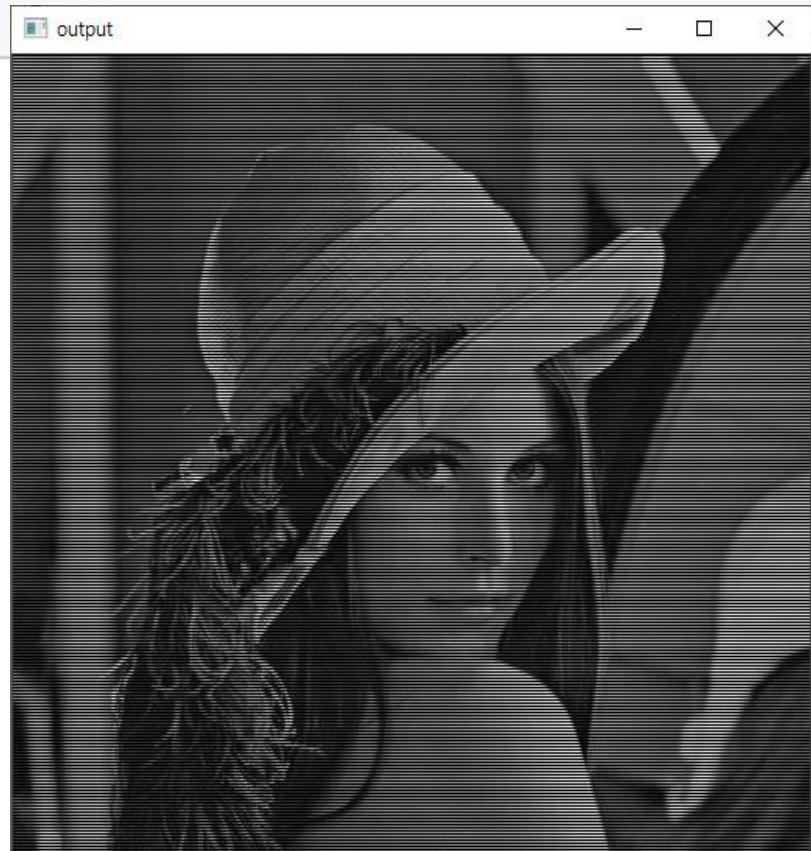
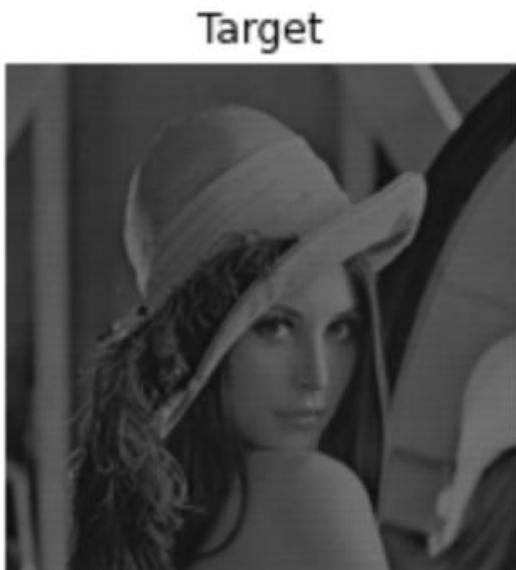
```

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 def img_pro2(func, img, *args, output=True, win=False, **kwargs):
5     img_ = func(img, *args, **kwargs)
6     if output:
7         if win:
8             cv2.imshow("Image", img_)
9             cv2.waitKey()
10            cv2.destroyAllWindows()
11        else:
12            fig, axes = plt.subplots(1,2)
13            axes[0].imshow(img, cmap="gray")
14            axes[0].axis("off"); axes[0].set_title("origin")
15            axes[1].imshow(img_, cmap="gray", interpolation=None)
16            axes[1].axis("off"); axes[1].set_title("target")
17    else:
18        return img_
```



02 Matplotlib을 이용한 출력 함수 사용

```
1 img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)  
2 img_pro2(black_lines, img)
```

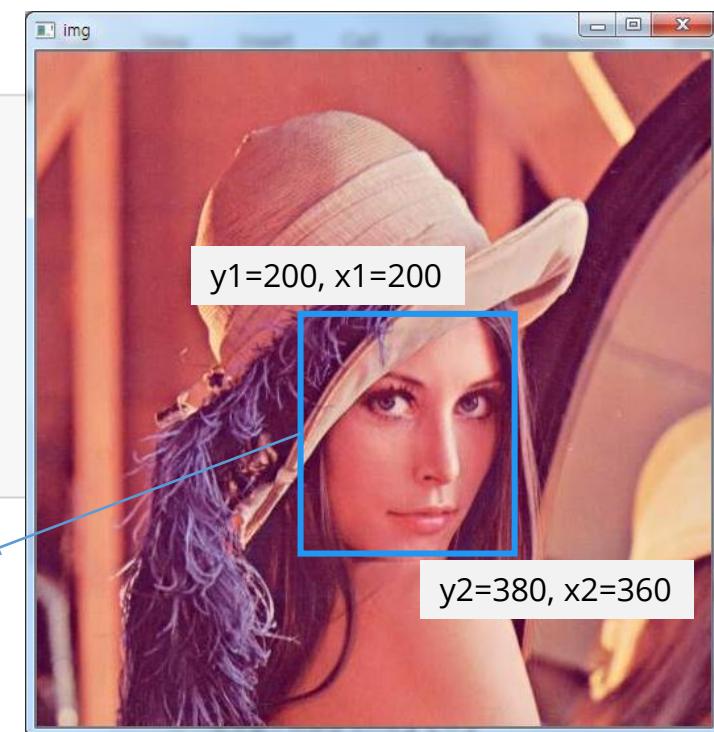


```
1 img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)  
2 img_pro2(black_lines, img, win=True)
```

이미지 작업시에는 특정 pixel단위 보다는 특정 영역단위로 작업을 하게 됩니다.

ROI 설정은 Numpy의 indexing방법을 사용합니다.

```
1 img = cv2.imread('./images/lena.jpg')
2 temp = img[200:380, 200:360]
3 cv2.imshow('img', img)
4 cv2.imshow('roi_img', temp)
5 cv2.waitKey(0)
6 cv2.destroyAllWindows()
```



2장. 기본 연산



2절. 채널과 영상

파이썬 OpenCV를 이용한
영상처리

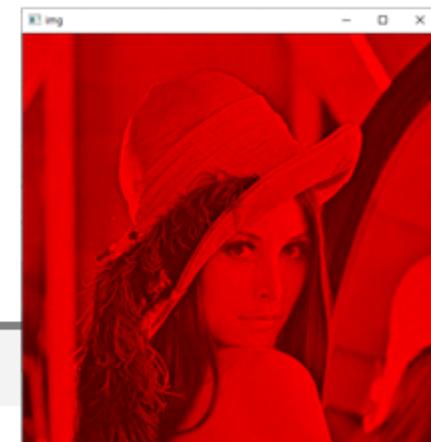
저자 허진경

Color Image는 3개의 채널 B,G,R로 구성이 되어 있습니다. 이것을 각 채널별로 분리할 수 있습니다.

```
1 img = cv2.imread('lena.jpg')
2 b, g, r= cv2.split(img)
3 print(np.mean(b), np.mean(g), np.mean(r), sep=", ")
```

(105.54556274414062, 98.9676513671875, 180.31714248657227)

```
1 img = cv2.imread('lena.jpg')
2 img[:, :, [0,1]] = 0
3
4 cv2.imshow('img', img)
5 cv2.waitKey(0)
6 cv2.destroyAllWindows()
```



cv2.split() 함수는 비용이 많이 드는 함수입니다. 가능하다면 Numpy indexing 방법을 사용하는 것이 더 효율적입니다.

이미지를 더하는 방법은 OpenCV의 cv2.add() 함수를 사용하는 방법과

Numpy 연산, 즉 $\text{img1} + \text{img2}$ 로 하는 방법이 있습니다.

OpenCV의 cv2.add() 는 Saturation 연산을 하고, Numpy는 Modulo 연산을 합니다.



- ▶ Saturation연산은 한계값을 정하고 그 값을 벗어나는 경우는 모두 특정 값으로 계산하는 방식입니다. 이미지에서는 0 이하는 모두 0, 255 이상은 모두 255로 표현하는 것입니다.
- ▶ Modulo연산은 ‘a와 b는 n으로 나눈 나머지 값이 같다’라는 의미입니다. 시계를 예로 들면 2와 14는 12로 나눈 나머지가 2로 동일합니다. 이미지에서는 연산의 결과가 256보다 큰 경우는 256으로 나눈 나머지 값으로 결정을 합니다.

02

OpenCV와 Numpy의 결과

Original 1



Original 2



OpenCV Add
(Saturation)



Numpy Add
(Modulo)



이미지를 서로 합칠 때 가중치를 두어 합치는 방법입니다.

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x) \quad \bullet \quad \alpha \text{ 값이 } 0 \rightarrow 1 \text{로 변함에 따라서 이미지가 전환된다.}$$

```
import cv2
import numpy as np

def weighted_blending(img1, img2, img1_weight=0.5):
    output = np.zeros(img1.shape, dtype=np.uint8)
    height, width = img1.shape[0:2]
    if len(img1.shape)==2:
        for y in range(height):
            for x in range(width):
                output[y,x] = int(img1[y,x]*img1_weight +
                                  img2[y,x]*(1-img1_weight))
    elif len(img1.shape)==3:
        for y in range(height):
            for x in range(width):
                blended = img1[y,x]*img1_weight + img2[y,x]*(1-img1_weight)
                output[y,x] = blended.astype(np.uint8)
    return output
```



```

import cv2
import numpy as np

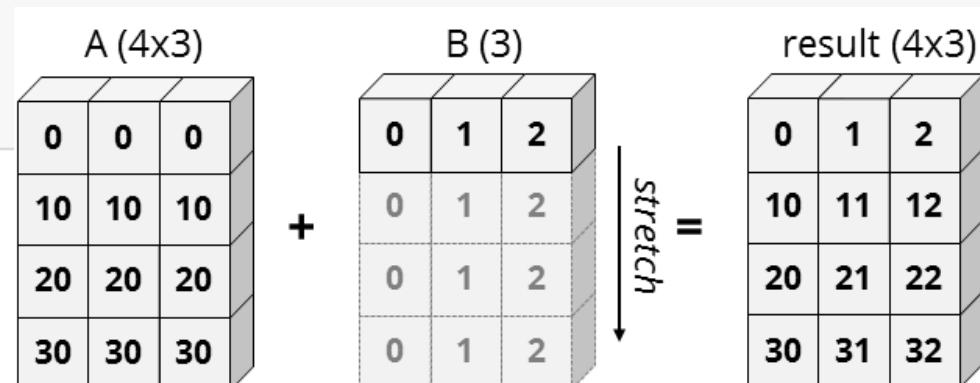
def ez_blending(img1, img2, img1_weight=0.5):
    output = img1*img1_weight + img2*(1-img1_weight)
    return output.astype(np.uint8)

```

```

new_img = ez_blending(img1, img2, img1_weight=0.3)
cv2.imshow('img', new_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



브로드캐스팅 규칙

화소의 2진수 값을 이용해 연산

특정 영역을 없애는 마스크 연산에 주로 사용함

p1	p2	bitwise_and (p1, p2)	bitwise_or (p1, p2)	bitwise_not (p1)	bitwise_택 (p1, p2)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

```

#-*- coding:utf-8 -*-
import cv2
import numpy as np

img1 = cv2.imread('logo.png')
img2 = cv2.imread('lena.jpg')

rows, cols, channels = img1.shape # 삽입할 이미지의 row, col, channel 정보
roi = img2[0:rows, 0:cols] # 대상 이미지에서 삽입할 이미지의 영역을 추출

#mask를 만들기 위해서 img1을 gray로 변경후 binary image로 전환
#mask는 logo부분이 흰색(255), 바탕은 검은색(0)
#mask_inv는 logo부분이 검은색(0), 바탕은 흰색(255)
img1gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img1gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)

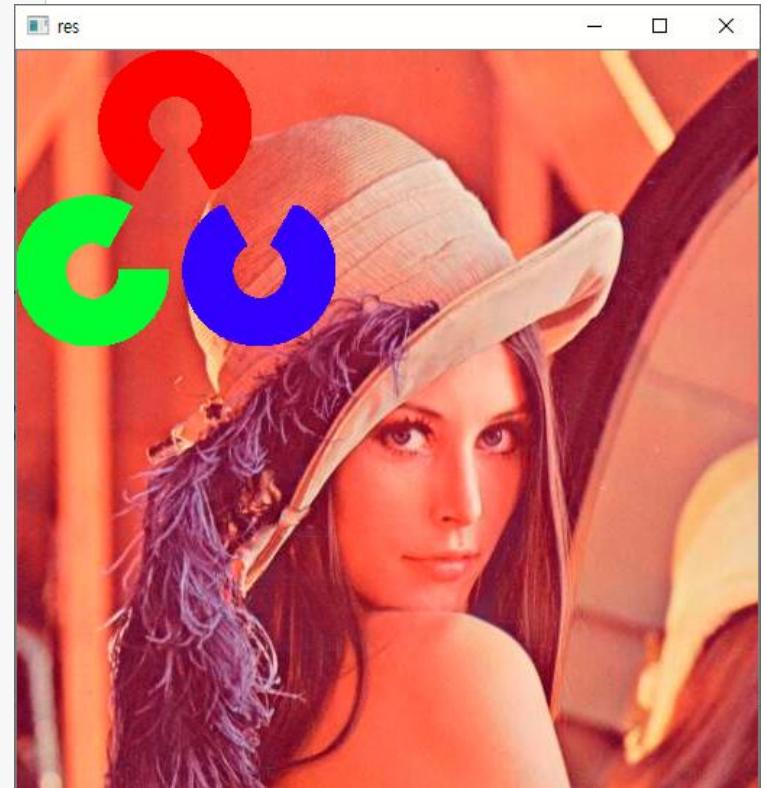
#bitwise_and 연산자는 둘다 0이 아닌 경우만 값을 통과 시킴.
#즉 mask가 검정색이 아닌 경우만 통과가 되기 때문에 mask영역 이외는 모두 제거됨.
#아래 img1_fg의 경우는 bg가 제거되고 fg(logo부분)만 남게 됨.
#img2_bg는 roi영역에서 logo부분이 제거되고 bg만 남게 됨.
img1_fg = cv2.bitwise_and(img1, img1, mask=mask)
img2_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)

#2개의 이미지를 합치면 바탕은 제거되고 logo부분만 합쳐짐.
dst = cv2.add(img1_fg, img2_bg)

img2[0:rows, 0:cols] = dst #합쳐진 이미지를 원본 이미지에 추가.

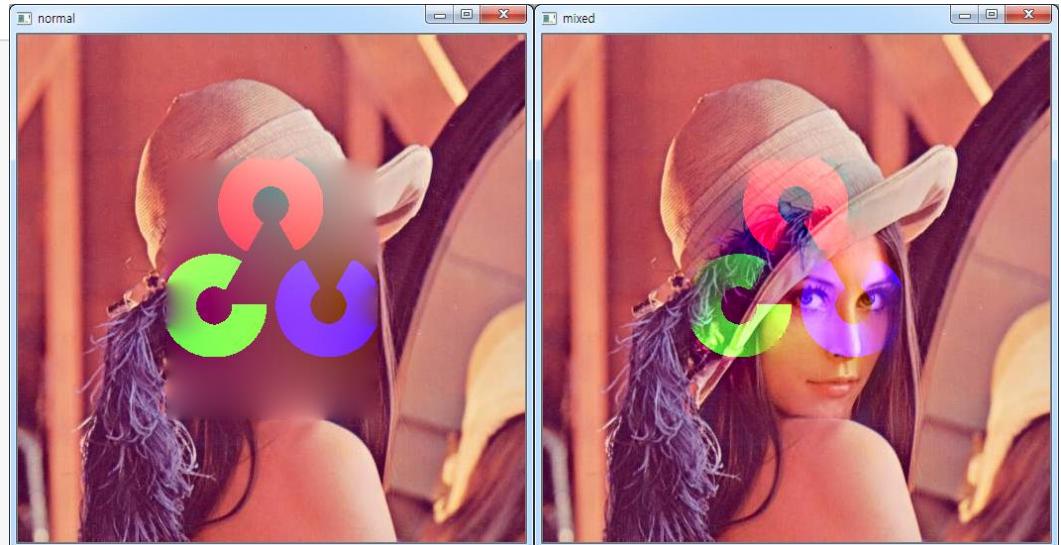
cv2.imshow('res', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



 cv2.seamlessClone (src, dst, mask, point, flags)

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 import numpy as np
4
5 img1 = cv2.imread('logo.png')
6 img2 = cv2.imread('lena.jpg')
7
8 mask = np.full_like(img1, 255)
9 height, width = img2.shape[:2]
10 center = (width//2, height//2)
11
12 normal = cv2.seamlessClone(img1, img2, mask, center, cv2.NORMAL_CLONE)
13 mixed = cv2.seamlessClone(img1, img2, mask, center, cv2.MIXED_CLONE)
14
15 cv2.imshow('normal', normal)
16 cv2.imshow('mixed', mixed)
17 cv2.waitKey(0)
18 cv2.destroyAllWindows()
```



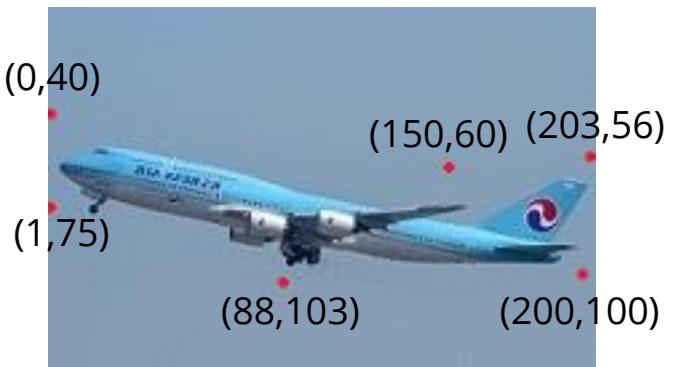
02

cv2.seamlessClone()



?

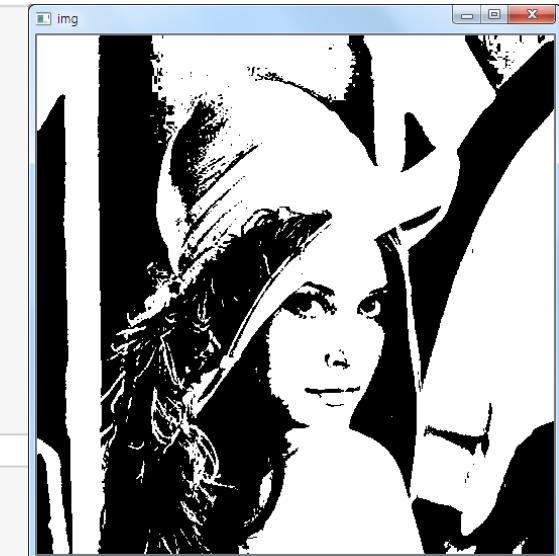
```
1 import cv2
2 import numpy as np
3
4 src = cv2.imread("airplane2.jpg")
5 dst = cv2.imread("red_sky.jpg")
6 src_mask = np.zeros(src.shape, src.dtype)
7 poly = np.array([ [0,40], [150,60], [203,56], [200,100],
                   [88,103], [1,75] ], np.int32)
8 cv2.fillPoly(src_mask, [poly], (255, 255, 255))
9 center = (600,150)
10 output = cv2.seamlessClone(src, dst, src_mask, center,
                               cv2.NORMAL_CLONE)
11
12 cv2.imshow('Output', output)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```



배열의 브로드캐스팅 규칙과 True/False가 1과 0으로 연산되는 것을 이용

```
1 import cv2
2 import numpy as np
3
4 def two_tone(img, threshold=128):
5     output = (img > threshold) * 255
6     return output.astype(np.uint8)
```

```
1 img = cv2.imread("./images/lena.jpg", 0)
2 new_img = two_tone(img, threshold=120)
3 cv2.imwrite('./images/lena_bin.jpg', new_img)
4 cv2.imshow('img', new_img)
5 cv2.waitKey(0)
6 cv2.destroyAllWindows()
```



cv2.threshold(src, thresh, maxValue, thresholdType) → retval, dst

- ▶ src – 단일 채널 입력 이미지(그레이스케일 이미지)
thresh – 임계값
maxValue – 임계값을 넘었을 때 적용할 값
thresholdType – 임계치를 적용할 타입(Thresholding Type)
- ▶ Thresholding Type은 아래와 같습니다.

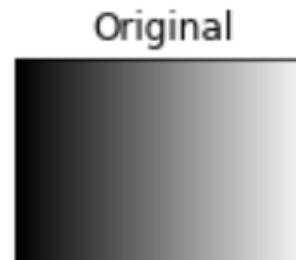
cv2.THRESH_BINARY

cv2.THRESH_BINARY_INV

cv2.THRESH_TRUNC

cv2.THRESH_TOZERO

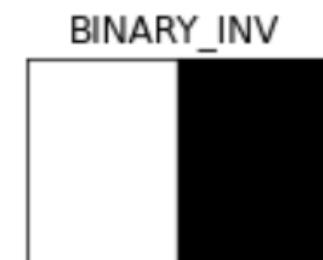
cv2.THRESH_TOZERO_INV



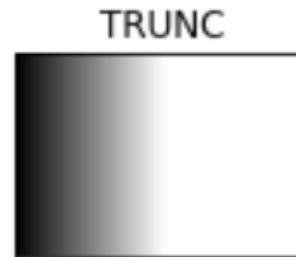
Original



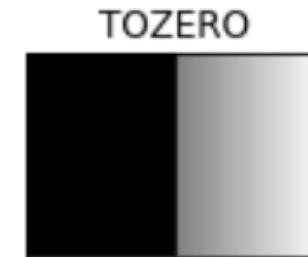
BINARY



BINARY_INV



TRUNC

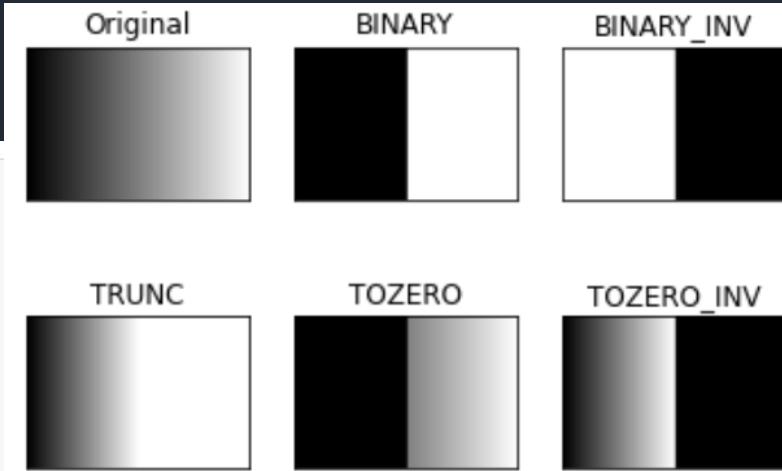


TOZERO



TOZERO_INV

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('gradation.png',0)
6
7 ret, thresh1 = cv2.threshold(img,127,255, cv2.THRESH_BINARY)
8 ret, thresh2 = cv2.threshold(img,127,255, cv2.THRESH_BINARY_INV)
9 ret, thresh3 = cv2.threshold(img,127,255, cv2.THRESH_TRUNC)
10 ret, thresh4 = cv2.threshold(img,127,255, cv2.THRESH_TOZERO)
11 ret, thresh5 = cv2.threshold(img,127,255, cv2.THRESH_TOZERO_INV)
12
13 titles = ['Original','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
14 images = [img,thresh1,thresh2,thresh3,thresh4,thresh5]
15
16 for i in range(6):
17     plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
18     plt.title(titles[i])
19     plt.xticks([])
20     plt.yticks([])
21
22 plt.show()
```





cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

- ▶ src – 단일 채널 입력 이미지(그레이스케일 이미지)
maxValue – 임계값을 넘었을 때 적용할 값
adaptiveMethod – 임계값을 결정하는 계산 방법(Adaptive Method)
thresholdType – 임계치를 적용할 타입(Thresholding Type)
blockSize – 임계치를 적용할 영역의 크기(가로/세로 화소 수)
C – 평균이나 가중평균에서 차감할 값
- ▶ Adaptive Method는 아래와 같습니다.
`cv2.ADAPTIVE_THRESH_MEAN_C` : blockSize * blockSize 안에 있는 화소
값의 평균에서 C값을 뺀 값을 임계치로 합니다.
`cv2.ADAPTIVE_THRESH_GAUSSIAN_C` : blockSize * blockSize 안에 있는
가우시안 윈도우 기반 가중치들의 합에서 C값을 뺀 값을 임계치로 합니다.

임계값을 이미지 전체에 적용하여 처리하기 때문에 하나의 이미지에 음영이 다르면 일부 영역이 모두 흰색 또는 검정색으로 보여지게 됩니다.

이런 문제를 해결하기 위해서 이미지의 작은 영역별로 임계치를 다르게 적용하는 것입니다. 이때 사용하는 함수가 `cv2.adaptiveThreshold()`입니다.

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img_gray = cv2.imread('house.jpg', 0)
6
7 _, img_bin = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
8 adt_mean = cv2.adaptiveThreshold(img_gray, 255,
9                                 cv2.ADAPTIVE_THRESH_MEAN_C,
10                                cv2.THRESH_BINARY, 15, 2)
11 adt_gaus = cv2.adaptiveThreshold(img_gray, 255,
12                                 cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
13                                cv2.THRESH_BINARY, 15, 2)
14
15 titles = ['Gray', 'Global', 'Mean', 'Gaussian']
16 images = [img_gray, img_bin, adt_mean, adt_gaus]
17
18 plt.figure(figsize=(8,6))
19 for i in range(4):
20     plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
21     plt.title(titles[i])
22     plt.xticks([]),plt.yticks([])
23
24 plt.show()
```



Gray



Global



Mean



Gaussian

제한된 색을 이용하여 음영이나 색을 나타내는 기법

플로이드-스테인버그 디더링(Floyd-Steinberg dithering) 기법, 평균 디더링(Average dithering), 순서 디더링(Ordered dithering), 임의 디더링(Random dithering) 등이 있음

참고 : <https://en.wikipedia.org/wiki/Dither>

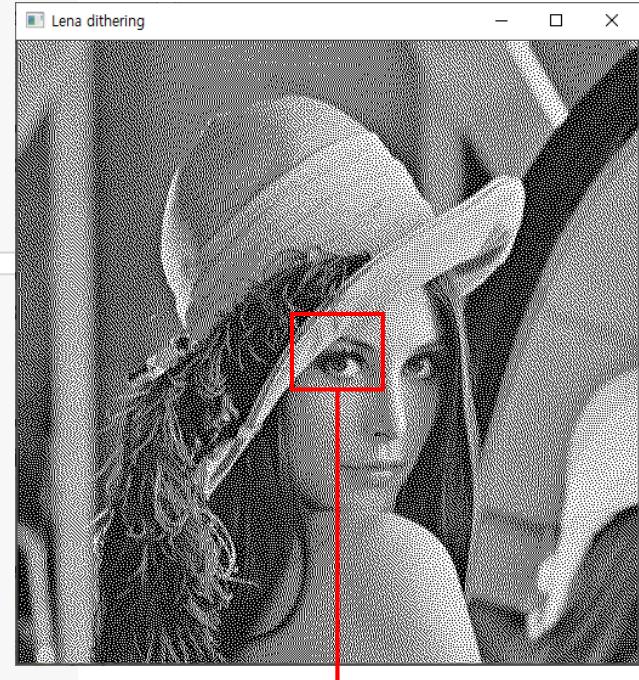
플로이드-스테인버그 디더링 : 영상 편집 소프트웨어에서 널리 쓰이는 알고리듬

화소의 양자화 오류를 주위의 화소로 분산시킴으로써 디더링을 수행함

$$\begin{bmatrix} & & * & \frac{7}{16} & \dots \\ \dots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \dots \end{bmatrix}$$

```
1 def minmax(pixel):
2     if pixel > 255:
3         pixel = 255
4     if pixel < 0:
5         pixel = 0
6     return pixel
```

```
1 import numpy as np
2
3 def dithering(img):
4     height, width = img.shape
5
6     for y in range(0, height-1):
7         for x in range(1, width-1):
8             p = img[y, x]
9             new_p = np.round(p/255) * 255
10            img[y, x] = new_p
11            error = p - new_p
12
13            img[y , x+1] = minmax(img[y , x+1] + error*7/16)
14            img[y+1, x-1] = minmax(img[y+1, x-1] + error*3/16)
15            img[y+1, x ] = minmax(img[y+1, x ] + error*5/16)
16            img[y+1, x+1] = minmax(img[y+1, x+1] + error*1/16)
17
18    return img
```



2장. 기본 연산



3절. 히스토그램

파이썬 OpenCV를 이용한
영상처리

저자 허진경

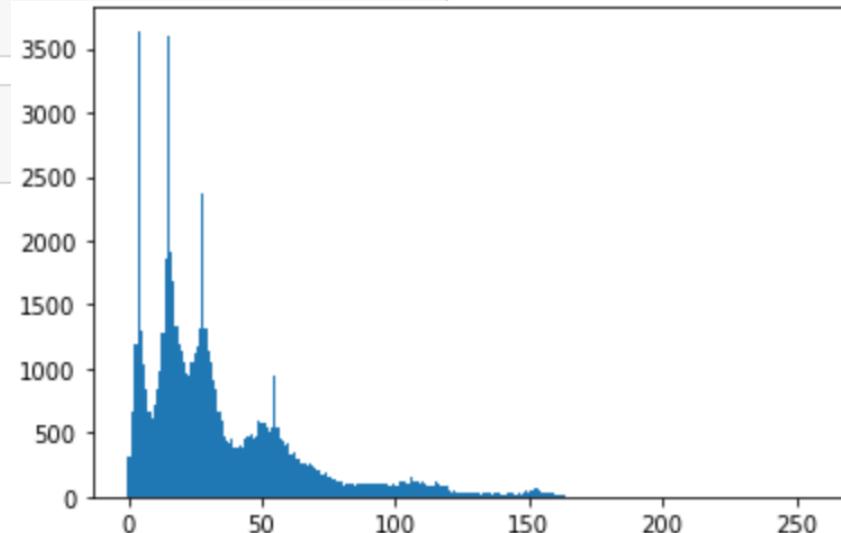
```
img = cv2.imread("couple2.jpg", flags=cv2.IMREAD_GRAYSCALE)
```

```
import numpy as np  
hist = np.zeros((256))
```

```
height, width = img.shape
```

```
for y in range(height):  
    for x in range(width):  
        hist[img[y,x]] = hist[img[y,x]] + 1
```

```
plt.bar(x=range(256), height=hist)
```

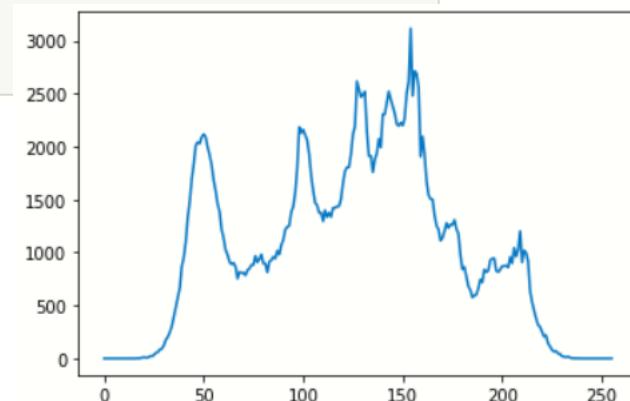




cv2.calcHist(*images, channels, mask, histSize, ranges [, hist [, accumulate]]*)

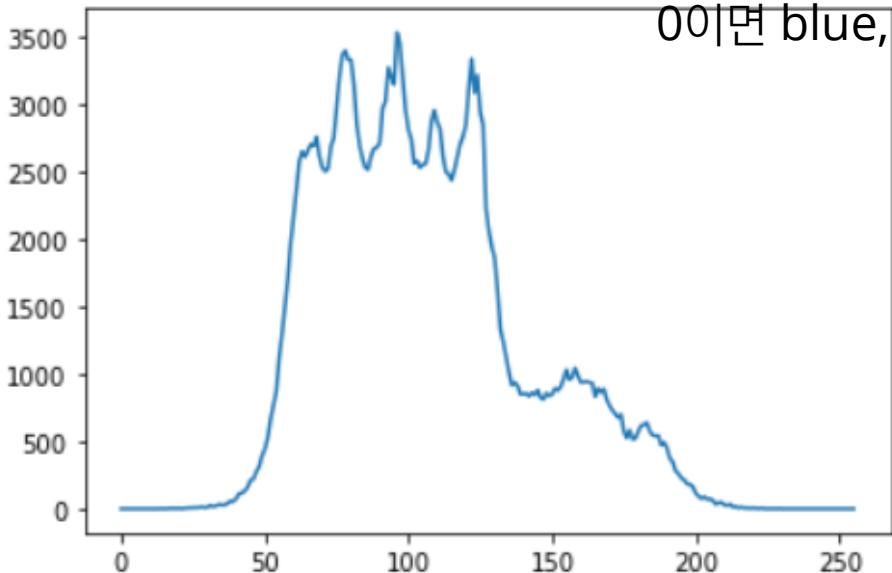
- ▶ image – 분석대상 이미지(uint8 or float32 type). Array 형태.
- ▶ channels – 분석 채널(X축의 대상). 이미지가 grayscale이면 [0], color 이미지이면 [0],[1] 형태(0 : Blue, 1: Green, 2: Red)
- ▶ mask – 이미지의 분석영역. None이면 전체 영역.
- ▶ histSize – BINS 값. [256]
- ▶ ranges – Range값. [0,256]

```
1 lena_gray = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2 hist = cv2.calcHist(images=[lena_gray], channels=[0], mask=None,
3                      histSize=[256], ranges=[0,256])
4 plt.plot(hist.flatten())
5 plt.show()
```

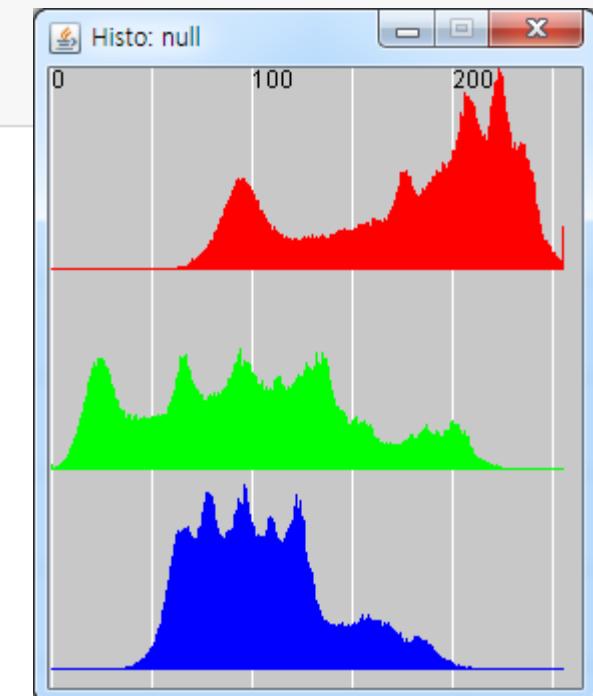


채널(channels)은 한 개 화소가 가지고 있는 정보

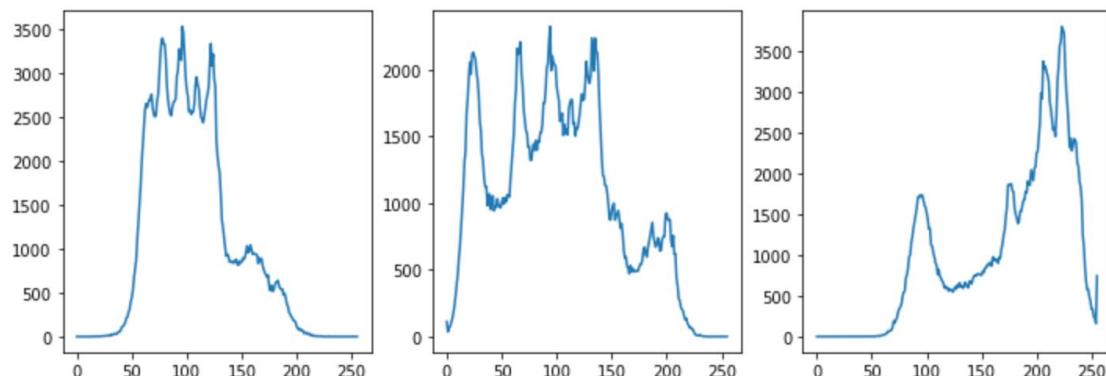
```
1 lena = cv2.imread("lena.jpg")
2 hist = cv2.calcHist(images=[lena], channels=[0], mask=None,
3                      histSize=[256], ranges=[0,256])
4 plt.plot(hist.flatten())
5 plt.show()
```



컬러영상은 채널이 0,1,2
0이면 blue, 1이면 green, 2이면 red



```
1 lena = cv2.imread("lena.jpg")
2 plt.figure(figsize=(12,4))
3
4 for i in range(3):
5     plt.subplot(1,3,i+1)
6     hist = cv2.calcHist(images=[lena],
7                          channels=[i], mask=None,
8                          histSize=[256], ranges=[0,256])
9     plt.plot(hist.flatten())
```





3장. 이미지 변환

영상 변환에 대해 설명합니다.

- 1절. 기하 변환
- 2절. 크기 변환
- 3절. 회전
- 4절. Affine 변환

3장. 이미지 변환



1절. 기하 변환

파이썬 OpenCV를 이용한
영상처리

저자 허진경



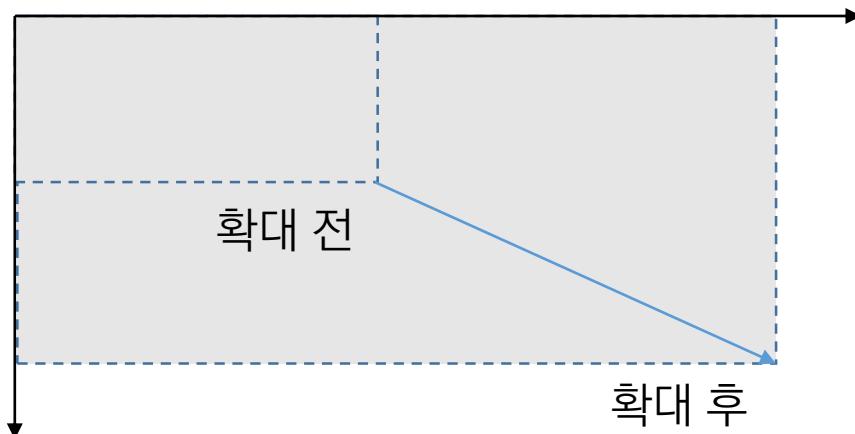
좌표 x 를 좌표 x' 로 변환하는 함수



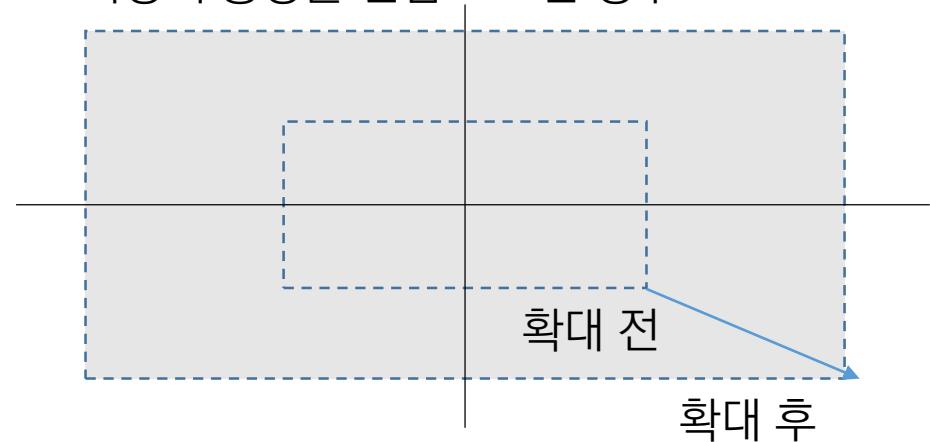
변환의 종류에는 몇가지 분류가 있습니다.

- ▶ 강체변환(Rigid-Body) : 크기 및 각도가 보존(ex; Translation, Rotation)
- ▶ 유사변환(Similarity) : 크기는 변하고 각도는 보존(ex; Scaling)
- ▶ 선형변환(Linear) : Vector 공간에서의 이동. 이동변환은 제외.
- ▶ Affine : 선형변환과 이동변환까지 포함. 선의 수평성은 유지.(ex;사각형→평행사변형)
- ▶ Perspective : Affine변환에 수평성도 유지되지 않음. 원근변환

화상의 좌측 위를 원점으로 한 경우



화상의 중앙을 원점으로 한 경우



3장. 이미지 변환



2절. 크기 변환

파이썬 OpenCV를 이용한
영상처리

저자 허진경

어떤 점 (x, y) 가 확대 또는 축소되어 (X, Y) 로 위치를 바꾼다고 하면...

$$X = a * x$$

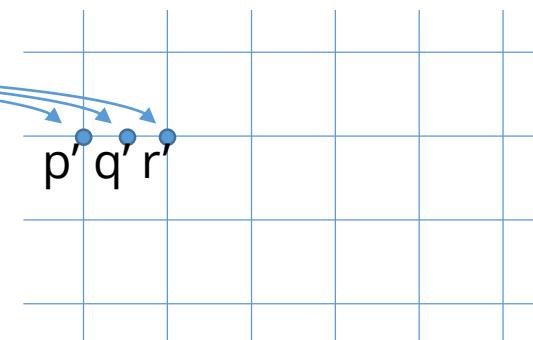
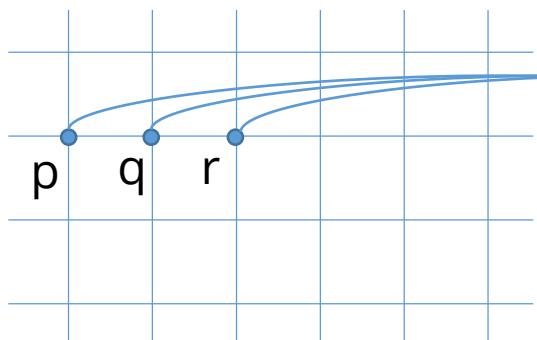
$$x = X/a$$

$$Y = b * y$$

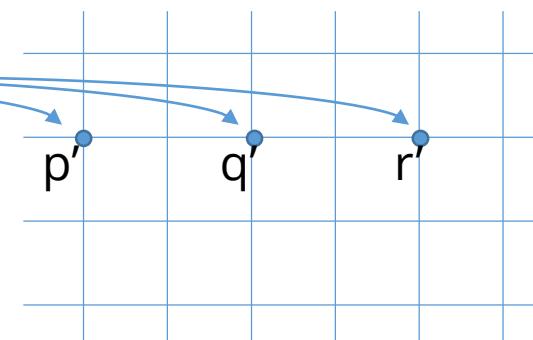
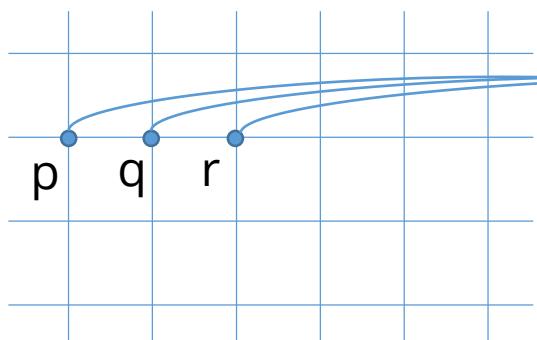
$$y = Y/b$$

입력 화상

출력 화상



1/2 축소



2배 확대

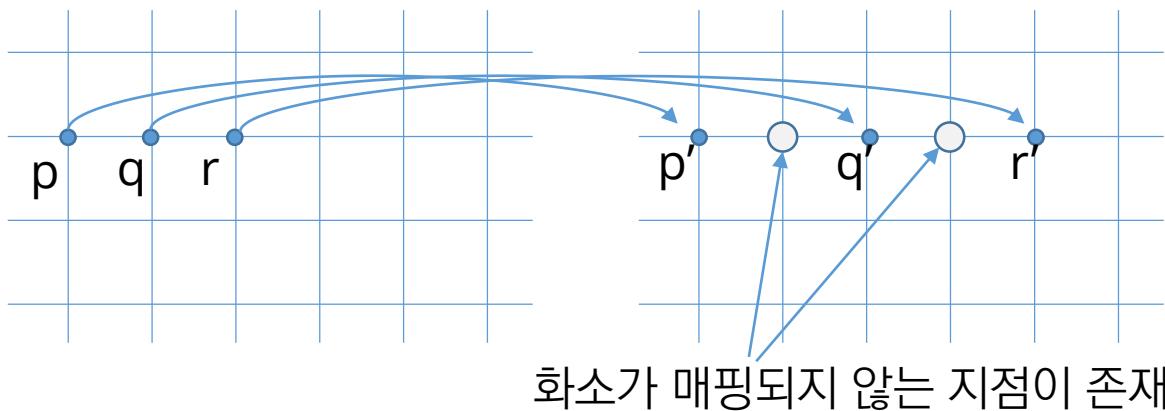
```
1 def scale_nogood(img, scale_x=1, scale_y=1):
2     height, width = img.shape
3     img_ = np.zeros((int(height*scale_y), int(width*scale_x)),
4                      dtype=np.uint8)
5     for y in range(height):
6         for x in range(width):
7             try:
8                 img_[int(y*scale_y), int(x*scale_x)] = img[y, x]
9             except:
10                pass
11 return img_
```

```
1 img = cv2.imread("lena.jpg", cv2.IMREAD_GRAYSCALE)
2 img_pro(scale_nogood, img, 0.5, 0.8)
```

배율을 1보다 큰 수로 했을 경우에는?



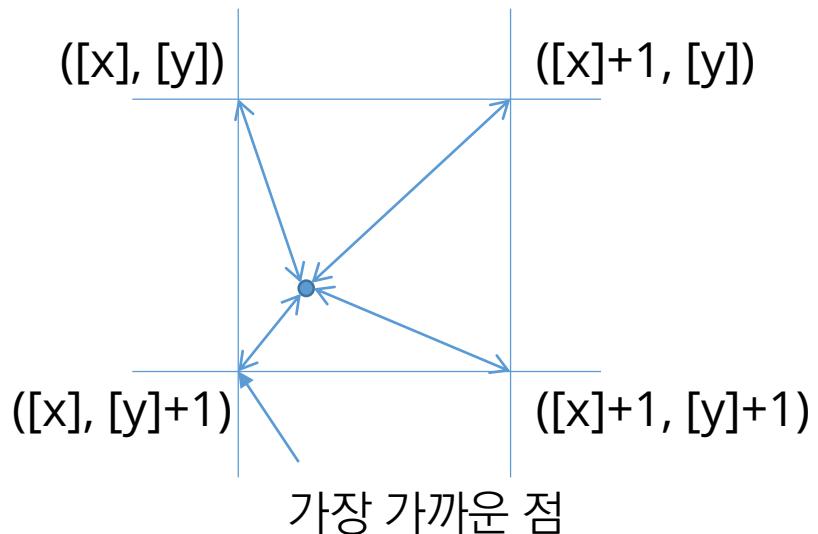
2배 확대 시



크기 변환 시 화소가 매핑되지 않는 값을 지정하는 방법

최근방법

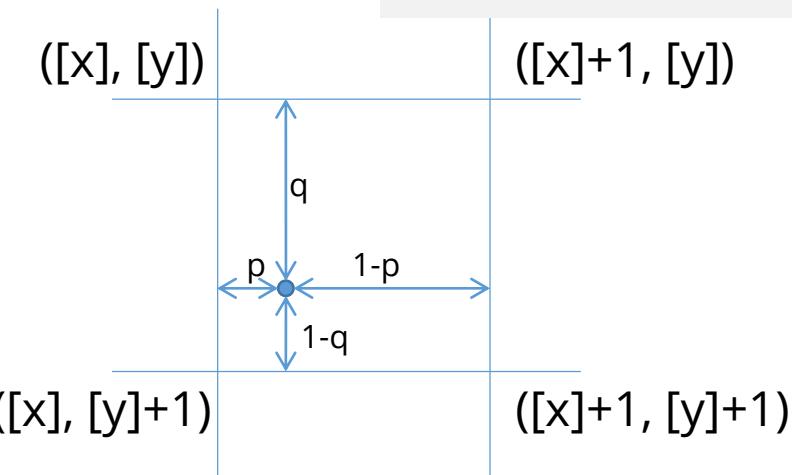
가장 가까운 좌표의 값을
이용하여 지정하는 방법



$[x], [y]$ 는 각각 x, y 를 넘지 않는 정수

선형방법

원본 영상에서 인접한 네 개의
픽셀 값을 이용하여 실수 좌표
상의 픽셀 값을 계산하는 방법.

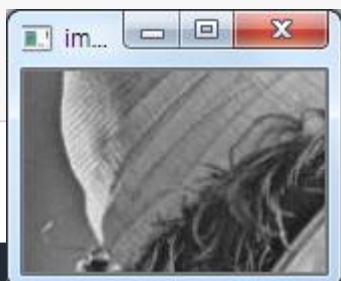


$$d(x, y) = (1 - p) * (1 - q) * d([x], [y]) + \\ p * (1 - q) * d([x] + 1, [y]) + \\ (1 - p) * q * d([x], [y] + 1) + \\ p * q * d([x] + 1, [y] + 1)$$

```
1 def scale_nearest(img, scale_x=1, scale_y=1):
2     height, width = img.shape
3     img_ = np.zeros((int(height*scale_y),int(width*scale_x)),
4                      dtype=np.uint8)
5     for y in range(height*scale_y):
6         for x in range(width*scale_x):
7             try :
8                 img_[y,x] = img[int(y/scale_y), int(x/scale_x)]
9             except:
10                 pass
11
12 return img_
```



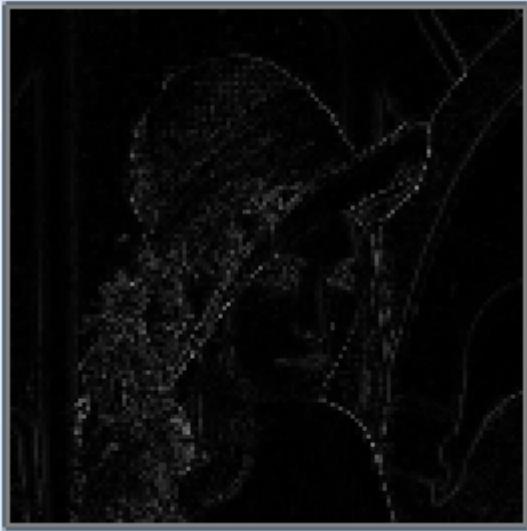
```
1 def scale_bilinear(img, scale_x=1, scale_y=1):
2     height, width = img.shape
3     img_ = np.zeros((int(height*scale_y), int(width*scale_x)), dtype=np.uint8)
4     for y in range(int(height*scale_y)):
5         for x in range(int(width*scale_x)):
6             p = x/scale_x-int(x/scale_x)
7             q = y/scale_y-int(y/scale_y)
8             try:
9                 X = int(x/scale_x)
10                Y = int(y/scale_y)
11                value = (1-p)*(1-q)*img[Y, X] + p*(1-q)*img[Y+1, X] \
12                    + (1-p)*q*img[Y, X+1] + p*q*img[Y+1, X+1]
13                if value > 255:
14                    img_[y, x] = 255
15                else:
16                    img_[y, x] = int(value)
17            except:
18                pass
19    return img_
```



03 scale_nearest(0.5) vs. maxpool2d



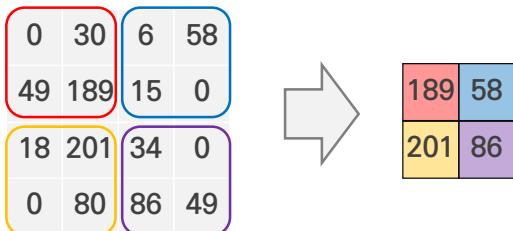
scale_nearest(0.5)



maxpool2d



2x2 Max Pooling

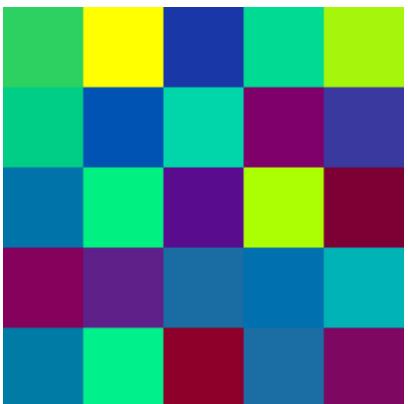
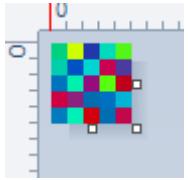


```
1 def maxpool2d(img):
2     height, width = img.shape
3     img_ = np.zeros((int(height/2),int(width/2)),
4                      dtype=np.uint8)
5     for y in range(int(height/2)):
6         for x in range(int(width/2)):
7             try :
8                 img_[y,x] = np.max(img[2*y:2*y+2, 2*x:2*x+2])
9             except:
10                 pass
11     return img_
```

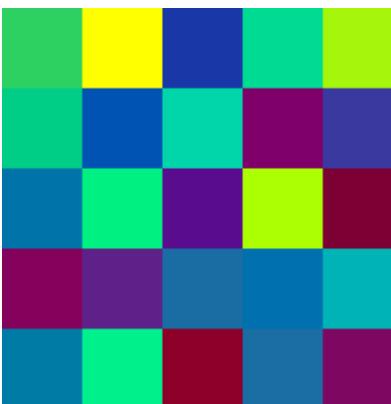
 cv2.resize(*img, dsize, fx, fy, interpolation*)

- ▶ *img* – 크기를 변경할 원본 이미지 객체입니다.
dsize – Manual Size. 가로, 세로 형태의 tuple(ex: (100,200))
fx – 가로 사이즈의 배수. 2배로 크게하려면 2. 반으로 줄이려면 0.5
 – 세로 사이즈의 배수
interpolation – 보간법
 - INTER_NEAREST : 가장 가까운 이웃 보간법
 - INTER_LINEAR : 쌍 선형 보간법(기본적으로 사용됨)
 - INTER_LINEAR_EXACT : 비트 쌍 선형 보간법
 - INTER_AREA : 영역 보간법(이미지를 줄일 때 주로 사용, 이미지를 확대하면 INTER_NEAREST 방법과 유사)
 - INTER_CUBIC : 4x4픽셀 이웃에 대한 쌍 입방 보간법
 - INTER_LANCZOS4 : 8x8픽셀 이웃에 대한 랜조스(Lanczos) 보간법

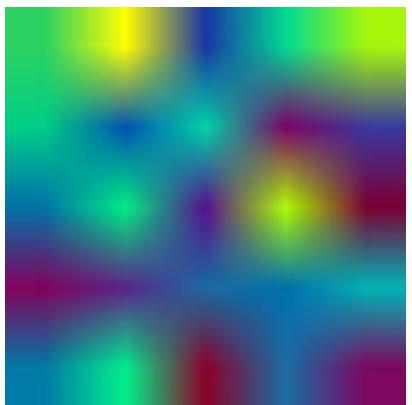
- ▶ 많이 사용되는 보간법은 사이즈를 줄일 때는 cv2.INTER_AREA , 사이즈를 크게할 때는 cv2.INTER_CUBIC , cv2.INTER_LINEAR 을 사용합니다.



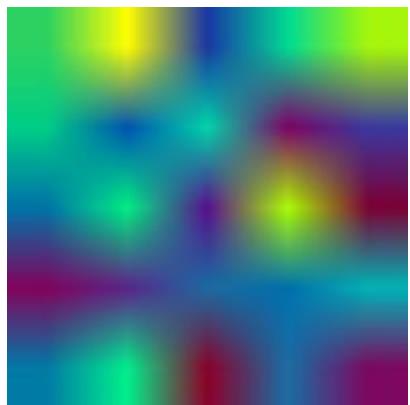
INTER_AREA



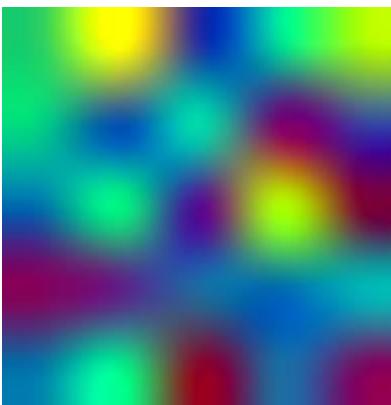
INTER_NEAREST



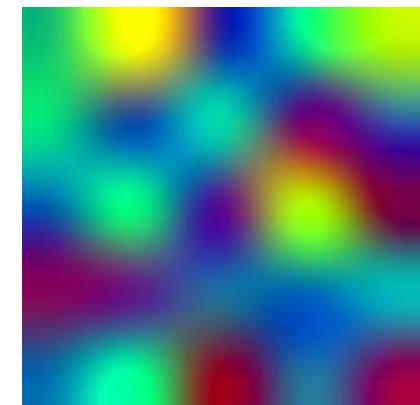
INTER_LINEAR



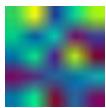
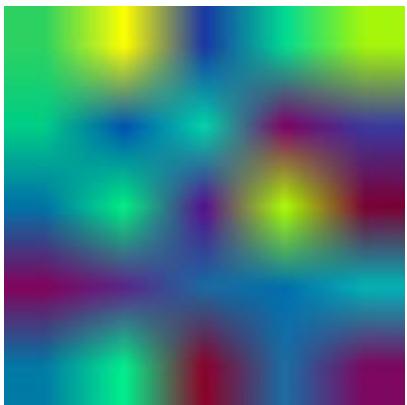
INTER_LINEAR_EXACT



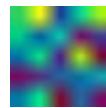
INTER_CUBIC



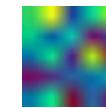
INTER_LANCZOS4



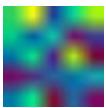
INTER_AREA



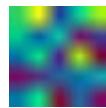
INTER_CUBIC



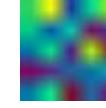
INTER_LANCZOS



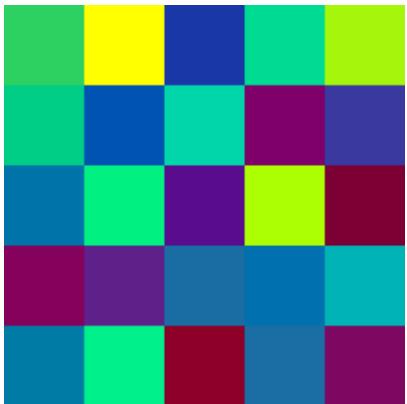
INTER_LINEAR



INTER_LINEAR_EXACT



INTER_NEAREST



INTER_AREA



INTER_CUBIC



INTER_LANCZOS



INTER_LINEAR



INTER_LINEAR_EXACT



INTER_NEAREST

3장. 이미지 변환

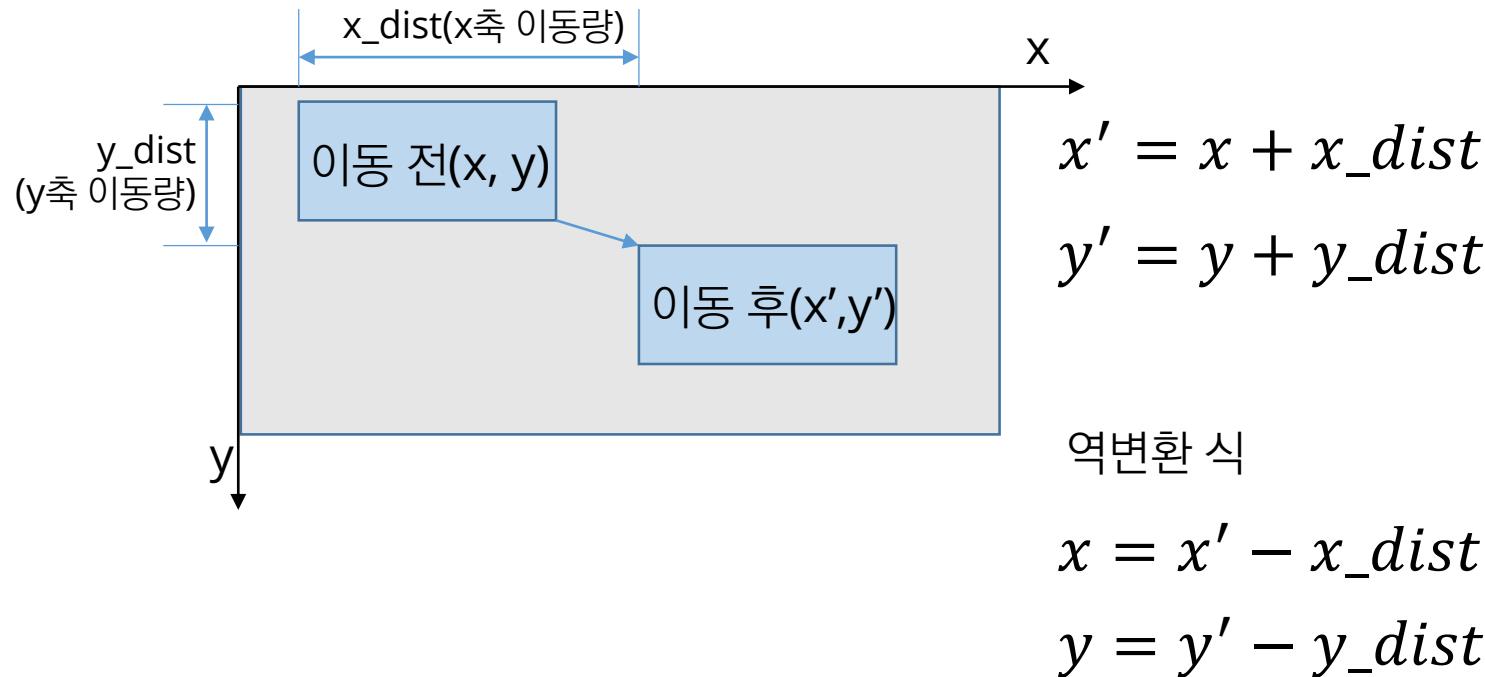


3절. 이동

파이썬 OpenCV를 이용한
영상처리

저자 허진경

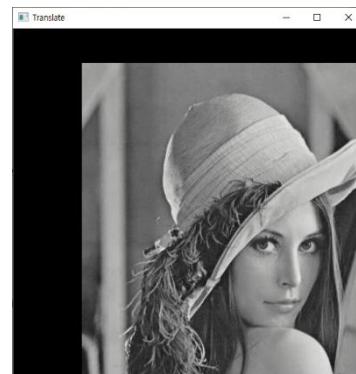
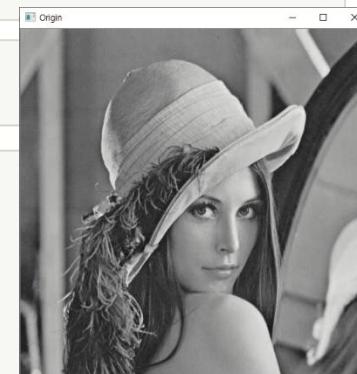
이미지를 특정 위치를 수평 또는 수직 방향으로 변경하는 것



```
1 import numpy as np
2
3 def translate(img, dist=(0,0)):
4     height, width = img.shape
5     img_ = np.zeros(img.shape, dtype=np.uint8)
6
7     for y in range(height):
8         for x in range(width):
9             X = x + dist[0]
10            Y = y + dist[1]
11            if (X < 0) | (X >= width) | (Y < 0) | (Y >= height):
12                continue
13            img_[Y, X] = img[y, x]
14
15    return img_
```

```
1 img = cv2.imread("./images/lena.jpg")
```

```
1 cv2.imshow("Origin", img)
2 cv2.imshow("Translate", translate(img, (100, 50)))
3 cv2.waitKey()
4 cv2.destroyAllWindows()
```



3장. 이미지 변환

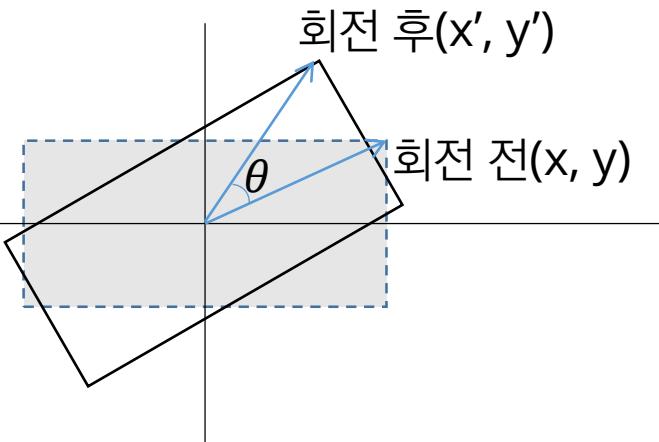


4절. 회전

파이썬 OpenCV를 이용한
영상처리

저자 허진경

이미지를 특정 위치를 기준으로 반시계방향으로 회전



$$x' = x * \cos\theta - y * \sin\theta$$

$$y' = x * \sin\theta + y * \cos\theta$$

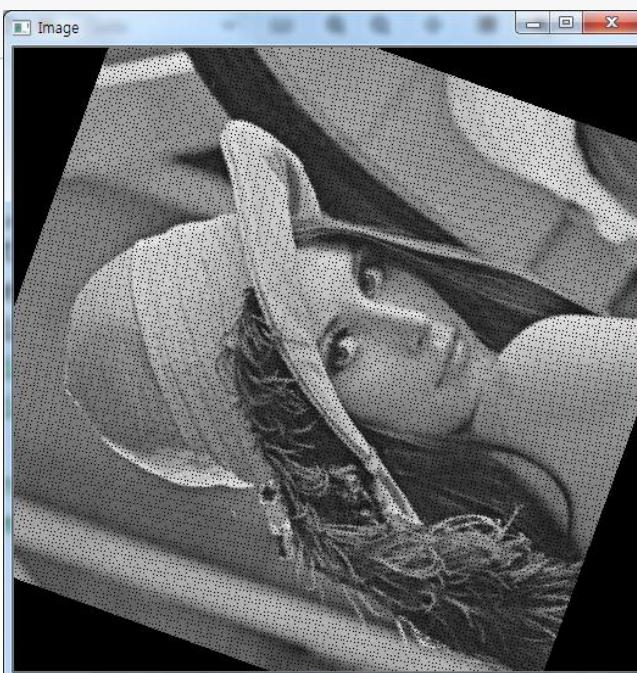
역변환 식

$$x = x' * \cos\theta + y' * \sin\theta$$

$$y = -x' * \sin\theta + y' * \cos\theta$$

```
1 import numpy as np
2
3
4 def rotate(img, deg=30):
5     height, width = img.shape
6     img_ = np.zeros(img.shape, dtype=np.uint8)
7     seta = np.pi / (180.0 / deg)
8     center_x = int(width / 2)
9     center_y = int(height / 2)
10
11    for y in range(height):
12        for x in range(width):
13            X = int((x-center_x)*np.cos(seta) +
14                     (y-center_y)*np.sin(seta)) + center_x
15            Y = int(-(x-center_x)*np.sin(seta) +
16                     (y-center_y)*np.cos(seta)) + center_y
17            if (X < 0) | (X >= width) | (Y < 0) | (Y >= height):
18                continue
19            img_[Y, X] = img[y, x]
20
21    return img_
```

```
21  for y in range(height-1):
22      for x in range(width-1):
23          if img_[y, x] == 0:
24              outer = [img_[y-1,x-1], img_[y-1,x], img_[y-1,x+1],
25                      img_[y ,x-1], img_[y ,x+1],
26                      img_[y+1,x-1], img_[y+1,x], img_[y+1,x+1]]
27              img_[y, x] = np.uint8(np.mean(outer))
28
return img_
```



3장. 이미지 변환



5절. Affine 변환

파이썬 OpenCV를 이용한
영상처리

저자 허진경

Affine Transformation은 선의 평행성은 유지가 되면서 이미지를 변환하는 작업입니다

이동, 확대, Scale, 반전까지 포함된 변환입니다.

Affine 변환을 위해서는 3개의 Match가 되는 점이 있으면 변환행렬을 구할 수 있습니다.

$$X = (x - x_0) * \cos\theta + (y - y_0) * \sin\theta - x_0$$

$$Y = -(x - x_0) * \sin\theta + (y - y_0) * \cos\theta + y_0$$

 cv2.getRotationMatrix2D(*center, angle, scale*) → M

- ▶ center – 이미지의 중심 좌표
- ▶ angle – 회전 각도
- ▶ scale – 크기 배율, 0.5이면 1/2크기, 1이면 원래 이미지 크기

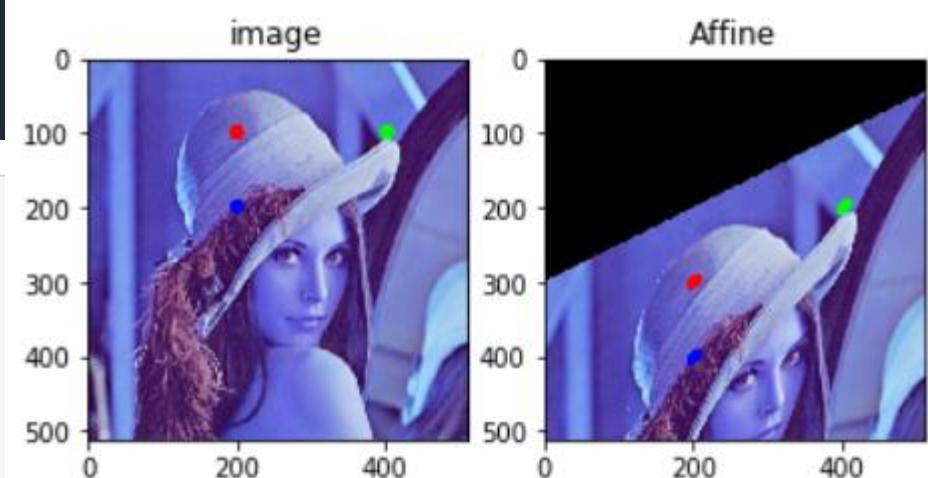
```
rows, cols = img.shape[:2]
```

```
M= cv2.getRotationMatrix2D((cols/2, rows/2), 90, 0.5)
```

 cv2.warpAffine(*src*, *M*, *dsize*)

- ▶ *src* – Image
 - ▶ *M* – 변환 행렬
 - ▶ *dsize* (tuple) – output image size(ex; (width=columns, height=rows))
-
- ▶ Affine 변환입니다.

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('lena.jpg')
6 rows, cols, ch = img.shape
7
8 pts1 = np.float32([[200,100],[400,100],[200,200]])
9 pts2 = np.float32([[200,300],[400,200],[200,400]])
10
11 # pts1의 좌표에 표시. Affine 변환 후 이동 점 확인.
12 cv2.circle(img, (200,100), 10, (255,0,0), -1)
13 cv2.circle(img, (400,100), 10, (0,255,0), -1)
14 cv2.circle(img, (200,200), 10, (0,0,255), -1)
15
16 M = cv2.getAffineTransform(pts1, pts2)
17
18 dst = cv2.warpAffine(img, M, (cols,rows))
19
20 plt.subplot(121),plt.imshow(img),plt.title('image')
21 plt.subplot(122),plt.imshow(dst),plt.title('Affine')
22 plt.show()
```



3장. 이미지 변환



6절. Perspective 변환

파이썬 OpenCV를 이용한
영상처리

저자 허진경

원근법 변환은 직선의 성질만 유지가 되고, 선의 평행성은 유지가 되지 않는 변환입니다.

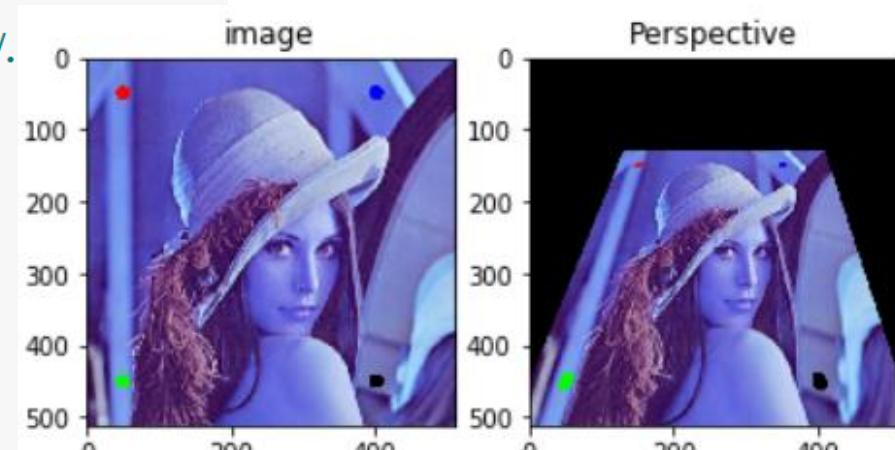
cv2.getPerspectiveTransform(src, dst) : 변환 행렬을 구하기 위해서 사용함

- ▶ src – 소스 이미지에서 사각형 꼭지점의 좌표입니다.
dst – 대상 이미지에서 해당 사각형 꼭지점의 좌표입니다.

cv.warpPerspective(src, dst, mapMatrix, flags, fillval=(0, 0, 0, 0))

- ▶ src – 입력 이미지.
dst – 크기가 dsize이고 src와 동일한 유형의 이미지를 출력합니다.
M – 3×3 변환 행렬.
dsize – 출력 이미지의 크기입니다.
flags – 보간 방법 (INTER_LINEAR 또는 INTER_NEAREST)과 선택적 플래그 WARP_INVERSE_MAP의 조합으로 M을 역변환으로 설정합니다. 기본값은 CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS
borderMode – 픽셀 외삽 방법 (BORDER_CONSTANT 또는 BORDER_REPLICATE).
borderValue – 경계가 일정한 경우에 사용되는 값. 기본적으로 0과 같습니다.

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 import numpy as np
4 from matplotlib import pyplot as plt
5
6 img = cv2.imread('lena.jpg')
7 # [x,y] 좌표점을 4x2의 행렬로 작성
8 # 좌표점은 좌상->좌하->우상->우하
9 pts1 = np.float32([[50,50],[50,450],[400,50],[400,450]])
10
11 # 좌표의 이동점
12 pts2 = np.float32([[150,150],[50,450],[350,150],[400,450]])
13
14 # pts1의 좌표에 표시. perspective 변환 후 이동 점 확인.
15 cv2.circle(img, (50,50), 10, (255,0,0), -1)
16 cv2.circle(img, (50,450), 10, (0,255,0), -1)
17 cv2.circle(img, (400,50), 10, (0,0,255), -1)
18 cv2.circle(img, (400,450), 10, (0,0,0), -1)
19
20 M = cv2.getPerspectiveTransform(pts1, pts2)
21 dst = cv2.warpPerspective(img, M, (512,512))
22
23 plt.subplot(121),plt.imshow(img),plt.title('image')
24 plt.subplot(122),plt.imshow(dst),plt.title('Perspective')
25 plt.show()
```



3장. 이미지 변환



7절. 푸리에 변환

파이썬 OpenCV를 이용한
영상처리

저자 허진경

주변 픽셀과의 밝기 변화가 많은 곳은 고주파로, 변화가 적은 곳은 저주파로 표현이 가능

이미지에서 고주파의 의미는 경계선을 의미하고, 저주파는 배경을 의미합니다

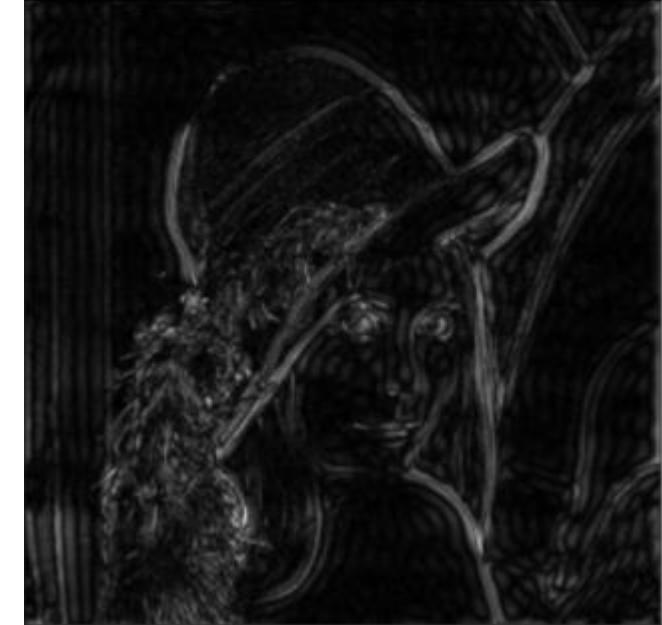
고주파를 제거하면 경계선이 사라지고, 저주파를 제거하면 경계선만 남게 됩니다.



원본 이미지



고주파 영역 제거



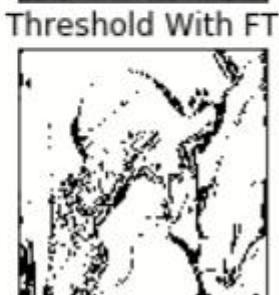
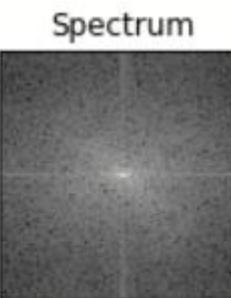
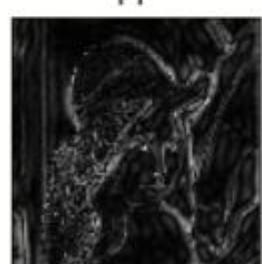
저주파 영역 제거

```
1 #-*- coding:utf-8 -*-
2 """
3 """
4 # Fourier Transform(푸리에 변환)
5     . 시간 도메인(x축)에서 표현된 신호(일반적인 파형 도표)를 주파수 도메인으로 변환.
6     . 시간축이 제거되어 대상의 전체적인 특징을 파악할 수 있음.
7     . 이미지에 적용이 되어 중심이 저주파 영역, 주변이 고주파 영역을 나타냄.
8     . 푸리에 변환을 하여 저주파 또는 고주파를 제거하고 다시 역으로 이미지로 변환 함으로써
9         이미지가공을 할 수 있음.
10    (ex; 푸리에 변환 후 중심의 저주파를 제거하고 다시 Image로 전환 하면 이미지의 경계선만 남게 됨.
11        푸리에 변환 후 주변의 고주파를 제거하면 모아레 패턴(휴대폰으로 모니터를 찍었을 때 나타나는 현상)
12        을 제거할 수 있음.(모니터의 고주파를 제거함.|)
13 )
14 """
15 import cv2
16 import numpy as np
17 from matplotlib import pyplot as plt
18
19
20 img = cv2.imread('lena.jpg')
21 b,g,r = cv2.split(img)
22 img = cv2.merge([r,g,b])
23 img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
25 # Fourier Transform을 적용.
26 적용을 하면 0,0, 즉 화면 좌측상단점이 중심이고, 거기에 저주파가 모여 있음.
27 분석을 용이하게 하기 위해 0,0을 이미지의 중심으로 이동 시키고 Log Scaling을 하여
28 분석이 용이한 결과값으로 변환
29 """
30 f = np.fft.fft2(img) # 이미지에 푸리에 변환 적용
31 fshift = np.fft.fftshift(f) #분석을 용이하게 하기 위해 주파수가 0인 부분을 중앙에 위치시킴. 중앙에 저주파가 -
32 magnitude_spectrum = 20*np.log(np.abs(fshift)) #spectrum 구하는 수학식.
33
34 rows, cols = img.shape
35 crow, ccol = int(rows/2), int(cols/2) # 이미지의 중심 좌표
36
37 # 중앙에서 10x10 사이즈의 사각형의 값을 1로 설정함. 중앙의 저주파를 모두 제거
38 # 저주파를 제거하였기 때문에 배경이 사라지고 경계선만 남게 됨.
39 d = 10
40 fshift[crow-d:crow+d, ccol-d:ccol+d] = 1
41
42 #푸리에 변환결과를 다시 이미지로 변환
43 f_ishift = np.fft.ifftshift(fshift)
44 img_back = np.fft.ifft2(f_ishift)
45 img_back = np.abs(img_back)
46
47 #threshold를 적용하기 위해 float type을 int type으로 변환
48 img_new = np.uint8(img_back);
49 ret, thresh = cv2.threshold(img_new,30,255,cv2.THRESH_BINARY_INV)
50
51 plt.subplot(221),plt.imshow(img, cmap = 'gray')
52 plt.title('Input Image'), plt.xticks([]), plt.yticks([])
```

03 넘파이를 이용한 푸리에 변환 3/3

```
53  
54 plt.subplot(222),plt.imshow(magnitude_spectrum, cmap = 'gray')  
55 plt.title('Spectrum'), plt.xticks([]), plt.yticks([])  
56  
57 plt.subplot(223),plt.imshow(img_back, cmap = 'gray')  
58 plt.title('FT'), plt.xticks([]), plt.yticks([])  
59  
60 plt.subplot(224),plt.imshow(thresh, cmap = 'gray')  
61 plt.title('Threshold With FT'), plt.xticks([]), plt.yticks([])  
62 plt.show()
```

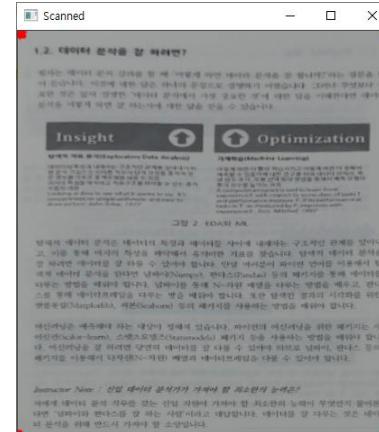
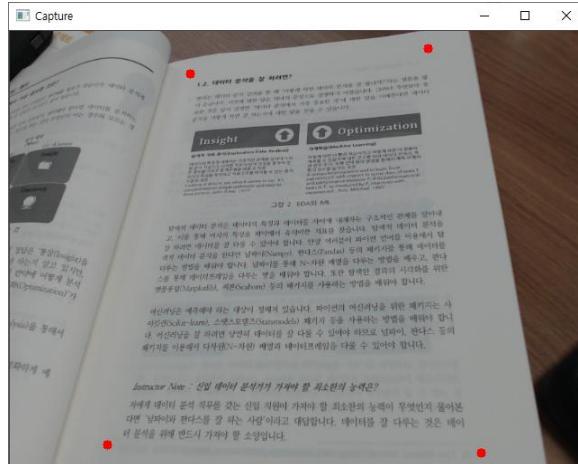
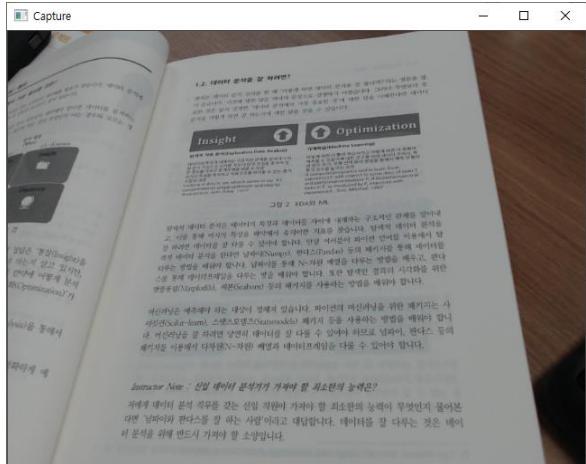


```

1 import cv2
2 from matplotlib import pyplot as plt
3
4 img = cv2.imread('lenagray.png',0)
5 dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
6
7 dft_shift = np.fft.fftshift(dft)
8 magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:, :, 0],dft_shift[:, :, 1]))
9
10 rows, cols = img.shape
11 crow,ccol = int(rows/2) , int(cols/2)
12
13 # 아래는 d 사이지의 사각형을 생성한 후, 사각형 바깥쪽을 제거하는 형태임.
14 # 즉, 고주파영역을 제거하게 됨.
15 # d값이 작을수록 사각형이 작고, 바깥영역 즉, 고주파영역이 많이 제거되기 때문에 이미지가 뭉개지고
16 # d값이 클수록 사각형이 크고, 바깥영역 즉, 고주파 영역이 적게 제거되기 때문에 원래 이미지와 가까워짐.
17
18 d = 30
19 mask = np.zeros((rows,cols,2),np.uint8)
20 mask[crow-d:crow+d, ccol-d:ccol+d] = 1
21 # apply mask and inverse DFT
22 fshift = dft_shift*mask
23 f_ishift = np.fft.ifftshift(fshift)
24 img_back = cv2.idft(f_ishift)
25 img_back = cv2.magnitude(img_back[:, :, 0],img_back[:, :, 1])
26
27 plt.subplot(121),plt.imshow(img, cmap = 'gray')
28 plt.title('Input Image'), plt.xticks([]), plt.yticks([])
29 plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
30 plt.title('FT'), plt.xticks([]), plt.yticks([])
31 plt.show()

```





카메라 연결 후 화면을
캡쳐하려면 C를 누릅니다.

스캔할 사각형 모서리
4곳을 클릭합니다.

직사각형으로 역
Perspective 변환된
이미지가 만들어집니다.



4장. 윤곽선 검출

윤곽선을 찾는 방법에 대해 설명합니다.

- 1절. 필터
- 2절. 윤곽선 추출 필터
- 3절. 캐니엣지

4장. 윤곽선 검출

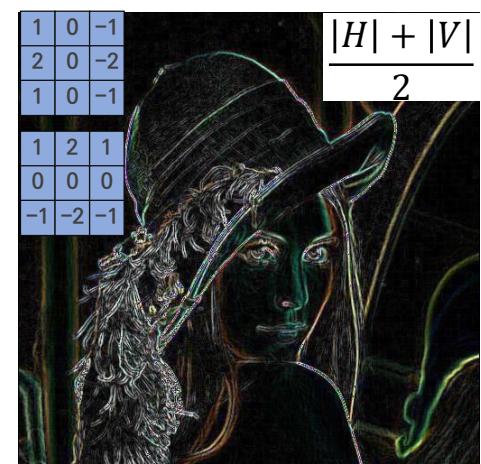
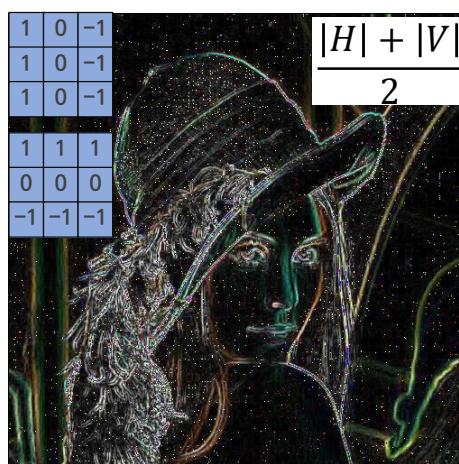
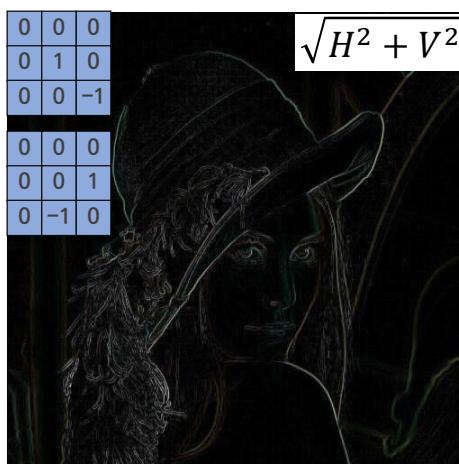
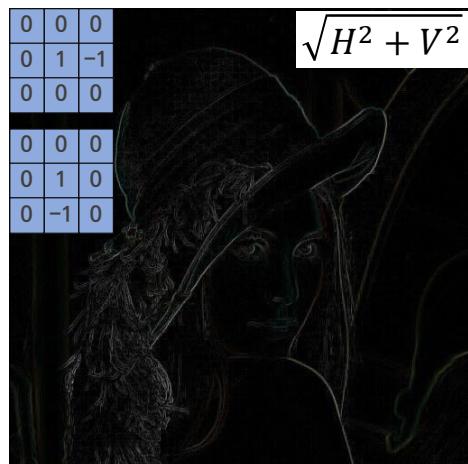


1절. 필터

파이썬 OpenCV를 이용한
영상처리

저자 허진경

행렬의 특성에 따라 원본 이미지로부터 특성이 강조된 이미지를 얻을 수 있음



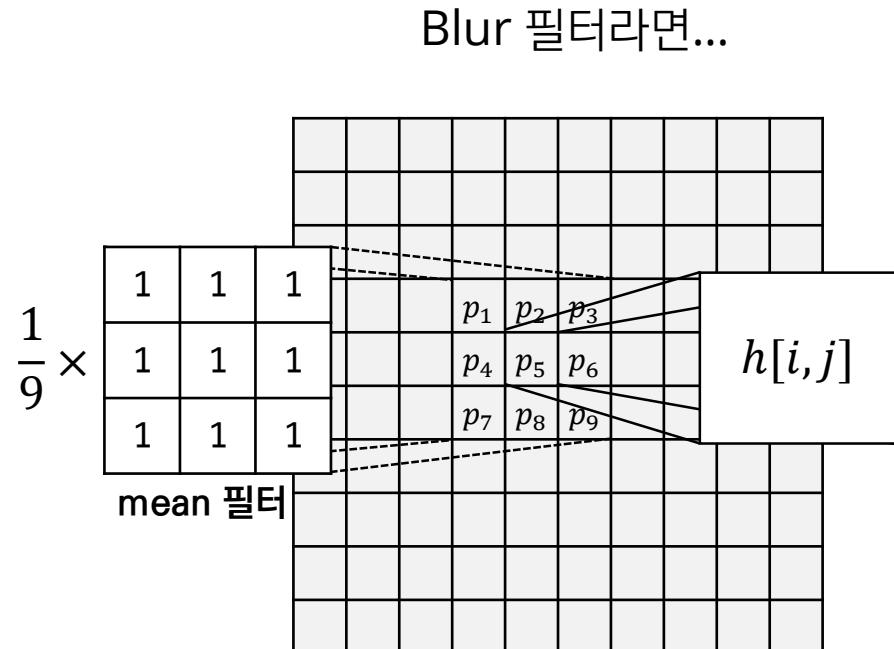
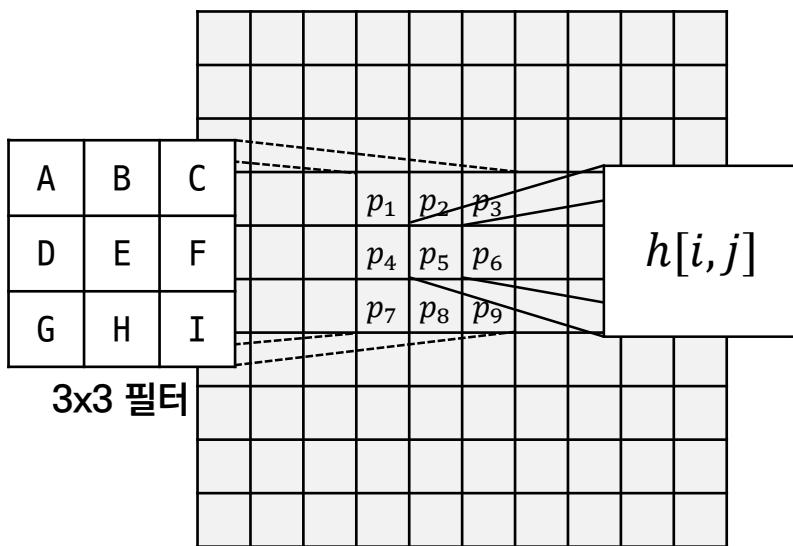
4장. 윤곽선 검출



2절. 합성곱

파이썬 OpenCV를 이용한
영상처리

저자 허진경



$$h[i, j] = A * p_1 + B * p_2 + C * p_3 + D * p_4 + E * p_5 + F * p_6 + G * p_7 + H * p_8 + I * p_9$$

1	3	3	0	0
0	3	1	0	0
0	0	2	1	1
0	0	3	0	1
3	2	1	3	0

5x5 입력 영상

0	1	0
1	0	1
1	1	0

3x3 필터

0	0	0	0	0
0	4	8	4	0
0	5	5	6	0
0	8	5	9	0
0	0	0	0	0

출력 영상

Zero padding : 5x5 이미지의 바깥 화소는 필터를 적용할 수 없으므로 0으로 채움

$$\text{img}[1,1] = (1 \times 0) + (3 \times 1) + (3 \times 0) + (0 \times 1) + (3 \times 0) + (1 \times 1) + (0 \times 1) + (2 \times 0) = 4$$

1	3	3	0	0
0	3	1	0	0
0	0	2	1	1
0	0	3	0	1
3	2	1	3	0

5x5 입력 영상

0	1	0
1	0	1
1	1	0

3x3 필터

0	0	0	0	0
0	4	8	4	0
0	5	5	6	0
0	8	5	9	0
0	0	0	0	0

출력 영상

$$\text{img}[1,2] = (3 \times 0) + (3 \times 1) + (0 \times 0) + (3 \times 1) + (1 \times 0) + (0 \times 1) + (2 \times 1) + (1 \times 0) = 8$$

1	3	3	0	0
0	3	1	0	0
0	0	2	1	1
0	0	3	0	1
3	2	1	3	0

5x5 입력 영상

0	1	0
1	0	1
1	1	0

3x3 필터

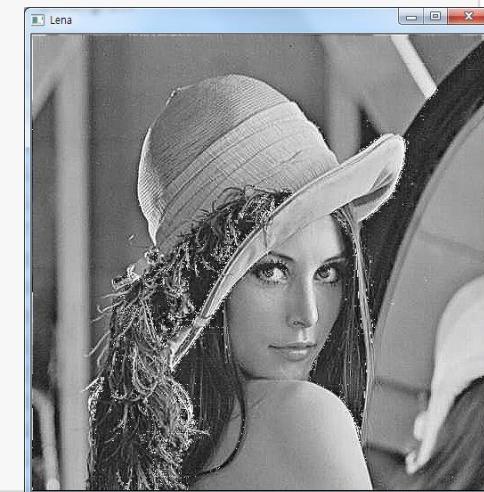
0	0	0	0	0
0	4	8	4	0
0	5	5	6	0
0	8	5	9	0
0	0	0	0	0

출력 영상

$$\text{img}[1,3] = (3 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times 1) + (2 \times 1) + (1 \times 1) + (1 \times 0) = 4$$

현재 화소와 주위의 화소에 필터(3x3 또는 5x5)를 적용시켜 영상처리 합니다.

```
1 def sharpen(img):
2     A = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape
4     for y in range(height):
5         for x in range(width):
6             try :
7                 S =  0*img[y-1, x-1] + -1*img[y-1, x] +  0*img[y-1, x+1] \
8                     + -1*img[y , x-1] +  5*img[y , x] + -1*img[y , x+1] \
9                     +  0*img[y+1, x-1] + -1*img[y+1, x] +  0*img[y+1, x+1]
10                if S>255:
11                    A[y,x] = 255
12                elif S<0:
13                    A[y,x] = 0
14                else:
15                    A[y,x] = S
16            except:
17                pass
18
19 return A
```



```
1 import numpy as np
2 def Conv2D(img, mask=None, padding="valid"):
3     if mask is not None :
4         sy, sx = int(len(mask)/2), int(len(mask[0])/2) # strides = mask.shape
5         if padding=="same": # zero padding
6             new_shape = img.shape + (sy, sx)
7             img_out = np.zeros(new_shape, dtype=np.uint8)
8         elif padding=="valid": # 처리 못하는 바깥 화소를 처리 안함
9             img_out = np.zeros(img.shape, dtype=np.uint8)
10
11     height, width = img_out.shape
12     for y in range(sy, height-sy):
13         for x in range(sx, width-sx):
14             roi = img[y-sy:y+sy+1, x-sx:x+sx+1]
15             filtered = roi * mask
16             img_out[y, x] = np.uint8(np.abs(np.sum(filtered)))
17
18     return img_out[sy:-sy, sx:-sx]
19 else:
20     print("Mask array not found!")
```

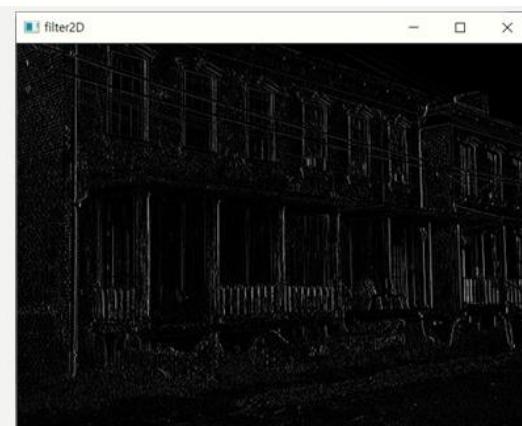
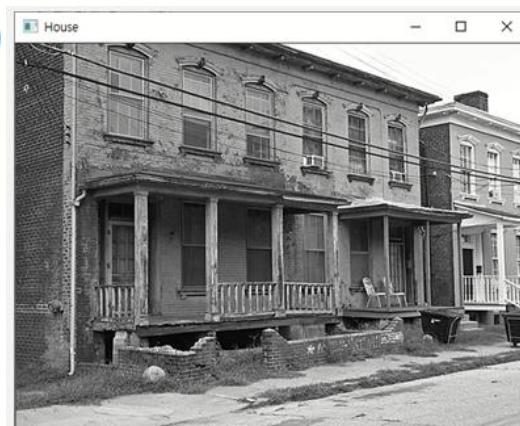
cv2.filter2D(src, ddepth, kernel, anchor=None, delta=None, borderType=None) → dst

- ▶ src - 입력 이미지입니다.
- ▶ ddepth - 출력 영상의 데이터 타입(예: cv2.CV_8U, cv2.CV_32F, cv2.CV_64F)을 지정합니다. -1을 지정하면 src와 같은 타입의 dst 영상을 생성합니다.
- ▶ kernel - 필터 커널 행렬입니다. 실수형입니다.
- ▶ anchor - 고정점 위치입니다. (-1, -1)이면 필터 중앙을 고정점으로 사용합니다.
- ▶ delta - 추가적으로 더할 값입니다.
- ▶ borderType - 가장자리 픽셀 확장 방식을 지정합니다.
- ▶ 반환값 dst - 반환하는 값은 출력 영상입니다.

borderType	설명
BORDER_CONSTANT	iiiiii abcdefgh iiiiii 일부 지정된 i
BORDER_REPLICATE	aaaaaaaa abcdefgh hhhhhhh
BORDER_REFLECT	fedcba abcdefgh hgfedcb
BORDER_WRAP	cdefgh abcdefgh abcdefg
BORDER_REFLECT_101	gfedcb abcdefgh gfedcba
BORDER_TRANSPARENT	uvwxyz abcdefgh ijklmno
BORDER_REFLECT101	BORDER_REFLECT_101와 같음
BORDER_DEFAULT	BORDER_REFLECT_101와 같음
BORDER_ISOLATED	ROI 밖을 사용 안함

04 cv2.filter2D()

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
6
7 kernel = np.array([[0,0,0],[0,1,-1],[0,0,0]])
8 output = cv2.filter2D(img,-1,kernel)
9
10 cv2.imshow("House", img)
11 cv2.imshow("filter2D", output)
12 cv2.waitKey()
13 cv2.destroyAllWindows()
```



4장. 윤곽선 검출



2절. 엣지 검출 필터

파이썬 OpenCV를 이용한
영상처리

저자 허진경

윤곽선 추출 필터 - 차분 필터

```

1 def diff_filter(img):
2     height, width = img.shape
3     img_ = np.zeros(img.shape, dtype=np.uint8)
4     for y in range(height):
5         for x in range(width):
6             try :
7                 H = 0*img[y-1, x-1] + 0*img[y-1, x] + 0*img[y-1, x+1] \
8                     + 0*img[y , x-1] + 1*img[y , x] + -1*img[y , x+1] \
9                     + 0*img[y+1, x-1] + 0*img[y+1, x] + 0*img[y+1, x+1]
10                V = 0*img[y-1, x-1] + 0*img[y-1, x] + 0*img[y-1, x+1] \
11                    + 0*img[y , x-1] + 1*img[y , x] + 0*img[y , x+1] \
12                    + 0*img[y+1, x-1] + -1*img[y+1, x] + 0*img[y+1, x+1]
13                img_[y,x] = np.sqrt(H**2 + V**2)
14            except:
15                pass
16    return img_

```

$$\begin{array}{|c|c|c|} \hline & H & V \\ \hline 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \\ \hline \end{array}
 \quad
 \begin{array}{|c|c|c|} \hline & H & V \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ \hline \end{array}$$

$$\sqrt{H^2 + V^2}$$

윤곽선 추출 필터 – 로버츠 필터

```

1 def reverts_filter(img):
2     height, width = img.shape
3     img_ = np.zeros(img.shape, dtype=np.uint8)
4     for y in range(height):
5         for x in range(width):
6             try :
7                 H = 0*img[y-1, x-1] + 0*img[y-1, x] + 0*img[y-1, x+1] \
8                     + 0*img[y , x-1] + 1*img[y , x] + 0*img[y , x+1] \
9                     + 0*img[y+1, x-1] + 0*img[y+1, x] + -1*img[y+1, x+1]
10
11                 V = 0*img[y-1, x-1] + 0*img[y-1, x] + 0*img[y-1, x+1] \
12                     + 0*img[y , x-1] + 0*img[y , x] + 1*img[y , x+1] \
13                     + 0*img[y+1, x-1] + -1*img[y+1, x] + 0*img[y+1, x+1]
14                 img_[y,x] = np.sqrt(H**2 + V**2)
15             except:
16                 pass
17     return img_

```

H	V
0 0 0	0 0 0
0 1 0	0 0 1
0 0 -1	0 -1 0

$$\sqrt{H^2 + V^2}$$

윤곽선 추출 필터 – 프리위트 필터

```

1 def prewitt_filter(img):
2     img_ = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape[:2]
4     for y in range(height):
5         for x in range(width):
6             try :
7                 H =  1*img[y-1, x-1] +  0*img[y-1, x] + -1*img[y-1, x+1] \
8                     +  1*img[y , x-1] +  0*img[y , x] + -1*img[y , x+1] \
9                     +  1*img[y+1, x-1] +  0*img[y+1, x] + -1*img[y+1, x+1]
10
11                 V =  1*img[y-1, x-1] +  1*img[y-1, x] +  1*img[y-1, x+1] \
12                     +  0*img[y , x-1] +  0*img[y , x] +  0*img[y , x+1] \
13                     + -1*img[y+1, x-1] + -1*img[y+1, x] + -1*img[y+1, x+1]
14
15                 img_[y,x] = (np.abs(H) + np.abs(V))/2
16             except:
17                 pass
18
return img_

```

H	V
1 0 -1	1 1 1
1 0 -1	0 0 0
1 0 -1	-1 -1 -1

$$\frac{|H| + |V|}{2}$$

윤곽선 추출 필터 – 소벨 필터

```

1 def sobel_filter(img):
2     img_ = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape[:2]
4     for y in range(height):
5         for x in range(width):
6             try :
7                 H =  1*img[y-1, x-1] +  0*img[y-1, x] + -1*img[y-1, x+1] \
8                     + 2*img[y , x-1] +  0*img[y , x] + -2*img[y , x+1] \
9                     + 1*img[y+1, x-1] +  0*img[y+1, x] + -1*img[y+1, x+1]
10
11                 V =  1*img[y-1, x-1] +  2*img[y-1, x] +  1*img[y-1, x+1] \
12                     + 0*img[y , x-1] +  0*img[y , x] +  0*img[y , x+1] \
13                     + -1*img[y+1, x-1] + -2*img[y+1, x] + -1*img[y+1, x+1]
14
15                 img_[y,x] = (np.abs(H) + np.abs(V))/2
16             except:
17                 pass
18
    return img_

```

H	V
1 0 -1	1 2 1
2 0 -2	0 0 0
1 0 -1	-1 -2 -1

$$\frac{|H| + |V|}{2}$$

윤곽선 추출 필터 – 라플라시안 필터

```

1 def laplacian1_filter(img):
2     img_ = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape[:2]
4     for y in range(height):
5         for x in range(width):
6             try :
7                 T = 0*img[y-1, x-1] + -1*img[y-1, x] + 0*img[y-1, x+1] \
8                     + -1*img[y , x-1] + 4*img[y , x] + -1*img[y , x+1] \
9                     + 0*img[y+1, x-1] + -1*img[y+1, x] + 0*img[y+1, x+1]
10
11                 img_[y,x] = np.abs(T)
12             except:
13                 pass
14

```

Image Processing - Edge Detection

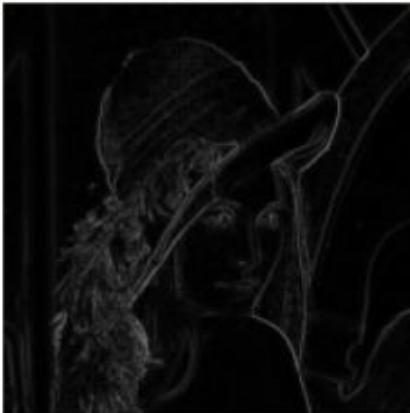
gray



two tone



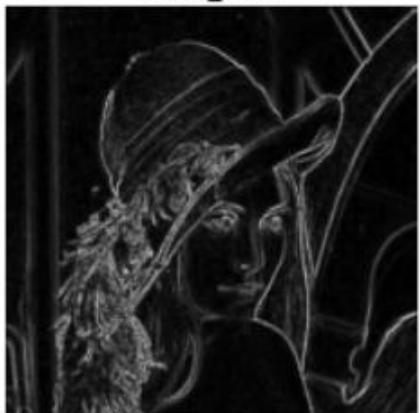
diff_filter



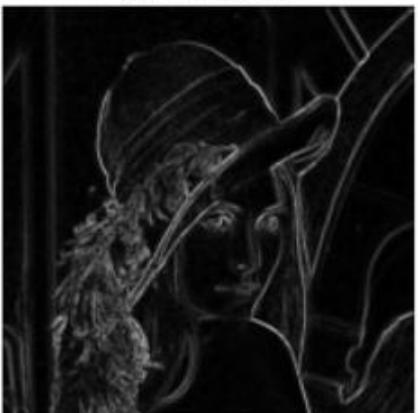
reverts_filter



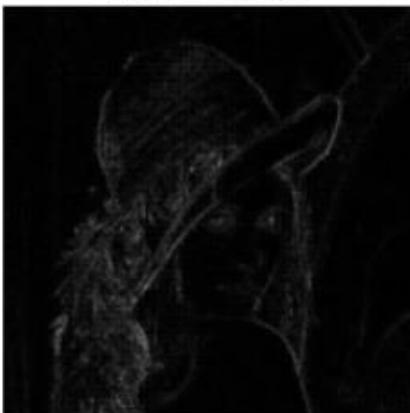
sovel_filter



prewitt_filter



laplacian_filter



cv2.Laplacian



4장. 윤곽선 검출



3절. 캐니 엣지

파이썬 OpenCV를 이용한
영상처리

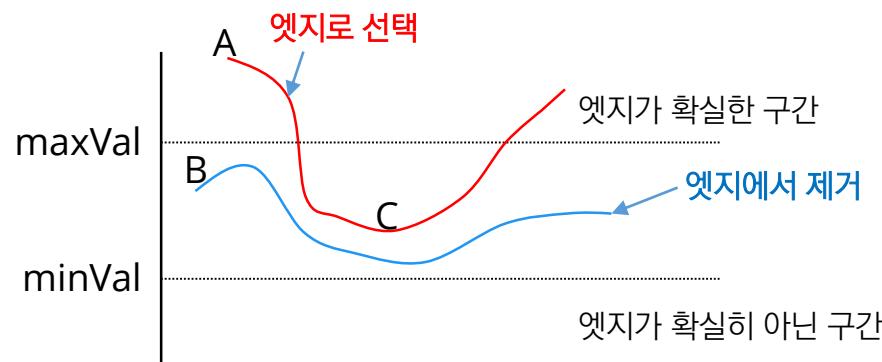
저자 허진경

1986년 존 캐니(John F. Canny)가 제안한 알고리즘

캐니 엣지 알고리즘



- ▶ 노이즈 제거 : 5x5 가우시안 블러링 필터 적용
- ▶ 엣지 검출 : 소벨 필터로 윤곽선 검출
- ▶ 비 최대치 억제 : 그레디언트 방향에서 검출된 엣지 중 가장 큰 값만 선택
- ▶ 이력 스레스홀딩 : 두 개의 경계값(Max, Min)을 지정해서 경계 영역에 있는 화소들 중 큰 경계값(Max) 밖의 픽셀과 연결성이 없는 화소를 제거





cv2.Canny(img, minVal, maxVal, apertureSize, L2gradient)

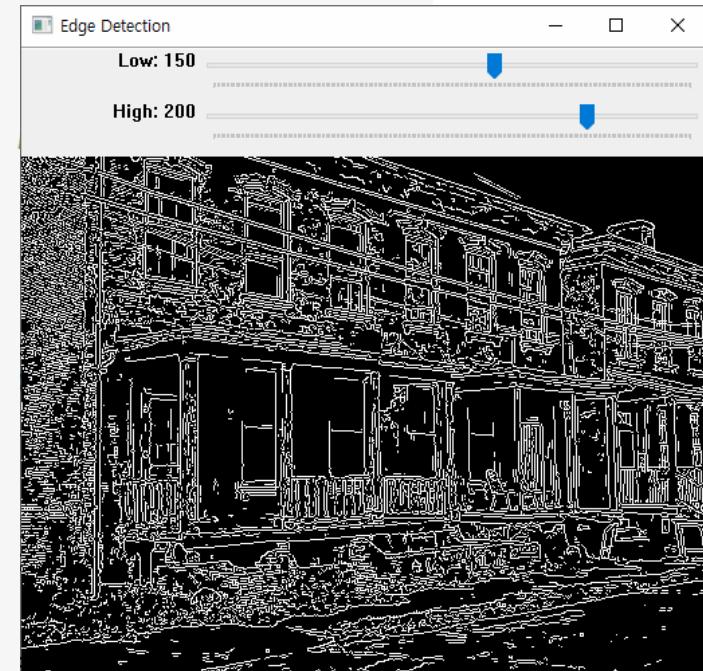
- ▶ img – 입력 이미지입니다.
- ▶ minVal – 캐니 엣지 알고리즘이 마지막 단계인 Hysteresis를 수행하기 위한 임계값1입니다.
- ▶ maxVal – 캐니 엣지 알고리즘이 마지막 단계인 Hysteresis를 수행하기 위한 임계값2입니다.
- ▶ apertureSize – 소벨 필터의 크기입니다. 3, 5, 7만 지정 가능합니다.
- ▶ L2gradient – 기울기 크기를 찾기 위한 방정식을 지정합니다. True이면 앞에서 설명한 Edge_Gradient 방정식을 사용하고 False이면 $Edge_Gradient (G) = |G_x| + |G_y|$ 를 이용합니다.

04 cv2.Canny()

```
1 lena_img = cv2.imread("./images/lena.jpg", cv2.IMREAD_GRAYSCALE)
2 cv2.imshow("Lena", cv2.Canny(lena_img, 100, 200))
3 cv2.waitKey()
4 cv2.destroyAllWindows()
```



```
1 def onChange(x):
2     low = cv2.getTrackbarPos("Low", "Edge Detection")
3     high = cv2.getTrackbarPos("High", "Edge Detection")
4
5     if low > high:
6         pass # print("Low threshold must be low than
7 else:
8     canny = cv2.Canny(img, low, high)
9     cv2.imshow("Edge Detection", canny)
10
11 import cv2
12
13 img = cv2.imread("house.jpg", cv2.IMREAD_GRAYSCALE)
14
15 cv2.namedWindow("Edge Detection")
16
17 cv2.createTrackbar("Low", "Edge Detection", 0, 255, onChange)
18 cv2.createTrackbar("High", "Edge Detection", 0, 255, onChange)
19 cv2.imshow("Edge Detection", img)
20 cv2.waitKey()
21 cv2.destroyAllWindows()
```





5장. 화질 개선

윤곽선을 찾는 방법에 대해 설명합니다.

- 1절. 잡음 제거
- 2절. 히스토그램과 밝기값 조절
- 2절. 콘트라스트 강조
- 3절. 히스토그램 평탄화

5장. 화질 개선



1절. 잡음 제거

파이썬 OpenCV를 이용한
영상처리

저자 허진경

주목 화소의 8근방 화소의 **평균**을 출력합니다.

	p_0	p_1	p_2
	p_3	p_4	p_5
	p_6	p_7	p_8

$$\frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

```

1 def mean_blur(img):
2     A = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape
4     for y in range(height):
5         for x in range(width):
6             try :
7                 S = 1*img[y-1, x-1] + img[y-1, x] + img[y-1, x+1] \
8                     + img[y , x-1] + img[y , x] + img[y , x+1] \
9                     + img[y+1, x-1] + img[y+1, x] + img[y+1, x+1]
10                S = S/9
11                if S>255:
12                    A[y,x] = 255
13                elif S<0:
14                    A[y,x] = 0
15                else:
16                    A[y,x] = int(S)
17            except:
18                pass
19
return A

```

1	1	1
1	1	1
1	1	1

$$\frac{1}{9}$$

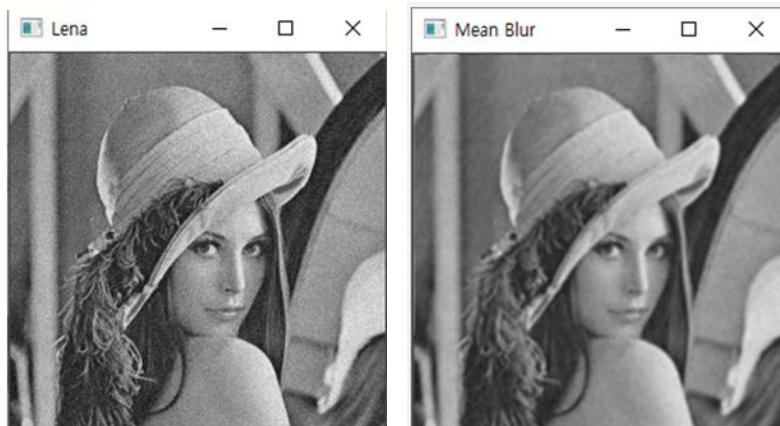


05 cv2.blur()

cv2.blur(img, ksize)

- ▶ img - 입력 이미지입니다.
- ▶ ksize - 필터 커널의 크기입니다. size=3형식으로 지정합니다.

```
1 import cv2  
2  
3 lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)  
4 cv2.imshow("Lena", lena_gray)  
5 cv2.imshow("Mean Blur", cv2.blur(lena_gray, 3))  
6 cv2.waitKey()  
7 cv2.destroyAllWindows()
```

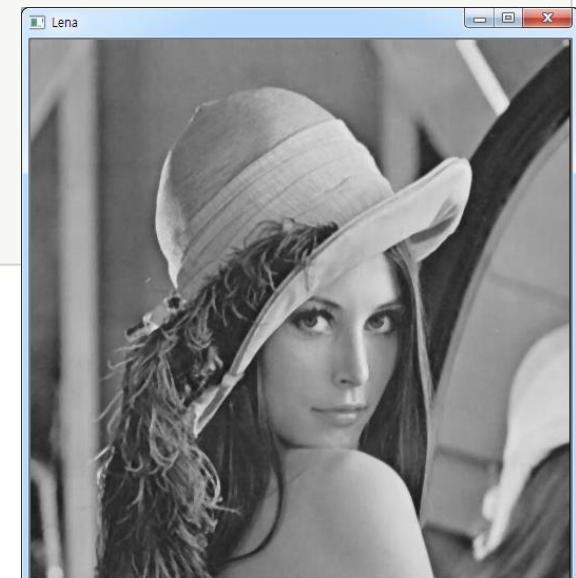


주목 화소의 8근방 화소의 **중위수**를 출력합니다.

p_0	p_1	p_2
p_3	p_4	p_5
p_6	p_7	p_8

$p_0 \sim p_8$ 화소를 정렬한 후 중앙 값을 선택

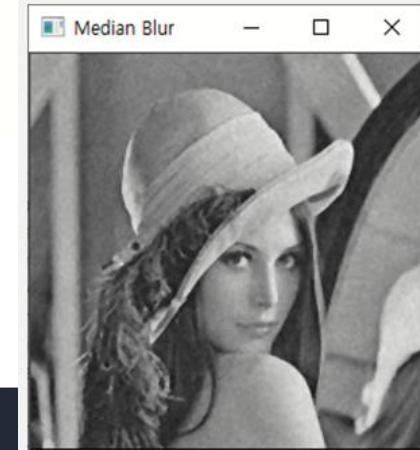
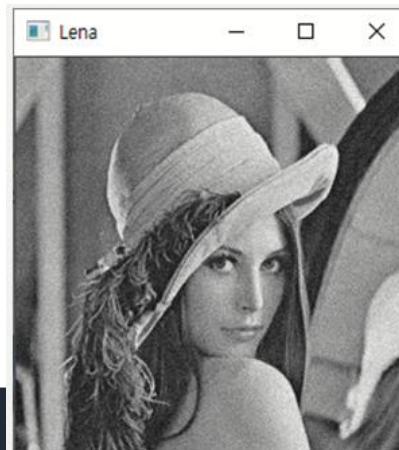
```
1 def median_blur(img):
2     A = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape[:2]
4     for y in range(height):
5         for x in range(width):
6             try :
7                 S = [img[y-1, x-1], img[y-1, x], img[y-1, x+1],
8                      img[y , x-1], img[y , x], img[y , x+1],
9                      img[y+1, x-1], img[y+1, x], img[y+1, x+1] ]
10                A[y, x] = np.median(S)
11            except:
12                pass
13    return A
```



 cv2.medianBlur(img, ksize)

- ▶ img - 입력 이미지입니다.
- ▶ ksize - 필터 커널의 크기입니다. size=3형식으로 지정합니다. 1보다 큰 홀수여야 합니다.

```
1 import cv2  
2  
3 lena_gray = cv2.imread("lena_std_5.jpg", cv2.IMREAD_GRAYSCALE)  
4 cv2.imshow("Lena", lena_gray)  
5 cv2.imshow("Median", cv2.medianBlur(lena_gray, 3))  
6 cv2.waitKey()  
7 cv2.destroyAllWindows()
```



scipy.ndimage.median_filter(input, size=None, footprint=None, output=None, mode='reflect', cval=0.0, origin=0)

```
1 from scipy import ndimage  
2  
3 result = ndimage.median_filter(lena_grey, size=3)  
4 cv2.imshow("Lena", median_blur(result))  
5 cv2.waitKey()  
6 cv2.destroyAllWindows()
```

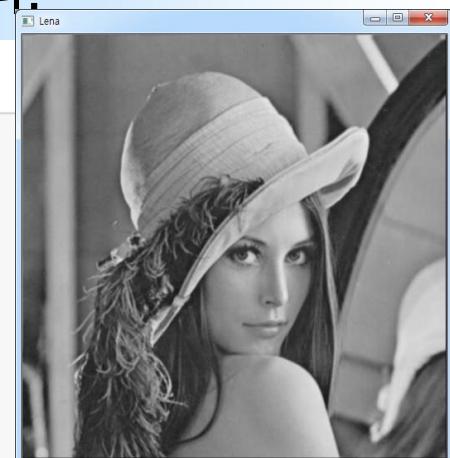


https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html

가우시안 분포를 갖는 커널을 이용해서 필터 처리

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \frac{1}{16}$$

중앙값이 가장 크고 멀어질 수록 그 값이 작아지는 커널을 사용합니다.



```

1 def gaussian_blur(img):
2     img_ = np.zeros(img.shape, dtype=np.uint8)
3     height, width = img.shape
4     for y in range(height):
5         for x in range(width):
6             try :
7                 T = 1*img[y-1, x-1] + 2*img[y-1, x] + 1*img[y-1, x+1] \
8                     + 2*img[y , x-1] + 4*img[y , x] + 2*img[y , x+1] \
9                     + 1*img[y+1, x-1] + 2*img[y+1, x] + 1*img[y+1, x+1]
10                img_[y,x] = int(T/16)
11            except:
12                pass
13    return img_

```



cv2.GaussianBlur(img, ksize, sigmaX)

- ▶ img – 입력 이미지입니다.
- ▶ ksize – 필터 커널의 크기입니다. ksize=(width, height) 형태로 지정합니다. width와 height는 서로 다를 수 있지만, 양수의 홀수로 지정해야 합니다.
- ▶ sigmaX – X방향 가우시안 커널 표준편차입니다.

5장. 화질 개선



2절. 모폴로지 연산

파이썬 OpenCV를 이용한
영상처리

저자 허진경

반복적으로 영역을 확장시켜 구멍을 채우거나, 잡음 또는 떨어진 부분을 제거하는 등 연산

침식(erode), 팽창(dilate), 열기(opening), 닫기(closing) 등이 있음

`cv2.erode(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst`

`cv2.dilate(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst`

`cv2.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst`

수축 : 화소의 부근에 하나라도 0이면 그 화소를 0으로, 그 외는 255로 처리

팽창 : 화소의 부근에 하나라도 255이면 그 화소를 255로, 그 외는 0으로 처리

```

1 def contraction(img_in):
2     height, width = img_in.shape
3     img_out = np.zeros(img_in.shape, np.uint8)
4     for y in range(1, height-1):
5         for x in range(1, width-1):
6             img_out[y,x] = img_in[y,x]
7             if img_in[y-1,x-1]==0:
8                 img_out[y,x] = 0
9             if img_in[y-1,x ]==0:
10                img_out[y,x] = 0
11            if img_in[y-1,x+1]==0:
12                img_out[y,x] = 0
13
14            if img_in[y ,x-1]==0:
15                img_out[y,x] = 0
16            if img_in[y ,x+1]==0:
17                img_out[y,x] = 0
18
19            if img_in[y+1,x-1]==0:
20                img_out[y,x] = 0
21            if img_in[y+1,x ]==0:
22                img_out[y,x] = 0
23            if img_in[y+1,x+1]==0:
24                img_out[y,x] = 0
25
    return img_out

```

```

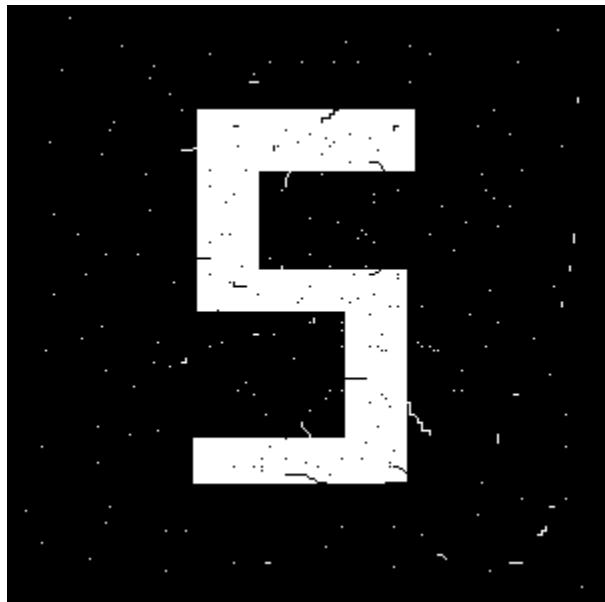
1 def expansion(img_in):
2     height, width = img_in.shape
3     img_out = np.zeros(img_in.shape, np.uint8)
4     for y in range(1, height-1):
5         for x in range(1, width-1):
6             img_out[y,x] = img_in[y,x]
7             if img_in[y-1,x-1]==255:
8                 img_out[y,x] = 255
9             if img_in[y-1,x ]==255:
10                img_out[y,x] = 255
11            if img_in[y-1,x+1]==255:
12                img_out[y,x] = 255
13
14            if img_in[y ,x-1]==255:
15                img_out[y,x] = 255
16            if img_in[y ,x+1]==255:
17                img_out[y,x] = 255
18
19            if img_in[y+1,x-1]==255:
20                img_out[y,x] = 255
21            if img_in[y+1,x ]==255:
22                img_out[y,x] = 255
23            if img_in[y+1,x+1]==255:
24                img_out[y,x] = 255
25
    return img_out

```

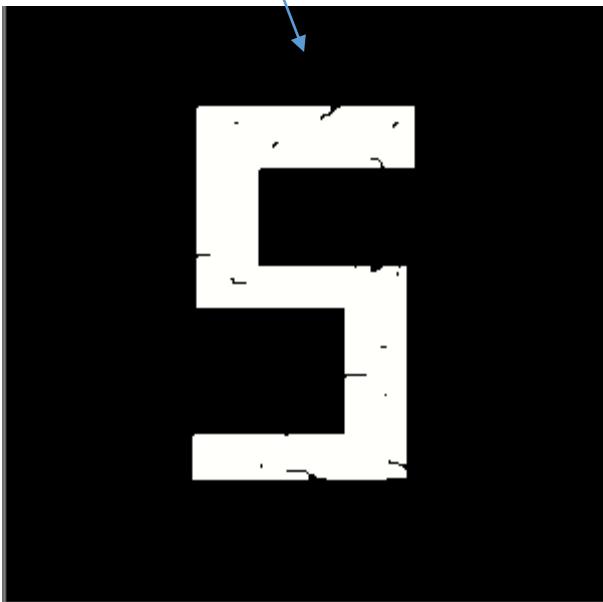
```
1 img5 = cv2.imread("./images/five.png", cv2.IMREAD_GRAYSCALE)
2 cv2.imshow("5", img5)
3 cv2.waitKey()
4 cv2.destroyAllWindows()
```

```
1 img5_cont = img_pro(contraction, img5, show=False)
2 img_pro(expansion, img5_cont)
```

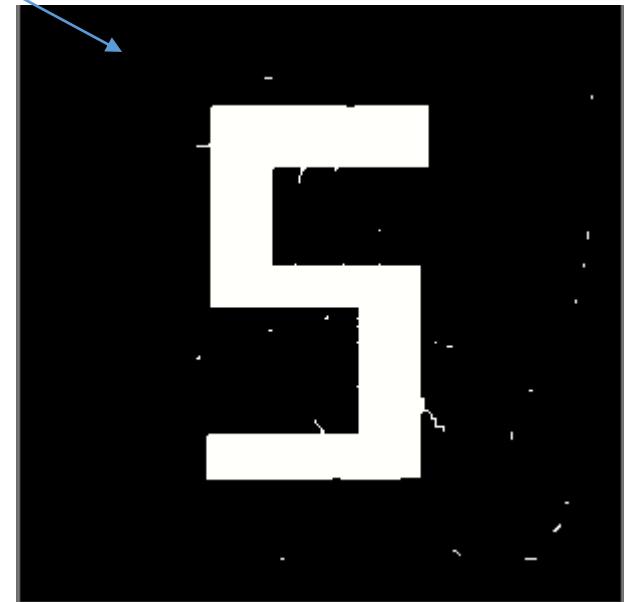
```
1 img5_exp = img_pro(expansion, img5, show=False)
2 img_pro(contraction, img5_exp)
```



원본



수축 후 팽창



팽창 후 수축



cv2.erode(src, dst, kernel, anchor=(-1,-1), iteration=1)

- ▶ src – 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- ▶ dst – src와 동일한 크기 및 타입의 출력 이미지입니다.
- ▶ kernel – 침식에 사용되는 필터커널입니다.
- ▶ anchor – 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- ▶ iteration – 침식이 적용되는 횟수입니다.

```
1 import cv2 as cv
2 import numpy as np
3
4 img = cv.imread('j.png',0)
5 kernel = np.ones((5,5),np.uint8)
6 cv2.imshow("j", img)
7 cv2.imshow("erode", cv.erode(img, kernel , iterations=1))
8 cv2.waitKey()
9 cv2.destroyAllWindows()
```



원본

수축



cv2.dilate(src, dst, kernel, anchor=(-1,-1), iteration=1)

- ▶ src – 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- ▶ dst – src와 동일한 크기 및 타입의 출력 이미지입니다.
- ▶ kernel – 침식에 사용되는 필터커널입니다.
- ▶ anchor – 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- ▶ iteration – 침식이 적용되는 횟수입니다.

```
1 import cv2 as cv
2 import numpy as np
3
4 img = cv.imread('j.png',0)
5 kernel = np.ones((5,5),np.uint8)
6 cv2.imshow("j", img)
7 cv2.imshow("dilate", cv.dilate(img, kernel , iterations=1))
8 cv2.waitKey()
9 cv2.destroyAllWindows()
```



원본

팽창



cv2.morphologyEx(src, dst, op, kernel, anchor=(-1,-1), iteration=1)

- ▶ src – 입력 이미지입니다. 채널 수는 임의로 지정할 수 있지만 깊이는 CV_8U, CV_16U, CV_16S, CV_32F 또는 CV_64F 중 하나여야 합니다.
- ▶ dst – src와 동일한 크기 및 타입의 출력 이미지입니다.
- ▶ op – 모폴로지 타입입니다. 아래 표를 참고하세요.
- ▶ kernel – 침식에 사용되는 필터커널입니다.
- ▶ anchor – 요소 내 현재 화소의 위치입니다. 기본값 (-1,-1)은 현재 화소가 필터 커널의 중심에 있음을 의미합니다.
- ▶ iteration – 침식이 적용되는 횟수입니다.

표 18. 모폴로지 타입

속성	설명
MORPH_ERODE	수축
MORPH_DILATE	팽창
MORPH_OPEN	수축 후 팽창(열기)
MORPH_CLOSE	팽창 후 수축(닫기)
MORPH_GRADIENT	a morphological gradient
MORPH_TOPHAT	"top hat"
MORPH_BLACKHAT	"black hat"
MORPH_HITMISS	"hit or miss" .-CV_8UC1 바이너리 이미지만 지원합니다.

수축 후 팽창은 노이즈를 제거하는 데 유용합니다.

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```



팽창 후 수축은 전경 개체 내부의 작은 구멍이나 개체의 작은 검은 점을 닫을 때 유용합니다.

```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```



그래디언트는 이미지의 팽창과 침식의 차이입니다.

```
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)
```



탑햇은 입력 이미지와 이미지 열기의 차이입니다. 아래 예제는 9x9 커널에 대해 수행됩니다.

```
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
```



블랙햇은 입력 이미지 수축과 입력 이미지의 차이입니다.

```
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)
```



히트미스 변환은 이진 이미지에서 패턴을 찾는 데 유용합니다.



0	0	0
1	1	0
0	1	0

0	-1	-1
0	0	-1
0	0	0

0	-1	-1
1	1	-1
0	1	0

그림 4. (커널). 왼쪽: '적중'할 커널. 중간: '미스'할 커널. 오른쪽: 최종 결합 커널

위의 이미지는 위/오른쪽 끝을 찾는 필터 커널을 적용한 결과입니다.

```
kernel = np.array([[0,-1,-1],[1,1,-1],[0,1,0]]), np.uint8)
```

5장. 화질 개선



3절. 화질 개선

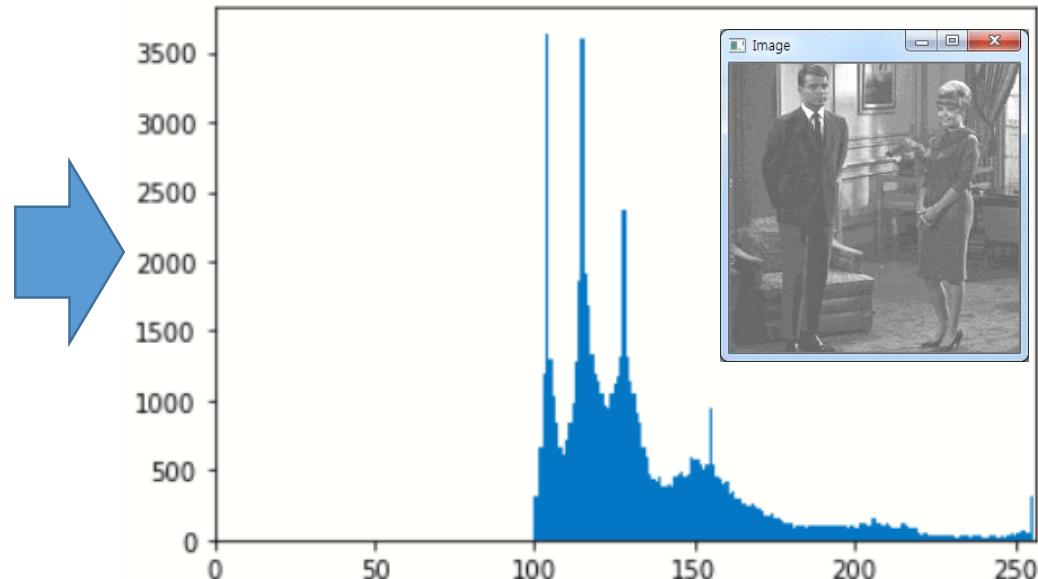
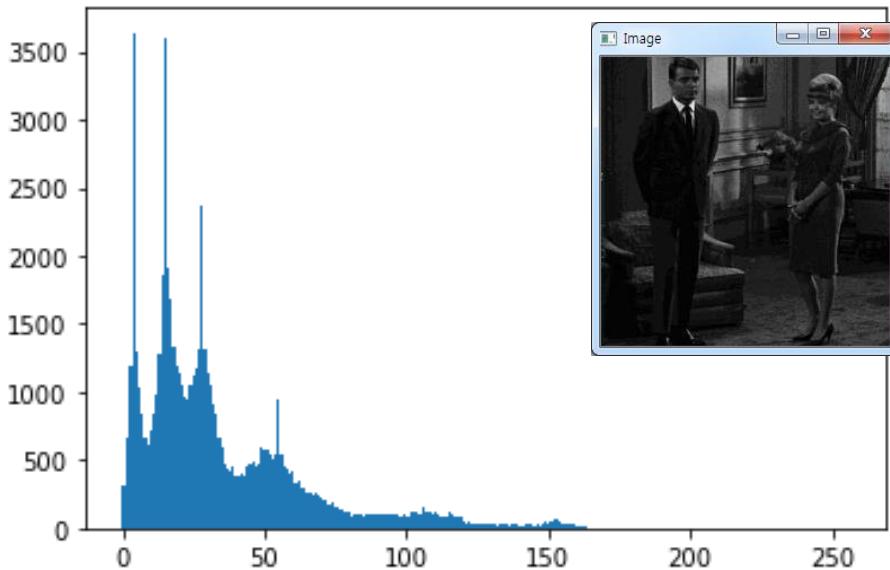
파이썬 OpenCV를 이용한
영상처리

저자 허진경

히스토그램과 밝기값 조절

5장. 화질 개선 / 3절. 화질 개선

```
1 def histogram(img):
2     height, width = img.shape
3     hist = np.zeros(256)
4     for y in range(height):
5         for x in range(width):
6             hist[img[y,x]] = hist[img[y,x]] + 1
7     return hist
```



밝기값 조절

5장. 화질 개선 / 3절. 화질 개선

```
1 def brightness(img, threshold=0):
2     height, width = img.shape
3     img_ = np.zeros(img.shape, dtype=np.uint8)
4
5     for y in range(height):
6         for x in range(width):
7             temp = img[y][x] + threshold;
8             if (temp > 255):
9                 img_[y][x] = 255
10            elif(temp < 0):
11                img_[y][x] = 0
12            else:
13                img_[y][x] = temp
14
15     return img_
```

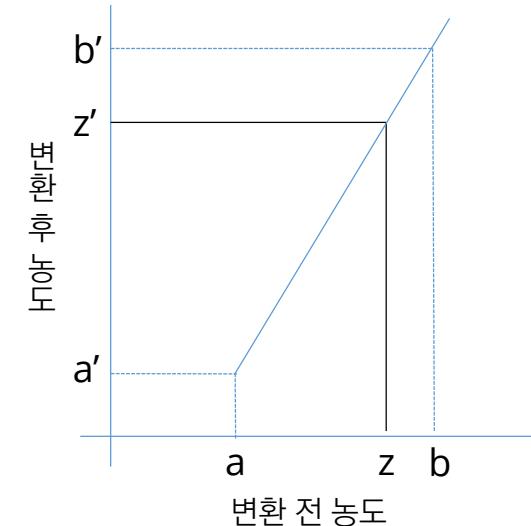
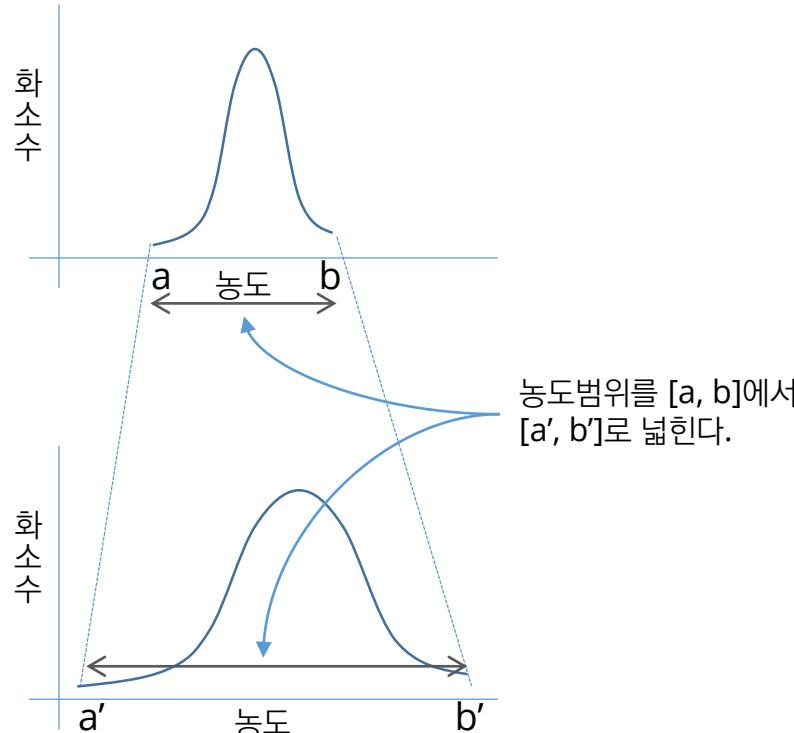


```
1 img = cv2.imread("couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 img_pro2(brightness, img, threshold=50)
```

콘트라스트 강조

5장. 화질 개선 / 3절. 화질 개선

명암의 대비를 극대화시켜 선명한 영상을 만드는 것.



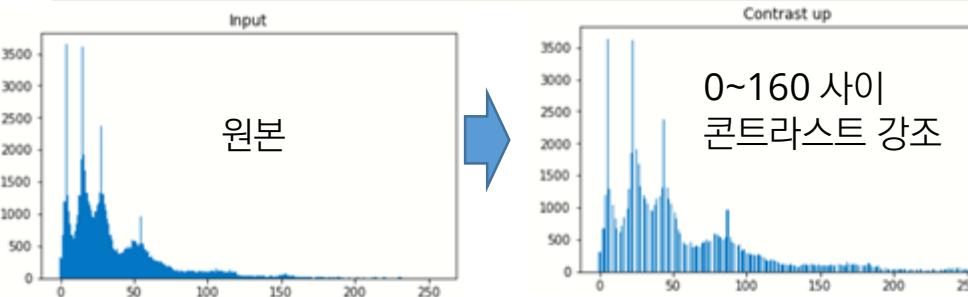
$$z' = \frac{b' - a'}{b - a} * (z - a) + a'$$

$$output = \frac{255 - 0}{high - low} * (x - low) + 0$$

콘트라스트 강조

5장. 화질 개선 / 3절. 화질 개선

```
1 def contrast(img, low=0, high=255):
2     height, width = img.shape
3     img_ = np.zeros(img.shape, dtype=np.uint8)
4
5     for y in range(height):
6         for x in range(width):
7             temp = (int)((255/(high-low)) * (img[y][x]-low));
8             if (temp > 255):
9                 img_[y][x] = 255
10            elif(temp < 0):
11                img_[y][x] = 0
12            else:
13                img_[y][x] = temp
14
return img_
```



cv2.normalize()

5장. 화질 개선 / 3절. 화질 개선

cv2.normalize()는 콘트라스트 강조를 구현한 함수입니다.



`cv2.normalize(src, dst, alpha, beta, flag) → dst`

- ▶ src – 입력 이미지입니다.
- ▶ dst – src와 동일한 크기 및 타입의 출력 이미지입니다.
- ▶ alpha – 정규화하기 위한 구간 1입니다. 이미지 콘트라스트 강조 시 0을 사용합니다.
- ▶ beta – 정규화하기 위한 구간 2입니다. 이미지 콘트라스트 강조 시 255를 사용합니다.
- ▶ flag – 알고리즘을 선택할 플래그입니다. 사용 가능한 플래그는 다음과 같습니다.

`cv2.NORM_MINMAX` : alpha와 beta 구간으로 정규화합니다.

`cv2.NORM_L1` : 전체 합으로 나눕니다.

`cv2.NORM_L2` : 단위벡터로 정규화합니다.

`cv2.NORM_INF` : 최댓값으로 나눕니다.

히스토그램 평활화

5장. 화질 개선 / 3절. 화질 개선

```
▼ 1 def equalization(img):
    2     height, width = img.shape
    3     img_in = img.copy()
    4     img_out = np.zeros(img_in.shape, dtype=np.uint8)
    5     histogram = np.zeros(256, dtype=int) # 화소를 수를 저장해야 함
    6     sum_histo = np.zeros(256, dtype=int) # 히스토그램 누적합
    7
    ▼ 8     for y in range(height):
    ▼ 9         for x in range(width):
        10             histogram[img_in[y,x]] = histogram[img_in[y,x]] + 1
        11
        12         sum = 0
        13         scale_factor = 255 / (width * height) # 정규화된 합을 계산하기 위함
    ▼ 14         for i in range(256):
        15             sum = sum + histogram[i]
        16             sum_histo[i] = np.ceil(sum*scale_factor)
        17
    ▼ 18         for y in range(height):
    ▼ 19             for x in range(width):
        20                 img_out[y,x] = sum_histo[img_in[y][x]]
        21
    22     return img_out
```

Numpy를 이용한 히스토그램 평활화

5장. 화질 개선 / 3절. 화질 개선

```
1 #-*coding:utf-8-*-  
2 import cv2  
3 import numpy as np  
4 from matplotlib import pyplot as plt  
5  
6 img = cv2.imread('couple2.jpg');  
7  
8 hist, bins = np.histogram(img.flatten(), 256,[0,256])  
9  
10 cdf = hist.cumsum()  
11  
12 cdf_m = np.ma.masked_equal(cdf,0) # cdf의 값이 0인 경우는 mask처리를 하여 계산에서 제외  
# mask처리가 되면 Numpy 계산에서 제외가 됨  
# cdf array에서 값이 0인 부분을 mask처리함  
13  
14 cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())  
15  
16 cdf = np.ma.filled(cdf_m,0).astype('uint8') #History Equalization 공식  
17  
18 img2 = cdf[img] # Mask처리를 했던 부분을 다시 0으로 변환  
19 plt.subplot(121),plt.imshow(img),plt.title('Original')  
20 plt.subplot(122),plt.imshow(img2),plt.title('Equalization')  
21 plt.show()
```

cv2.equalizeHist()

5장. 화질 개선 / 3절. 화질 개선

cv2.equalizeHist(src[, dst]) → dst

```
1 import cv2
2 img = cv2.imread("couple2.jpg", cv2.IMREAD_GRAYSCALE)
3 img_ = cv2.equalizeHist(img)
4 cv2.imshow("Flatening", img_)
5 cv2.waitKey()
6 cv2.destroyAllWindows()
```

히스토그램 평활화 전/후

5장. 화질 개선 / 3절. 화질 개선

```
1 img = cv2.imread("images/couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 hist1 = cv2.calcHist(images=[img], channels=[0], mask=None,
3                      histSize=[256], ranges=[0,256]) # 원본 이미지
4 img2 = cv2.equalizeHist(img)
5 hist2 = cv2.calcHist(images=[img2], channels=[0], mask=None,
6                      histSize=[256], ranges=[0,256]) # 히스토그램 평탄화 이미지
```

```
1 plt.figure(figsize=(15,4))
2 plt.subplot(121); plt.imshow(img, cmap="gray"); plt.title("Before")
3 plt.subplot(122); plt.imshow(img2, cmap="gray"); plt.title("After")
4 plt.show()
```

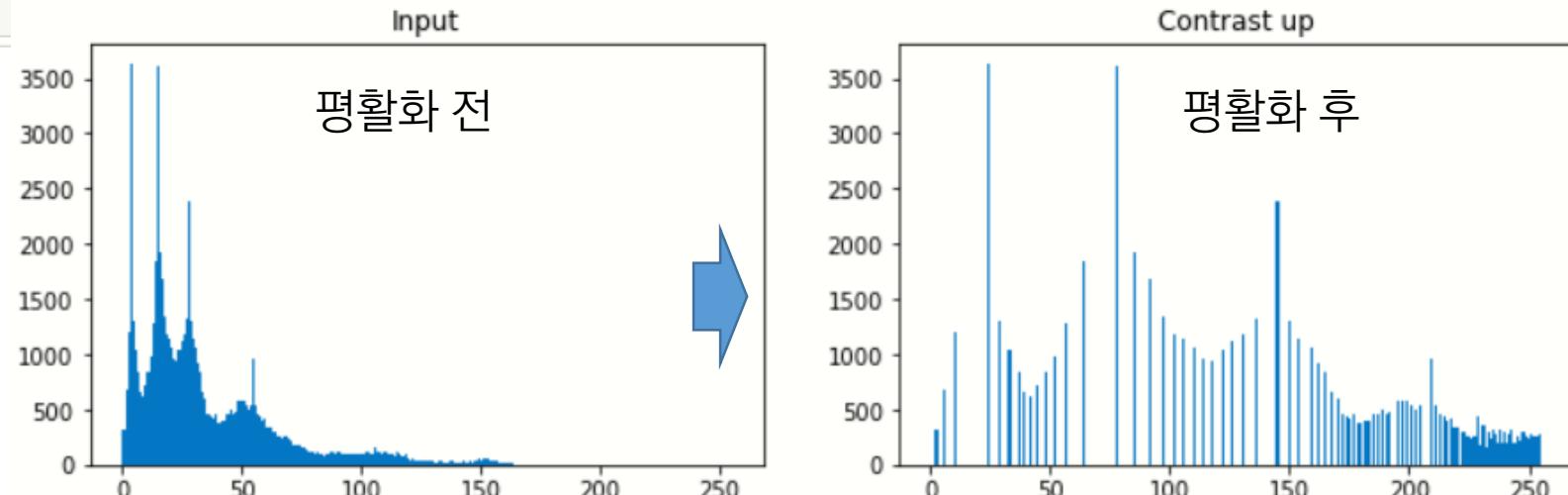


히스토그램 평활화 전/후

5장. 화질 개선 / 3절. 화질 개선

```
1 img = cv2.imread("images/couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 hist1 = cv2.calcHist(images=[img], channels=[0], mask=None,
3                      histSize=[256], ranges=[0,256]) # 원본 이미지
4 img2 = cv2.equalizeHist(img)
5 hist2 = cv2.calcHist(images=[img2], channels=[0], mask=None,
6                      histSize=[256], ranges=[0,256]) # 히스토그램 평활화 이미지
```

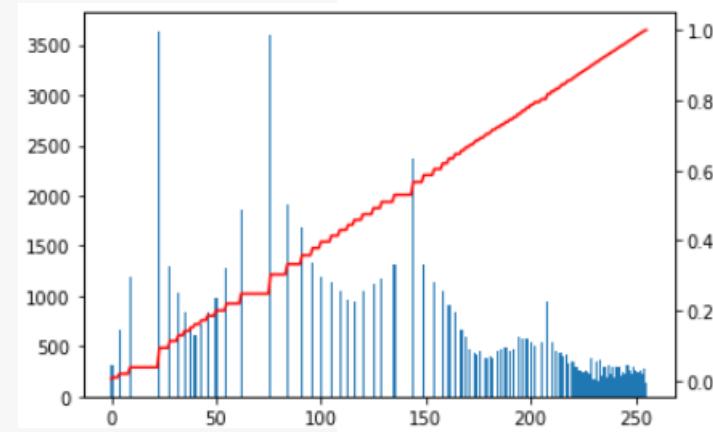
```
1 plt.figure(figsize=(15,4))
2 plt.subplot(121); plt.bar(range(256), hist1.flatten()); plt.title("Before")
3 plt.subplot(122); plt.bar(range(256), hist2.flatten()); plt.title("After")
4 plt.show()
```



누적 히스토그램 그래프

5장. 화질 개선 / 3절. 화질 개선

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 img = cv2.imread("couple256.jpg", cv2.IMREAD_GRAYSCALE)
6 img2 = cv2.equalizeHist(img)
7
8 hist = histogram(img2)
9
10 cumsum_hist = np.cumsum(hist)
11 sum_hist = np.sum(hist)
12 plt.bar(range(256), hist, width=1)
13 ax = plt.twinx()
14 ax.plot(range(256), cumsum_hist/sum_hist, 'r')
15 plt.show()
```



CLAHE(Contrast Limited Adaptive Histogram Equalization)

이미지를 작은 title형태로 나누어 그 title안에서 Equalization을 적용하는 방식입니다. 그런데 여기서도 한가지 문제가 있습니다. 작은 영역이다 보니 작은 노이즈(극단적으로 어둡거나, 밝은 영역)가 있으면 이것이 반영이 되어 원하는 결과를 얻을 수 없게 됩니다. 이 문제를 피하기 위해서 contrast limit라는 값을 적용하여 이 값을 넘어가는 경우는 그 영역은 다른 영역에 균일하게 배분하여 적용을 합니다.

cv2.createCLAHE(clipLimit=40.0, tileGridSize=(8,8))

- ▶ clipLimit : 대비 제한을 위한 임계값입니다.
- ▶ tileGridSize : 히스토그램 이퀄라이제이션을 위한 그리드 크기입니다.
입력 이미지는 동일한 크기의 직사각형 타일로 분할됩니다. tileGridSize는 행과 열의 타일 수를 정의합니다.

cv2.createCLAHE()

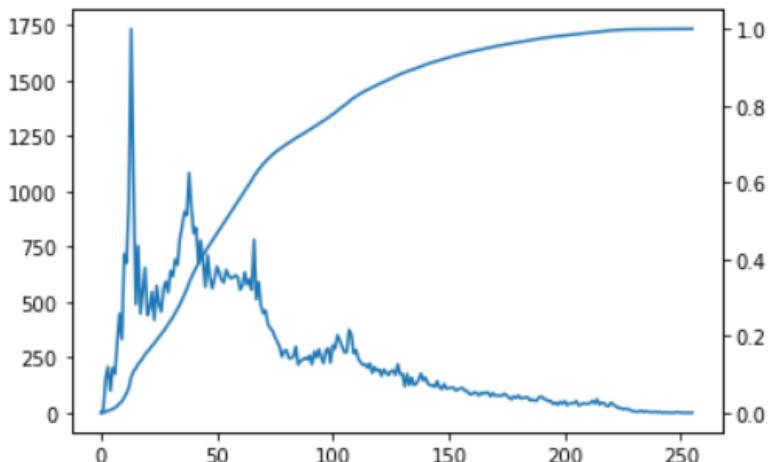
5장. 화질 개선 / 3절. 화질 개선

```
1 #-*-coding:utf-8-*-
2 import cv2
3 import numpy as np
4 from matplotlib import pyplot as plt
5
6
7 img = cv2.imread('couple2.jpg',0);
8
9 # contrast limit가 2.0이고 tileSize는 8x8
10 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
11 img2 = clahe.apply(img)
12
13 img = cv2.resize(img,(400,400))
14 img2 = cv2.resize(img2,(400,400))
15
16 dst = np.hstack((img, img2))
17 cv2.imshow('img',dst)
18 cv2.waitKey()
19 cv2.destroyAllWindows()
```

cv2.createCLAHE()

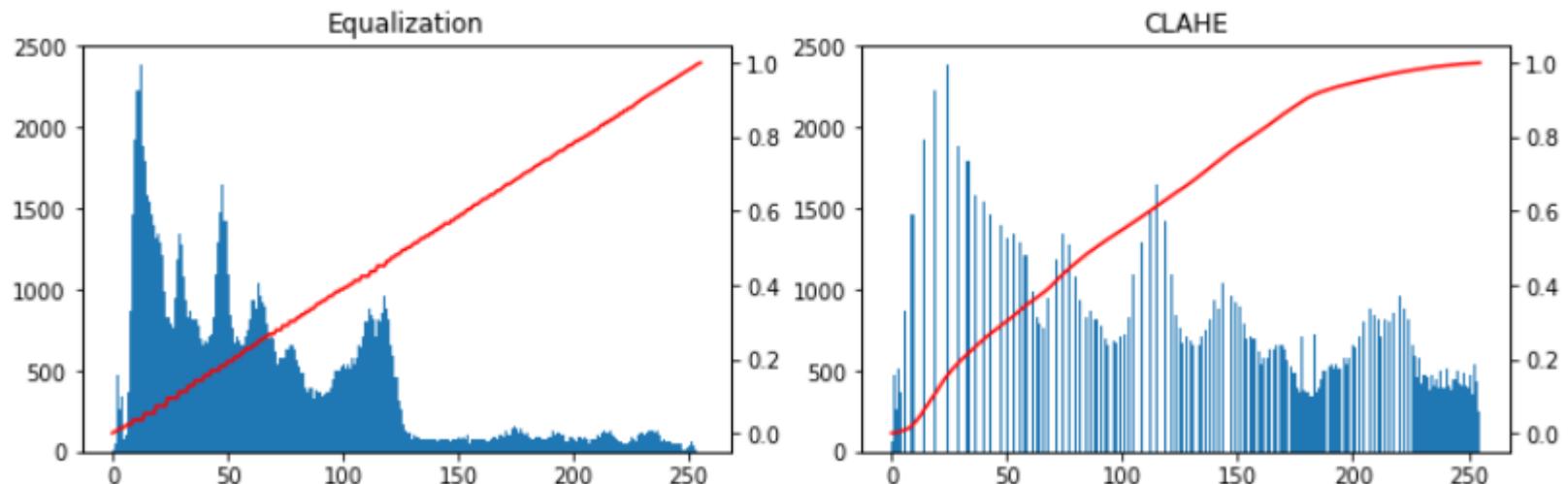
5장. 화질 개선 / 3절. 화질 개선

```
1 img = cv2.imread("images/couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
3 img2 = clahe.apply(img)
4
5 hist2 = histogram(img2)
6 cumsum_hist = np.cumsum(hist2)
7 sum_hist = np.sum(hist2)
8 plt.plot(range(256), hist2)
9 ax = plt.twinx()
10 ax.plot(range(256), cumsum_hist/sum_hist)
11 plt.show()
```



equalization vs. CLAHE

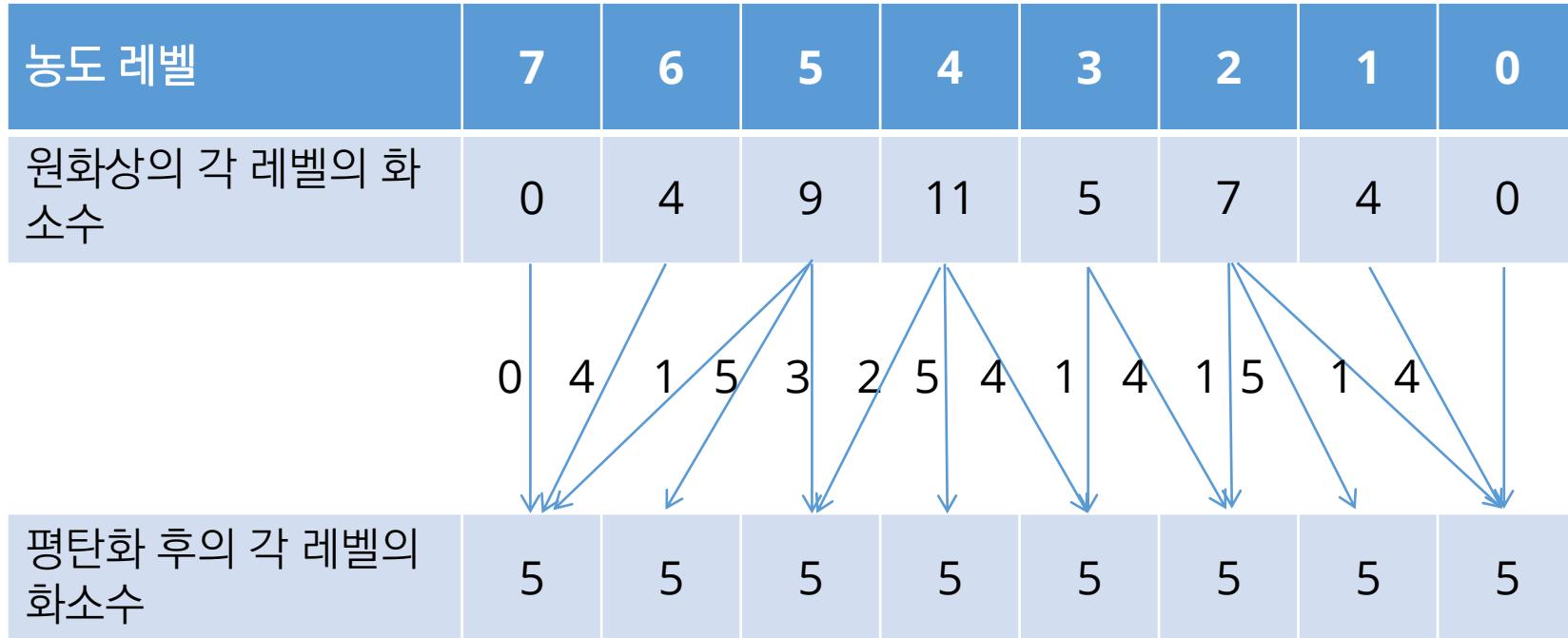
5장. 화질 개선 / 3절. 화질 개선



완벽하게 평평한 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

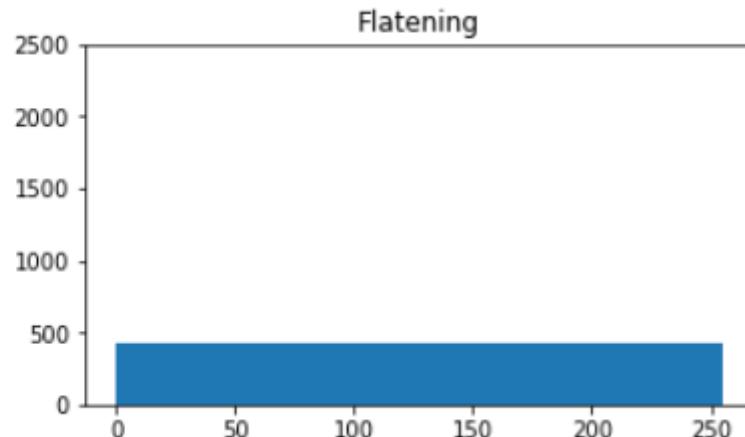
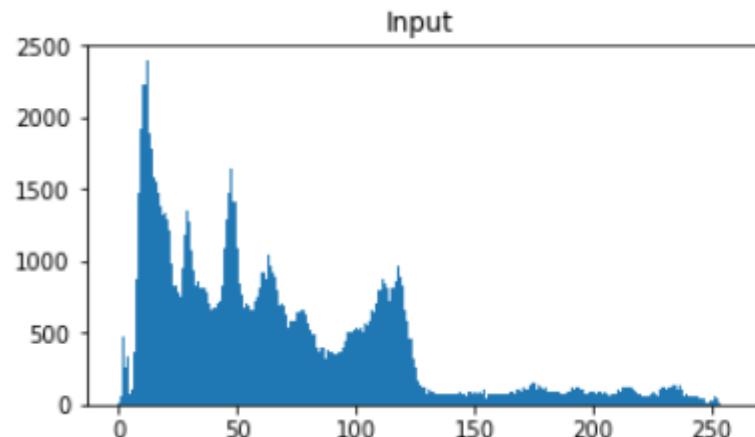
원화상의 화소수가 적은 부분은 압축되고, 화소수가 많은 부분은 신장됩니다.



Histogram Equalization의 결과는 밝은 이미지나 어두운 이미지 어떤 것을 사용해도 동일한 결과가 나옵니다. 이것은 이미지의 인식을 할 때 유용합니다. 예를 들면 얼굴인식을 할 때 대상 이미지를 Equalization을 하고 나면 동일한 밝기가 되기 때문에 동일한 환경에서 작업을 할 수가 있습니다.

완벽하게 평평한 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선



완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

```
1 def flattening(img):
2     height, width = img.shape
3     img_in = img.copy()
4     img_out = np.zeros(img.shape, dtype=np.uint8)
5     histogram = np.zeros(256, dtype=int) # 화소를 수를 저장해야 함
6
7     for y in range(height):
8         for x in range(width):
9             histogram[img_in[y,x]] = histogram[img_in[y,x]] + 1
10
11    low = 255 # 현재 평균화로 옮겨지는 단계
12    high = 255
13    delta = 0 # 주위 화소 레벨에 따라 선택되는 화소 수
14    avg_pixel_count = (int)((height*width)/256) #평탄화 후의 1농도 레벨의 화소 수
15
16    for i in range(255,-1,-1) :
17        sum = 0
18        while (sum < avg_pixel_count):
19            sum = sum + histogram[low]
20            low = low - 1
21
22        low = low + 1
23        delta = histogram[low]-(sum-avg_pixel_count); #평균화 하고 남은 화소값 유지하기 위해
24        buffer = sort_by_weight(img_in, low, histogram[low])
25
26        if (low<high):
27            for y in range(height):
28                for x in range(width):
29                    if (img_in[y,x] >= low+1) & (img_in[y,x] <= high) :
30                        img_out[y,x] = i
31
32        for j in range(delta):
33            img_out[int(buffer[j].y), int(buffer[j].x)] = i
34            img_in[int(buffer[j].y), int(buffer[j].x)] = low+1
35
36        histogram[low] = histogram[low] - delta
37        high = low
38
39    return img_out
```



완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

```
1 class SortedPixel():
2     def __init__(self, x, y, weight):
3         self.x = x
4         self.y = y
5         self.weight = weight
6
7
8     def sort_by_weight(pixels, level, count):
9         height, width = pixels.shape
10        data = []
11        inum = 0
12        weight = 0
13
14        for y in range(height):
15            for x in range(width):
16                try:
17                    if(pixels[y,x]==level):
18                        w = get_weight(pixels, x, y)
19                        data.append(SortedPixel(x, y, w))
20                        inum = inum + 1
21                except:
22                    pass
23
24    from operator import attrgetter
25    data.sort(key=attrgetter('weight')) # weight를 이용해 리스트의 인스턴스를 정렬
26
27    return data
```

완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

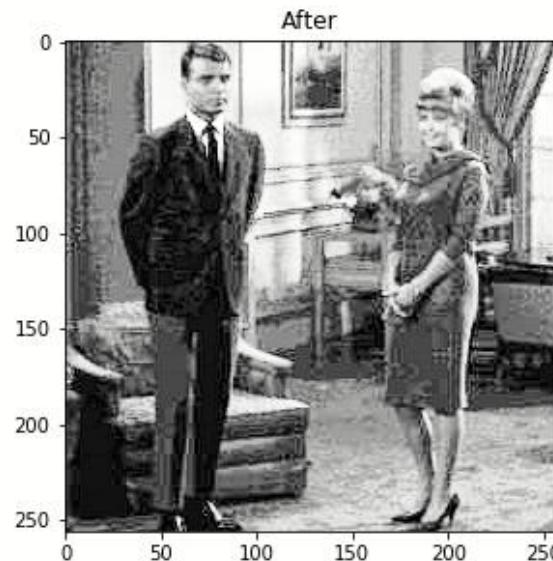
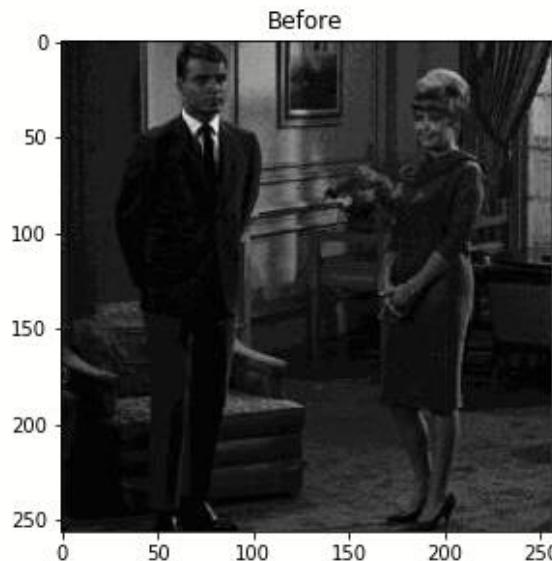
```
1 import numpy as np
2 def get_weight(pixels, x, y):
3     weight = 0
4     xm = x-1
5     ym = y-1
6     xp = x+1
7     yp = y+1
8     d = np.zeros(8, dtype=int)
9     height, width = pixels.shape
10
11    if (xm < 0):
12        xm = x
13    if (ym < 0):
14        ym = y
15    if (xp >= width):
16        xp = x
17    if (yp >= height):
18        yp = y
19
20    d[0] = pixels[ym, xm];      d[1] = pixels[ym, x];      d[2] = pixels[ym, xp]
21    d[3] = pixels[y,  xm];      d[4] = pixels[y,  xp];
22    d[5] = pixels[yp, xm];      d[6] = pixels[yp, x];      d[7] = pixels[yp, xp]
23
24    weight = np.sum(d)
25
26    return weight;
```

완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

```
1 img = cv2.imread("./images/couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 img2 = flattening(img)
```

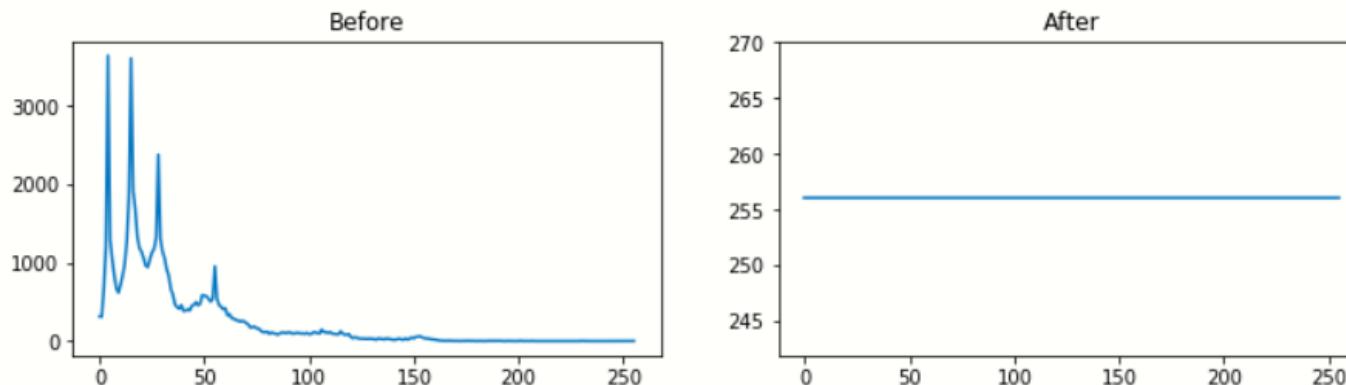
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 plt.figure(figsize=(10,5))
4 plt.subplot(121); plt.imshow(img, cmap="gray"); plt.title("Before")
5 plt.subplot(122); plt.imshow(img2, cmap="gray"); plt.title("After")
6 plt.show()
```



완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

```
1 hist1 = cv2.calcHist(images=[img], channels=[0], mask=None,  
2                         histSize=[256], ranges=[0,256]) # 원본 이미지  
3 hist2 = cv2.calcHist(images=[img2], channels=[0], mask=None,  
4                         histSize=[256], ranges=[0,256]) # 히스토그램 평탄화 이미지  
5  
6 plt.figure(figsize=(12,3))  
7 plt.subplot(121); plt.plot(range(256), hist1.flatten()); plt.title("Before")  
8 plt.subplot(122); plt.plot(range(256), hist2.flatten()); plt.title("After")  
9 plt.show()
```



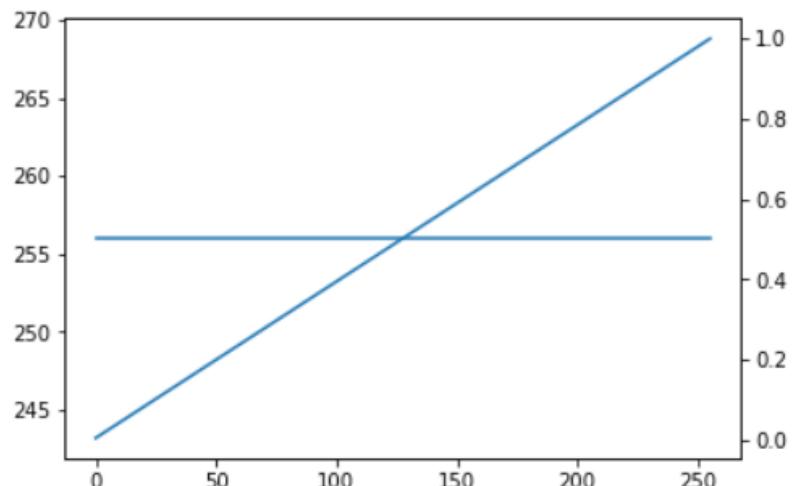
```
1 hist2.flatten()
```

```
array([256., 256., 256., 256., 256., 256., 256., 256., 256., 256., 256.,
```

완벽하게 평형판 히스토그램 균일화

5장. 화질 개선 / 3절. 화질 개선

```
1 img = cv2.imread("images/couple2.jpg", cv2.IMREAD_GRAYSCALE)
2 img2 = flatening(img)
3
4 hist2 = histogram(img2)
5 cumsum_hist = np.cumsum(hist2)
6 sum_hist = np.sum(hist2)
7 plt.plot(range(256), hist2)
8 ax = plt.twinx()
9 ax.plot(range(256), cumsum_hist/sum_hist)
10 plt.show()
```





6장. 특징 추출

특징을 찾는 방법에 대해 설명합니다.

- 1절. 윤곽선 검출과 특징 파라미터
- 2절. 화상의 라벨링
- 3절. 특징 파라미터를 이용한 추출

6장. 특징 추출



1절. 라벨링 특징 파라미터

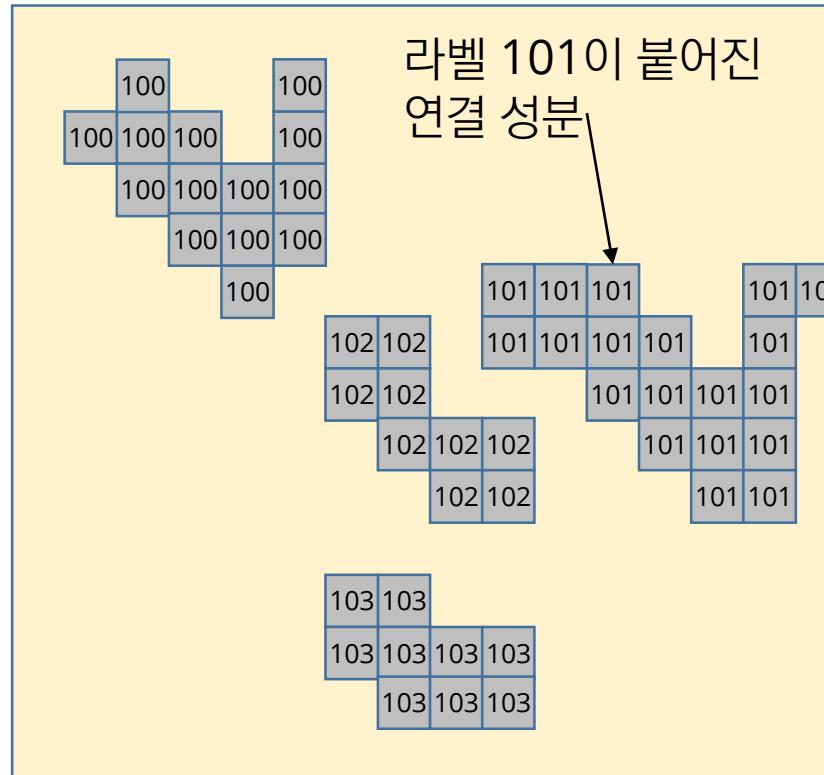
파이썬 OpenCV를 이용한
영상처리

저자 허진경

라벨링

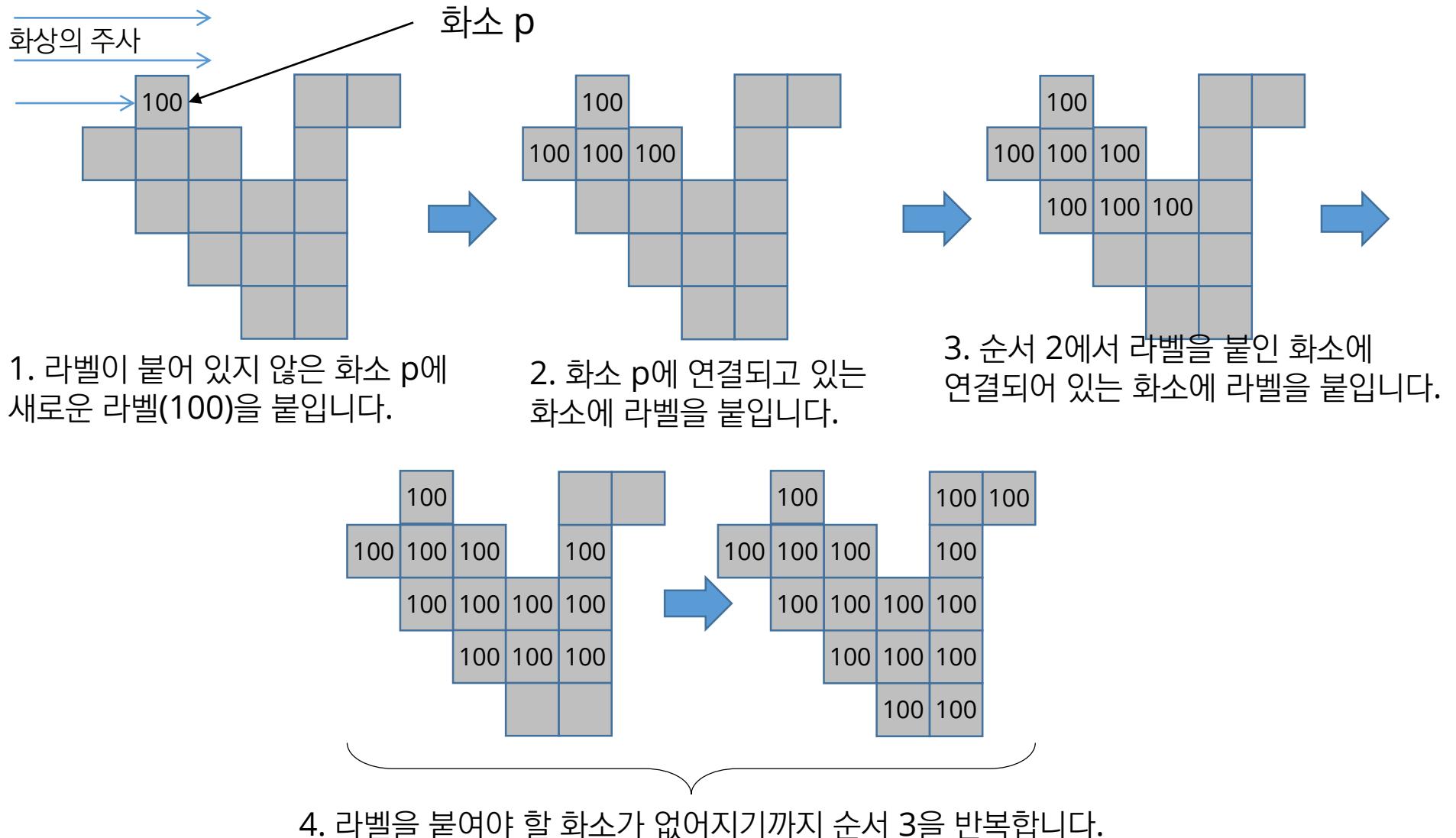
6장. 특징 추출 / 1절. 라벨링 특징 파라미터

연결되어 있는 모든 화소에 같은 번호를 붙이고 다른 연결 성분에는 다른 번호를 붙임



라벨링

6장. 특징 추출 / 1절. 라벨링 특징 파라미터



라벨링

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 def labelset(img, xs, ys, label):
2     height, width = img.shape
3     img[ys, xs] = label
4     while True:
5         cnt = 0
6         for y in range(1, height-1):
7             for x in range(1, width-1):
8                 if img[y, x] == label:
9                     if img[y, x+1] == 255:
10                        img[y, x+1] = label; cnt = cnt+1
11                     if img[y-1, x+1] == 255:
12                        img[y-1, x+1] = label; cnt = cnt+1
13                     if img[y-1, x] == 255:
14                        img[y-1, x] = label; cnt = cnt+1
15                     if img[y-1, x-1] == 255:
16                        img[y-1, x-1] = label; cnt = cnt+1
17                     if img[y, x-1] == 255:
18                        img[y, x-1] = label; cnt = cnt+1
19                     if img[y+1, x-1] == 255:
20                        img[y+1, x-1] = label; cnt = cnt+1
21                     if img[y+1, x] == 255:
22                        img[y+1, x] = label; cnt = cnt+1
23                     if img[y+1, x+1] == 255:
24                        img[y+1, x+1] = label; cnt = cnt+1
25         if cnt==0:
26             return (0, img)
27     return (1, img)
```

들여쓰기
주의하세요.

라벨링

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 def labeling(img_in, base=100):
2     height, width = img_in.shape
3     img_label = img_in.copy()
4
5     label = base # 베이스 값부터 라벨링 함
6
7     for y in range(1, height-1):
8         for x in range(1, width-1):
9             if img_label[y,x] == 255 :
10                 if label >= 255:
11                     print("Error! too many labels")
12                     return -1
13
14                 _, img_label = labelset(img_label, x, y, label)
15                 label = label + 1
16
17     cnt = label - base
18
19     return img_label, cnt
```

라벨링

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 img = cv2.imread('./images/shape.png', 0)
2 cv2.imshow('image', img)
3 cv2.waitKey(0)
4 cv2.destroyAllWindows()
```

```
1 img = cv2.imread('./images/shape.png', 0)
2 ret, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
3 labeled_img, cnt = labeling(bin_img)
4 cnt
```

4



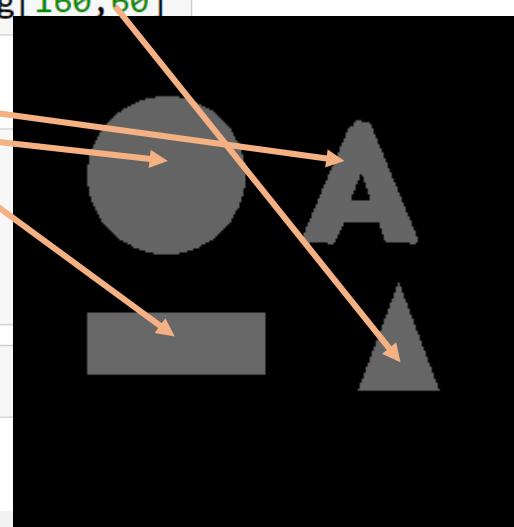
```
1 # 원, A, 세모, 네모 안의 임의 좌표
2 labeled_img[80, 80], labeled_img[69,173], labeled_img[160,190], labeled_img[160,60]
```

(1, 2, 3, 4)

```
1 # Labeled_img.shape
2 cv2.imshow('image', labeled_img)
3 cv2.waitKey(0)
4 cv2.destroyAllWindows()
```

```
1 cv2.imwrite('./images/shape_labeled.png',labeled_img)
```

True



cv2.connectedComponents()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 img = cv2.imread('./images/shape.png', 0)
2 ret, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
3 cnt, labeled_img = cv2.connectedComponents(bin_img)
```

```
1 print(labeled_img[80,80], labeled_img[69,173], end=" ")
2 print(labeled_img[160,190], labeled_img[160,60])
```

1 2 3 4

```
1 labeled_img.dtype
dtype('int32')
```

타입이 uint8이
아니고 int32임

```
1 labeled_img = labeled_img*50
```

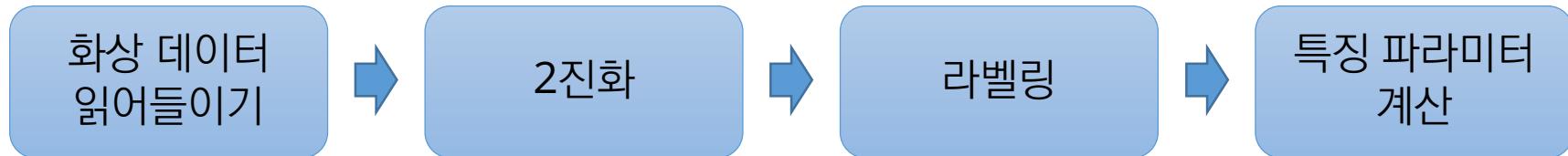
화소 1,2,3은 구분이
안되므로 50을 곱해서
0화소와 구분되도록 함

```
1 cv2.imshow('image', labeled_img.astype(np.uint8))
2 cv2.waitKey(0)
3 cv2.destroyAllWindows()
```



특징 파라미터 추출

6장. 특징 추출 / 1절. 라벨링 특징 파라미터



- 면적
- 주위 길이
- 원형도
- 중심 위치

면적과 중심

6장. 특징 추출 / 1절. 라벨링 특징 파라미터



면적 : 물체에 포함된 화소 수를 계산합니다.



중심 : 백색 화소의 위치 $(x_i, y_i) (i = 0, \dots, n - 1)$ 의 평균값을 계산합니다.

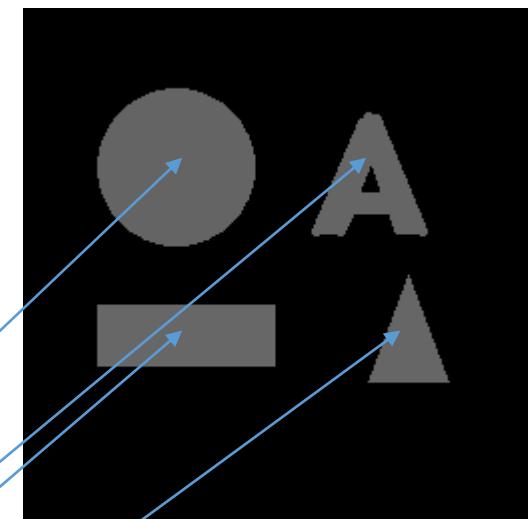
$$\left(\frac{1}{n} \sum_{i=0}^{n-1} x_i, \frac{1}{n} \sum_{i=0}^{n-1} y_i \right)$$

n은 화소의 수, 즉 면적을 의미합니다.

면적과 중심 위치

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 def calc_size(image_label, label):
2     tx, ty = 0, 0
3     cx, cy = 0, 0 # 반환할 중심점
4     total = 0
5     height, width = image_label.shape
6     for y in range(height):
7         for x in range(width):
8             if image_label[y,x]==label:
9                 tx = tx + x
10                ty = ty + y
11                total = total + 1
12
13    if total==0:
14        return 0
15    cx = int(tx/total)
16    cy = int(ty/total)
17    return (cx, cy, total)
```



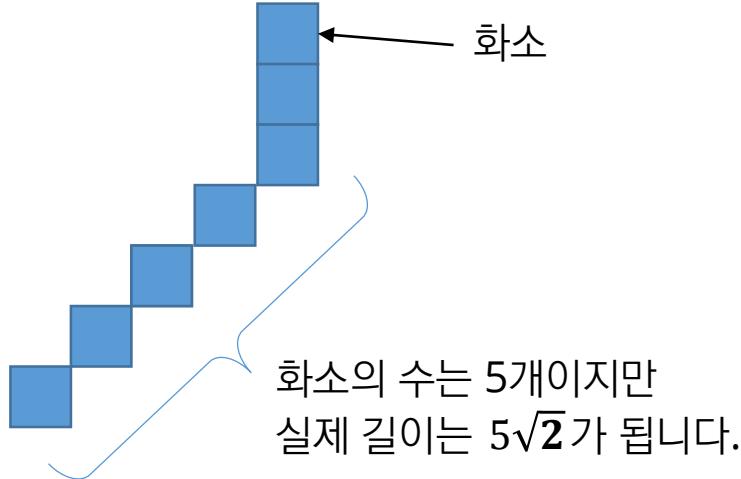
```
1 img = cv2.imread('./images/shape.png', 0)
2 ret, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
3 labeled_img, cnt = labeling(bin_img, base=100)
4 for i in range(cnt):
5     cx, cy, area = calc_size(labeled_img, i+100)
6     print(cx, cy, area)
```

76 79 4880
172 87 1797
192 168 1108
81 163 2759

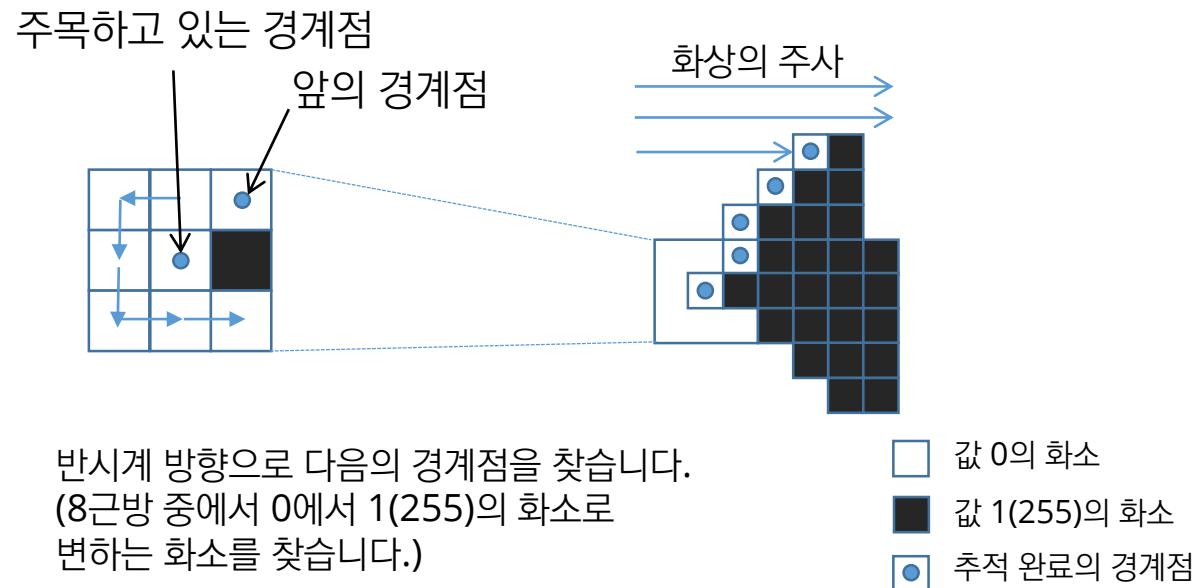
주위 길이

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

원주 길이 계산 : 윤곽선의 화소수를 계산하며, 경사 방향은 $\sqrt{2}$ 배하여 계산해야 합니다.



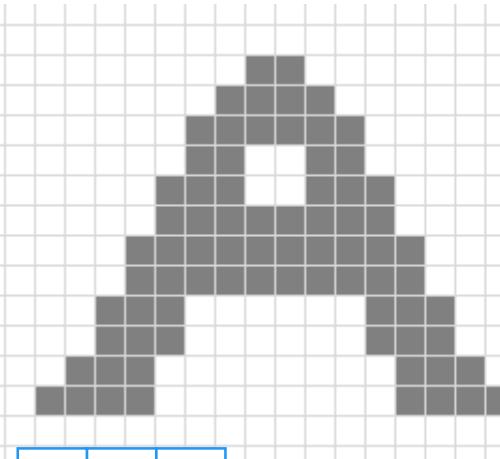
주위 길이 계산



경계부의 추적

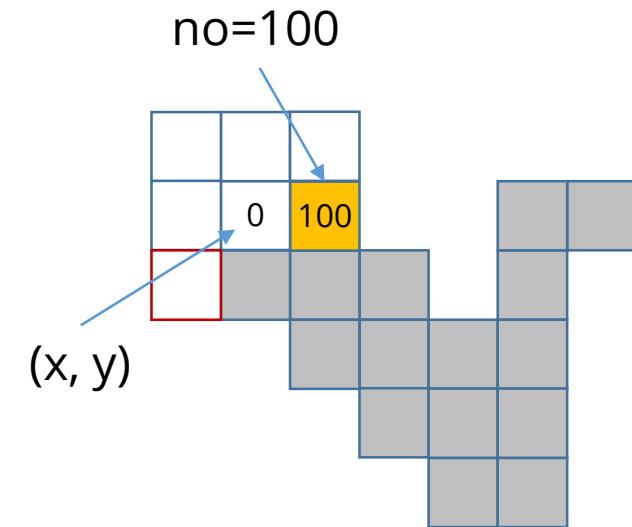
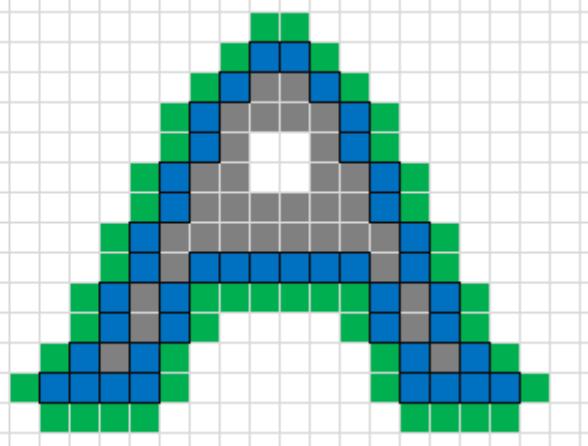
주위 길이 계산 코드를 위해서...

6장. 특징 추출 / 1절. 라벨링 특징 파라미터



3	2	1
4		0
5	6	7

시작 : vec=5



vec=3	vec=4	vec=5	vec=6	vec=7	vec=0	vec=1	vec=2
3 2 1	3 2 1	3 2 1	3 2 1	3 2 1	3 2 1	3 2 1	3 2 1
4	4	4	4	4	4	4	4
5 6 7	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7

new_vec = 0 new_vec = 1 new_vec = 2 new_vec = 3 new_vec = 4 new_vec = 5 new_vec = 6 new_vec = 7
 x=x+1 x=x+1 x=x x=x-1 x=x-1 x=x-1 x=x x=x+1
 y=y y=y-1 y=y-1 y=y-1 y=y y=y+1 y=y+1 y=y+1

주위 길이

6장. 특징 추출 / 1절. 라벨링 특징 파라미터



원형도

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

 면적, 원주 길이를 토대로 형상의 복잡한 정도를 측정하는 특징량입니다.

$$e = 4\pi * \text{면적} / (2 * \text{원주길이})$$

 원의 경우, 원주 길이 $2\pi r$, 면적 πr^2 이므로, $e=1.00$ 이 됩니다.

 도형이 복잡할수록 작은 값이 됩니다.

주위 길이 계산 1/3

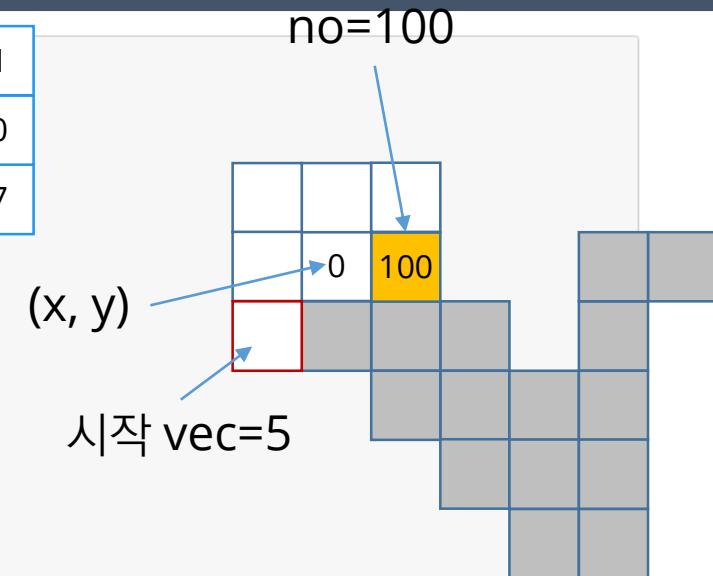
6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 def trace(image_label, xs, ys):
2     length = 0
3     x, y = xs, ys
4     no = image_label[y, x+1]
5     vec = 5
6
7     while True:
8         if (x==xs) & (y==ys) & (length!=0):
9             return (length, image_label)
10        image_label[y, x] = 255
11
12        if vec==3:
13            if (image_label[y ,x+1]!=no) & (image_label[y-1,x+1]==no):
14                x=x+1; y=y ; length=length+1; vec=0; continue
15            else:
16                vec=4
17        if vec==4:
18            if (image_label[y-1,x+1]!=no) & (image_label[y-1,x ]==no):
19                x=x+1; y=y-1; length=length+np.sqrt(2); vec=1; continue
20            else:
21                vec=5
22        if vec==5:
```

파이썬은 switch~case 문이 없으므로 else 블록을 이용해서 다음 if 문장이 실행되도록 해야 합니다.

이후의 모든 문장을 건너뛰고 다음 while 문을 실행합니다.

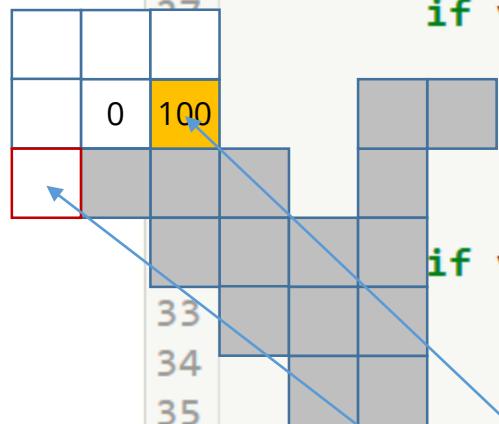
3	2	1
4		0
5	6	7



주위 길이 계산 2/3

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
22     if vec==5:
23         if (image_label[y-1,x ]!=no) & (image_label[y-1,x-1]==no):
24             x=x ; y=y-1; length=length+1;                     vec=2; continue
25         else:
26             vec=6
27
28     if vec==6:
29         if (image_label[y-1,x-1]!=no) & (image_label[y ,x-1]==no):
30             x=x-1; y=y-1; length=length+np.sqrt(2); vec=3; continue
31         else:
32             vec=7
33
34     if vec==7:
35         if (image_label[y ,x-1]!=no) & (image_label[y+1,x-1]==no):
36             x=x-1; y=y ; length=length+1;                     vec=4; continue
37         else:
38             vec=0
39
40     if vec==0:
41         if (image_label[y+1,x-1]!=no) & (image_label[y+1,x ]==no):
42             x=x-1; y=y+1; length=length+np.sqrt(2); vec=5; continue
43         else:
44             vec=1
45
46     if vec==1:
47         if (image_label[y+1,x ]!=no) & (image_label[y+1,x+1]==no):
```



주위 길이 계산 3/3

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
42 if vec==1:  
43     if (image_label[y+1,x ]!=no) & (image_label[y+1,x+1]==no):  
44         x=x ; y=y+1; length=length+1;             vec=6; continue  
45     else:  
46         vec=2  
47 if vec==2:  
48     if (image_label[y+1,x+1]!=no) & (image_label[y ,x+1]==no):  
49         x=x+1; y=y+1; length=length+np.sqrt(2); vec=7; continue  
50     else:  
51         vec=3  
52
```

각 레이블을 갖는 도형들의 주위 길이 계산

```
1 def calc_length(image_label, label):  
2     height, width = image_label.shape  
3     for y in range(height):  
4         for x in range(width):  
5             if image_label[y,x] == label:  
6                 length, image_label = trace(image_label, x-1, y)  
7                 return length, image_label
```

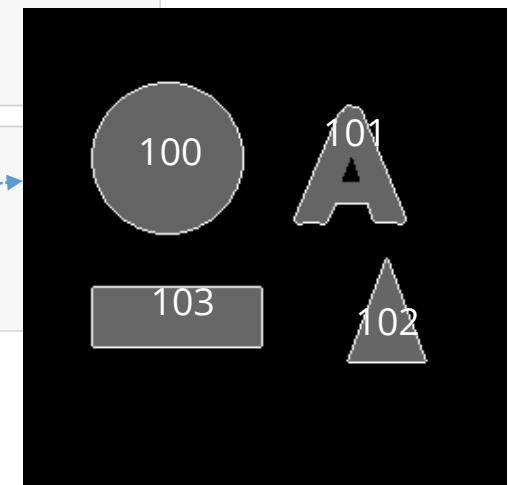
중심점, 면적, 둘레길이, 원형도

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 def features(image_label, cnt):
2     areas = []; centers = []; lengths = []; ratios = []
3     featured_img = np.zeros(image_label.shape, dtype=np.uint8)
4     for i in range(cnt):
5         cx, cy, area = calc_size(image_label, i+100)
6         centers.append((cx, cy)); areas.append(area)
7         print(i+100, cx, cy, area, end=" ")
8
9         length, featured_img = calc_length(image_label, i+100)
10        lengths.append(length)
11        ratio = 4*np.pi*area / (length*length)
12        ratios.append(ratio)
13        print(length, ratio)
14
15    return (areas, centers, lengths, ratios), featured_img
```

```
1 img = cv2.imread('./images/shape.png', 0)
2 ret, bin_img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
3 labeled_img, cnt = labeling(bin_img, 100)
4 feat, featured_img = features(labeled_img, cnt)
```

```
100 76 79 4880 263.7645019878176 0.8814486001080429
101 172 87 1797 226.99494936611694 0.4382536609116099
102 192 168 1108 168.2253967444163 0.49200158512447256
103 81 163 2759 241.6568542494924 0.5936949146124189
```



Contour

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

동일한 색 또는 동일한 강도를 가지고 있는 영역의 경계선을 연결한 선

대상의 외형을 파악하는데 유용하게 사용이 됩니다.



- ▶ 정확도를 높히기 위해서 Binary Image를 사용합니다. `threshold`나 `canny edge`를 선처리로 수행합니다.
 - ▶ `cv2.findContours()` 함수는 원본 이미지를 직접 수정하기 때문에, 원본 이미지를 보존 하려면 `Copy`해서 사용해야 합니다.
 - ▶ OpenCV에서는 `contours`를 찾는 것은 검은색 배경에서 하얀색 대상을 찾는 것과 비슷합니다. 그래서 대상은 흰색, 배경은 검은색으로 해야 합니다.
- ▶ OpenCV에서 `contours`를 찾고, 그리기 위해서 아래 2개의 함수를 사용합니다.

`cv2.findContours()`

`cv2.drawContours()`

cv2.findContours()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

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

- ▶ 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, hierarchy 관계를 구성하지 않음.

cv2.RETR_CCOMP : 모든 contours line을 찾으며, hierarchy관계는 2-level로 구성함.

cv2.RETR_TREE : 모든 contours line을 찾으며, 모든 hierarchy관계를 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

cv2.drawContours()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

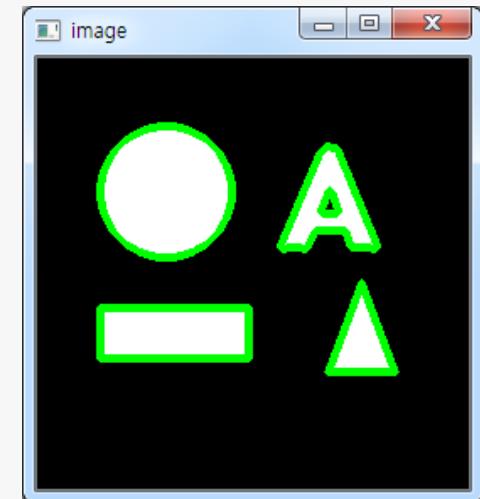
cv2.drawContours(image, contours, contourIdx, color
[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]]) -> dst

- ▶ image – 원본 이미지
- contours – contours 정보.
- contourIdx – contours list type에서 몇 번째 contours line을 그릴 것인지. -1 이면 전체
- color – contours line color
- thickness – contours line의 두께. 음수이면 contours line의 내부를 채움.

cv2.drawContours()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('images/shape.png')
5 imggray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6
7 ret, bin_img = cv2.threshold(imggray, 127, 255, 0)
8
9 contours, hierarchy = cv2.findContours(bin_img,
10                         cv2.RETR_TREE,
11                         cv2.CHAIN_APPROX_SIMPLE)
12
13 image = cv2.drawContours(img, contours, -1, (0,255,0), 3)
14
15 cv2.imshow('image', image)
16 cv2.waitKey(0)
17 cv2.destroyAllWindows()
```



cv2.moments()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

Image Moment는 대상을 구분할 수 있는 특징(Area, Perimeter, 중심점)을 의미

대상을 구분한 후, 다른 대상과 구분하기 위해 대상을 설명(describe)하는 자료로 사용

`cv2.moments(array, binaryImage) → retval`

- ▶ array – 8비트 단일채널 이미지 또는 1xN 또는 Nx1 2D 점들입니다.
- ▶ binaryImage – True이면 0이 아닌 모든 이미지 픽셀은 1로 처리됩니다. 매개변수는 이미지에만 사용됩니다.
- ▶ 반환값 retval – dict_item 유형의 모멘트 정보를 반환합니다.

cv2.moments()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 import numpy as np
4
5 img = cv2.imread('images/rectangle.jpg')
6 imggray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7 ret, bin_img = cv2.threshold(imggray, 127, 255, 0)
8
9 contours, hierarchy = cv2.findContours(bin_img,
10                         cv2.RETR_TREE,
11                         cv2.CHAIN_APPROX_SIMPLE)
12
13 cnt = contours[0] # 첫번째 contours의 moment 특징 추출
14 M = cv2.moments(cnt)
15
16 print(M.items())
```

```
dict_items([('m00', 16728.0), ('m10', 2308464.0), ('m01', 1555704.0), ('m20', 376580736.0), ('m11', 214687152.0), ('m02', 154053728.0), ('m30', 67979647872.0), ('m21', 35022008448.0), ('m12', 21259414464.0), ('m03', 16070422320.0), ('mu20', 58012704.0), ('mu11', 0.0), ('mu02', 9373256.0), ('mu30', 0.0), ('mu21', 0.0), ('mu12', 0.0), ('mu03', 0.0), ('nu20', 0.20731707317073172), ('nu11', 0.0), ('nu02', 0.0334967320261438), ('nu30', 0.0), ('nu21', 0.0), ('nu12', 0.0), ('nu03', 0.0)])
```

cv2.moments()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 print(M.items())
```

```
dict_items([('m00', 16728.0), ('m10', 2308464.0), ('m01', 1555704.0), ('m20', 376580736.0), ('m11', 214687152.0), ('m02', 154053728.0), ('m30', 67979647872.0), ('m21', 35022008448.0), ('m12', 21259414464.0), ('m03', 16070422320.0), ('mu20', 58012704.0), ('mu11', 0.0), ('mu02', 9373256.0), ('mu30', 0.0), ('mu21', 0.0), ('mu12', 0.0), ('mu03', 0.0), ('nu20', 0.20731707317073172), ('nu11', 0.0), ('nu02', 0.0334967320261438), ('nu30', 0.0), ('nu21', 0.0), ('nu12', 0.0), ('nu03', 0.0)])
```

```
1 cv2.contourArea(cnt) # 면적, m00과 같음
```

16728.0

```
1 cv2.arcLength(cnt, True) # 폐곡선 도형을 만들어 둘레 길이 구함
```

572.0

```
1 cv2.arcLength(cnt, False) # 시작점과 끝점을 연결하지 않고 둘레 길이 구함
```

368.0

cv2.contourArea()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

cv2.contourArea(contour, oriented) → retval

- ▶ contour – 컨투어 객체입니다.
- ▶ oriented – True이면 함수는 윤곽선 방향(시계 방향 또는 시계 반대 방향)에 따라 부호 있는 영역 값을 반환합니다. 이 기능을 사용하면 영역의 부호를 사용하여 윤곽선의 방향을 결정할 수 있습니다. 기본값은 False이며 절댓값이 리턴됨을 의미합니다.
- ▶ 반환값 retval – 컨투어 영역입니다. 모멘트의 결과인 dict_items의 m00과 같습니다.

cv2.arcLength()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

cv2.arcLength(curve, closed) → retval

- ▶ curve – 2D 점의 입력 벡터입니다.
- ▶ close – True이면 폐곡선 도형의 둘레 길이를 구하고, False 이면 시작점과 끝 점을 연결하지 않고 둘레 길이를 구합니다.
- ▶ 반환값 retval – 둘레 길이입니다.

컨투어 속성

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

 Aspect Ratio : Contours Line의 가로 세로 비율 속성입니다.

- ▶ AspectRatio=Width/Height

cv2.boundingRect() 함수를 이용하여 가로/세로 크기를 구한 후에 사용합니다.

```
x, y, w, h = cv2.boundingRect(cont)
```

```
aspect_ratio = float(w)/h
```

 Extend : Contour Line을 포함하는 사각형 면적대비 Contour의 면적 비율입니다.

- ▶ Extend=ObjectArea / BoundingRectagleArea

```
area = cv2.contourArea(cont) # Contour Line의 면적
```

```
x, y, w, h = cv2.boundingRect(cont)
```

```
rect_area = w * h # 사각형 면적
```

```
extend = float(area) / rect_area
```

컨투어 속성

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

 Solidity : Solidity Ratio(고형비)는 Convex hull 면적 대비 Contour의 면적 비율입니다.

- ▶ $\text{Solidity} = \text{ContourArea} / \text{ConvexHullArea}$

```
area = cv2.contourArea(cont) # Contour Line 면적
```

```
hull = cv2.convexHull(cont) # Convex hull line
```

```
hull_area = cv2.contourArea(hull) # Convex hull 면적
```

```
solidity = float(area) / hull_area
```

 Extream Points : Contour Line의 좌우상하의 끝점을 찾는 방법입니다.

- ▶ `leftmost = tuple(cont[cont[:, :, 0].argmin()][0])`
- ▶ `rightmost = tuple(cont[cont[:, :, 0].argmax()][0])`
- ▶ `topmost = tuple(cont[cont[:, :, 1].argmin()][0])`
- ▶ `bottommost = tuple(cont[cont[:, :, 1].argmax()][0])`

cv2.approxPolyDP()

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

cv2.findContours() 함수에 의해서 찾은 contours line은 각각의 contours point를 가지고 있습니다. 이 Point를 연결하여 Line을 그리게 됩니다. 이때 이 point의 수를 줄여 근사한 line을 그릴 때 사용되는 방법입니다.

Point의 수를 줄이는데 사용되는 방식은 Douglas-Peucker algorithm 입니다.

근사치를 찾는데 사용되는 함수는 cv2.approxPolyDP() 입니다.

cv2.approxPolyDP(curve, epsilon, closed[, approxCurve]) → approxCurve

- ▶ curve – contours point array

epsilon – original cuve와 근사치의 최대거리. 최대거리가 클 수록 더 먼 곳의 Point까지 고려하기 때문에 Point수가 줄어듬.

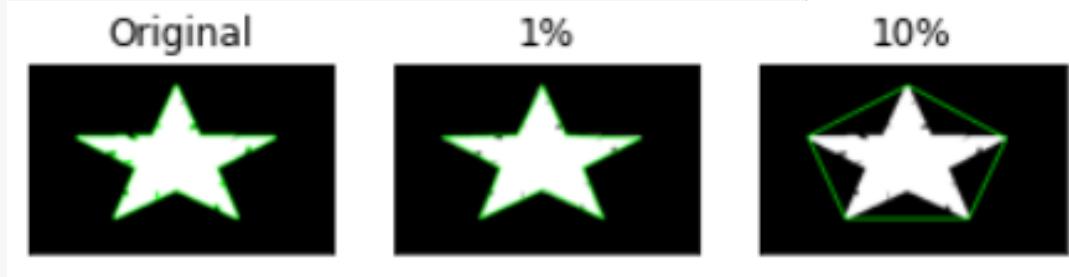
closed – 폐곡선 여부

- ▶ 반환 값은 근사치가 적용된 contours point array

Contour Approximation

6장. 특징 추출 / 1절. 라벨링 특징 파라미터

```
1 import cv2
2 import numpy as np
3 from cv2 import imread('star.png')
4 img = cv2.imread('images/bad_rectangle.png')
5 img1 = img.copy()
6 img2 = img.copy()
7
8 imggray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9 ret, bin_img = cv2.threshold(imggray, 127, 255, 0)
10
11 contours, hierarchy = cv2.findContours(bin_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
12 cnt = contours[0]
13
14 epsilon1 = 0.01*cv2.arcLength(cnt, True)
15 epsilon2 = 0.1*cv2.arcLength(cnt, True) # 용하는 숫자가 커질 수록 Point의 갯수는 감소
16
17 approx1 = cv2.approxPolyDP(cnt, epsilon1, True)
18 approx2 = cv2.approxPolyDP(cnt, epsilon2, True)
19
20 cv2.drawContours(img, [cnt], 0, (0, 255, 0), 3)
21 cv2.drawContours(img1, [approx1], 0, (0, 255, 0), 3)
22 cv2.drawContours(img2, [approx2], 0, (0, 255, 0), 3)
23
24 titles = ['Original', '1%', '10%']
25 images = [img, img1, img2]
26
27 for i in range(3):
28     plt.subplot(1, 3, i+1), plt.title(titles[i]), plt.imshow(images[i])
29     plt.xticks([]), plt.yticks([])
30
31
32 plt.show()
```



6장. 특징 추출



2절. 객체 탐지

파이썬 OpenCV를 이용한
영상처리

저자 허진경

컨투어를 포함하는 사각형 찾기

6장. 특징 추출 / 2절. 객체 탐지

Contours Line을 둘러싸는 사각형을 그리는 방법은 2가지가 있습니다.

Straight Bounding Rectangle : 대상의 Rotation은 무시한 사각형 모양입니다.

```
# Straight Rectangle
x, y, w, h = cv2.boundingRect(cnt)
img1 = cv2.rectangle(img1,(x,y),(x+w, y+h),(0,255,0), 3) # green
```

Rotated Rectangle : 대상을 모두 포함하면서, 최소한의 영역을 차지하는 사각형 모양입니다.

```
# Rotated Rectangle
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
img1 = cv2.drawContours(img1, [box], 0, (0,0,255), 3) # blue
```

컨투어를 포함하는 원 찾기

6장. 특징 추출 / 2절. 객체 탐지

Contours line을 완전히 포함하는 원 중 가장 작은 원을 그릴 수 있습니다.

```
# Minimum Enclosing Circle
(x,y), radius = cv2.minEnclosingCircle(cnt)
center = (int(x), int(y))
radius = int(radius)
img1 = cv2.circle(img1, center, radius, (255,255,0), 3) # yellow
```

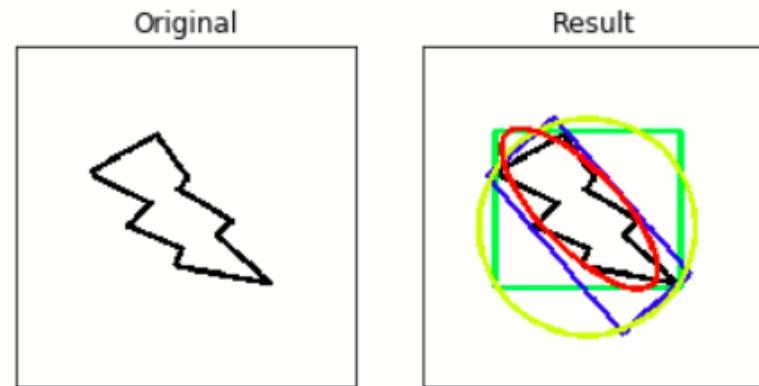
Contours Line을 둘러싸는 타원을 그릴 수 있습니다.

```
# Fitting an Ellipse
ellipse = cv2.fitEllipse(cnt)
img1 = cv2.ellipse(img1, ellipse, (255,0,0), 3) #red
```

Rectangle, Circle, Ellipse

6장. 특징 추출 / 2절. 객체 탐지

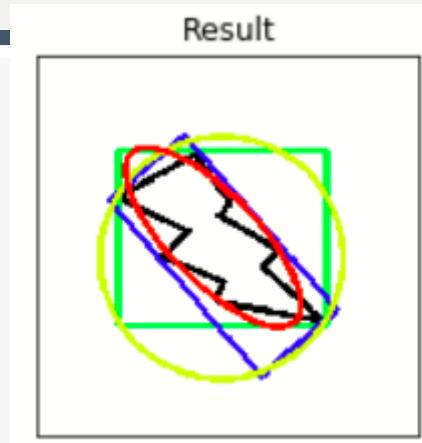
```
1 #-*- coding:utf-8 -*-
2 import cv2
3 import numpy as np
4 from matplotlib import pyplot as plt
5
6 img = cv2.imread('images/lightning.png')
7 img1 = img.copy()
8
9 imggray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10 ret, bin_img = cv2.threshold(imggray, 127, 255, 0)
11
12 contours, hierarchy = cv2.findContours(bin_img,
13                                         cv2.RETR_TREE,
14                                         cv2.CHAIN_APPROX_SIMPLE)
15
16 cnt = contours[1]
17
18 # Straight Rectangle
19 x, y, w, h = cv2.boundingRect(cnt)
20 img1 = cv2.rectangle(img1,(x,y),(x+w, y+h),(0,255,0), 3) # green
21
```



Rectangle, Circle, Ellipse

6장. 특징 추출 / 2절. 객체 탐지

```
22 # Rotated Rectangle
23 rect = cv2.minAreaRect(cnt)
24 box = cv2.boxPoints(rect)
25 box = np.int0(box)
26 img1 = cv2.drawContours(img1, [box], 0, (0,0,255), 3) # blue
27
28 # Minimum Enclosing Circle
29 (x,y), radius = cv2.minEnclosingCircle(cnt)
30 center = (int(x), int(y))
31 radius = int(radius)
32 img1 = cv2.circle(img1, center, radius,(255,255,0),3) # yellow
33
34 # Fitting an Ellipse
35 ellipse = cv2.fitEllipse(cnt)
36 img1 = cv2.ellipse(img1, ellipse,(255,0,0),3) #red
37
38 titles = ['Original','Result']
39 images = [img, img1]
40
41 for i in range(2):
42     plt.subplot(1,2,i+1), plt.title(titles[i]), plt.imshow(images[i])
43     plt.xticks([]), plt.yticks([])
44 plt.show()
```



Convex Hull

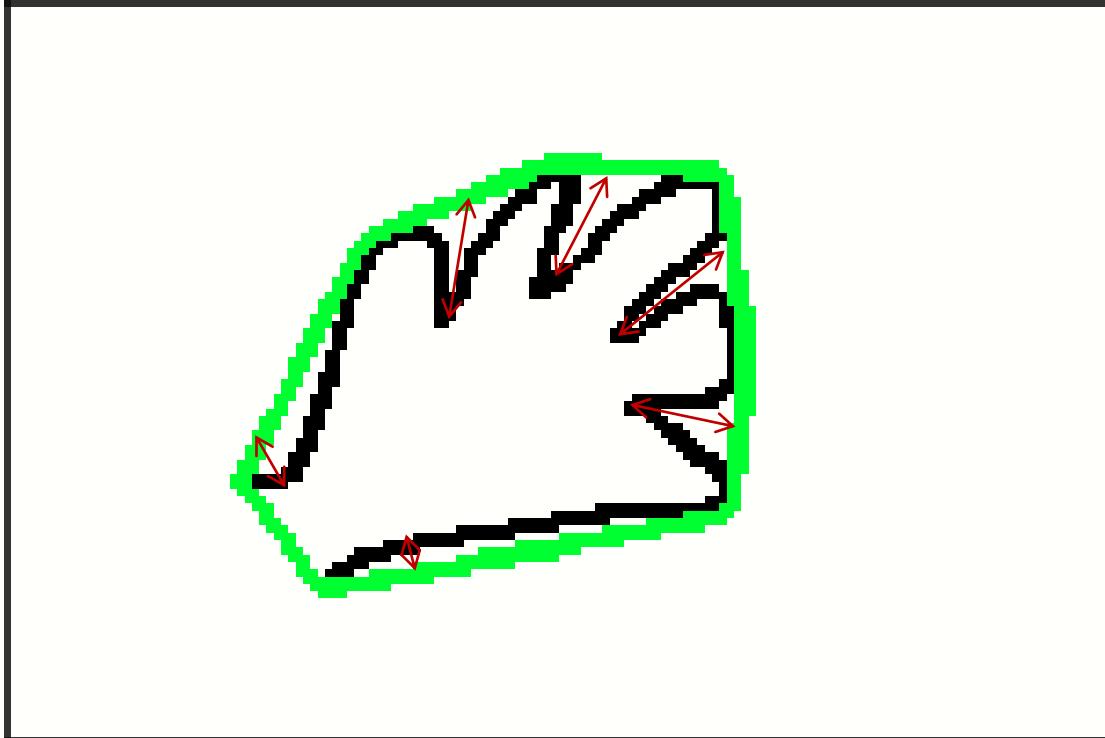
6장. 특징 추출 / 2절. 객체 탐지



contours point를 모두 포함하는 불록한 외관선을 의미합니다.



Contour Approximation과 유사한 결과지만, 방법은 전혀 다릅니다.

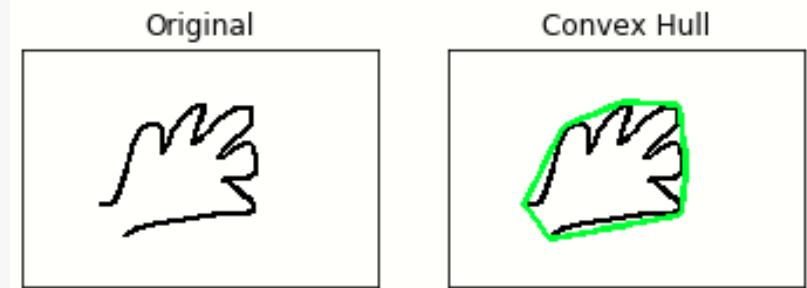


그림에서 초록 선이 Convex Hull을 나타내고 화살표의 차이가 convexity defect라고 합니다. convexity defect는 contours와 hull과의 최대차이를 나타냅니다.

Convex Hull

6장. 특징 추출 / 2절. 객체 탐지

```
1 #-*- coding:utf-8 -*-
2 import cv2
3 import numpy as np
4 from matplotlib import pyplot as plt
5
6 img = cv2.imread('images/hand.png')
7 img1 = img.copy()
8
9 imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10 ret, bin_img = cv2.threshold(imgray, 127, 255, 0)
11
12 contours, hierarchy = cv2.findContours(bin_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
13
14 contour = contours[1] # 10/ 손모양 주변의 contour
15 hull = cv2.convexHull(contour)
16
17 cv2.drawContours(img1, [hull], 0, (0,255,0), 3)
18
19 titles = ['Original', 'Convex Hull']
20 images = [img, img1]
21
22 for i in range(2):
23     plt.subplot(1,2,i+1), plt.title(titles[i]), plt.imshow(images[i])
24     plt.xticks([]), plt.yticks([])
25
26 plt.show()
```



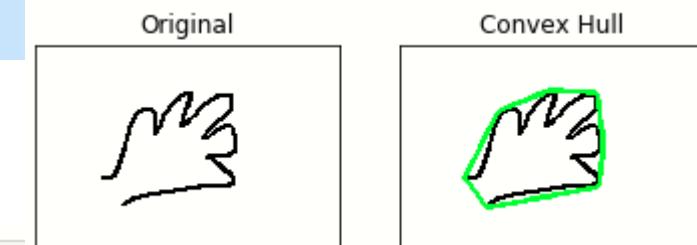
Checking Convexity

6장. 특징 추출 / 2절. 객체 탐지

cv2.isContourConvex() 함수는 contour가 convex인지 아닌지 판단하여 True 또는 False를 Return합니다.

convex란 contour line이 볼록하거나 최소한 평평한 것을 의미합니다.

앞 예제에는 2개의 contour가 있는데, 첫번째는 이미지의 전체 외곽선(사각형)이고 두번째는 손 모양의 contour line입니다.



```
1 cv2.isContourConvex(contours[0]) # 외곽선 contour Line
```

True

```
1 cv2.isContourConvex(contours[1]) # 손 모양 contour Line
```

False

6장. 특징 추출



3절. OpenCV와 머신러닝을
이용한 필기체 숫자 인식

파이썬 OpenCV를 이용한
영상처리

저자 허진경

손글씨 숫자 인식하기

6장. 특징 추출



OpenCV의 손글씨 숫자 데이터(숫자별로 20x20이미지 500개씩)

숫자이미지 분류하기

6장. 특징 추출

```
1 import cv2  
2 import numpy as np  
3  
4 img = cv2.imread('images/digits.png', cv2.IMREAD_GRAYSCALE)  
5 print(img.shape)
```

(1000, 2000)

```
1 cv2.imshow("number", img)  
2 cv2.waitKey(0)  
3 cv2.destroyAllWindows()
```

```
1 cells = [np.hsplit(row, 100) for row in np.vsplit(img, 50)]  
2 x = np.array(cells)
```

```
1 x.shape
```

(50, 100, 20, 20)

숫자이미지 분류하기

6장. 특징 추출

x[0,0] # 숫자 0

머신러닝 분류 모형

6장. 특징 추출

```
1 # x[0,0] # 숫자 0
```

```
1 X = x[:, :].reshape(-1, 400).astype(np.float32)
2 y = np.repeat(np.arange(10), 500)
```

훈련데이터셋과 학습 데이터셋으로 분리

```
1 from sklearn.model_selection import train_test_split
2 train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3)
```

```
1 from sklearn.ensemble import RandomForestClassifier
2 model = RandomForestClassifier()
3 model.fit(train_X, train_y)
4 print(model.score(test_X, test_y))
```

검증 데이터셋을 이용해서 정확도 확인

0.9313333333333333

모델 저장하고 불러오기

6장. 특징 추출

 pickle을 이용해서 예측 모형을 파일에 저장하고 불러올 수 있습니다.

```
1 import pickle  
2 with open("number.model", "wb") as f:  
3     pickle.dump(model, f)
```

피클링

```
1 import pickle  
2 with open("number.model", "rb") as f:  
3     model = pickle.load(f)
```

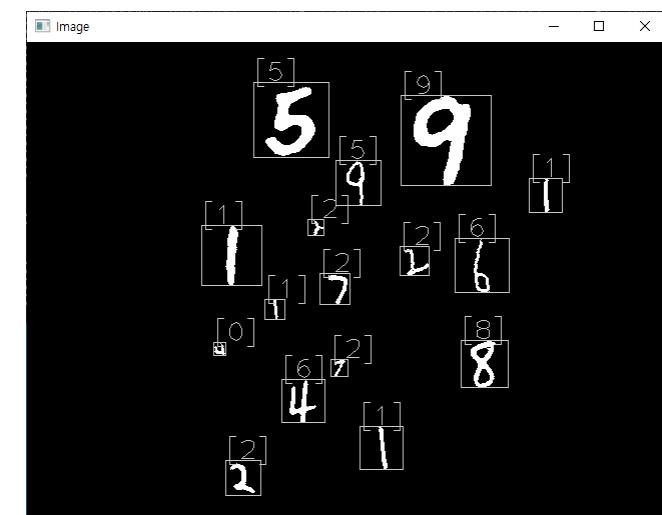
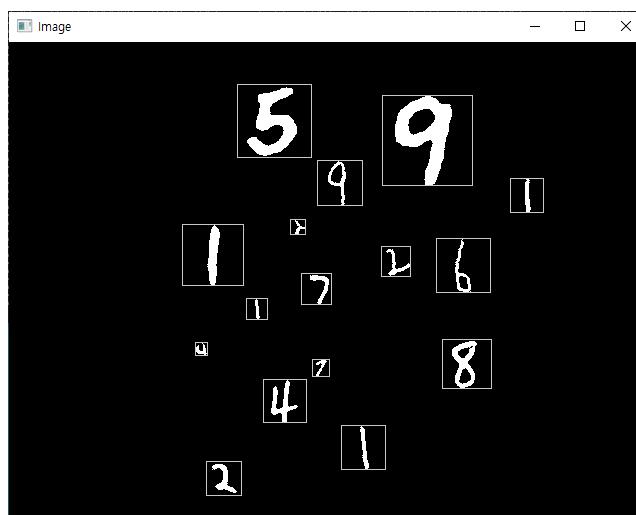
언피클링

카메라를 이용한 ML기반 필기체 숫자 인식하기

6장. 특징 추출



1. Color 프레임 읽기
2. 회색조 이미지 변환
3. 이진화(반전)
4. 컨투어 탐색
5. 예측



카메라를 이용한 ML기반 필기체 숫자 인식하기

6장. 특징 추출

```
1 import cv2
2 cap = cv2.VideoCapture(0)
3 if cap.isOpened():
4     while True:
5         ret, img = cap.read()
6         if ret:
7             g_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)           컬러 이미지를 회색조 이미지로 변환
8             thr, bin_img = cv2.threshold(g_img, 110, 255, cv2.THRESH_BINARY_INV)    이진화(반전) 처리
9             contours, hierarchy = cv2.findContours(bin_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
10            try:
11                for i in range(len(contours)):
12                    contour = contours[i]
13                    (x,y), radius = cv2.minEnclosingCircle(contour)      컨투어 찾기
14                    if radius > 3:
15                        xs, xe = int(x-radius), int(x+radius)
16                        ys, ye = int(y-radius), int(y+radius)
17                        cv2.rectangle(bin_img, (xs,ys), (xe,ye), (200,0,0), 1)
18                        roi = bin_img[ys:ye, xs:xe]
19                        dst = cv2.resize(roi, dsize=(50, 50), interpolation=cv2.INTER_AREA)
20                        dst = cv2.resize(dst, dsize=(16, 16), interpolation=cv2.INTER_AREA)
21                        A = np.zeros((20,20))
22                        A[2:-2,2:-2] = dst[:, :]                           왜? 이 코드가 필요할까요?
23                        A = A.reshape(-1,400)
24                        num = model.predict(A)
25                        cv2.putText(bin_img, str(num), (xs, ys), cv2.FONT_HERSHEY_PLAIN, 2, (200,0,0))
26            except Exception as e:
27                print(e)
28                pass
```

카메라를 이용한 ML기반 필기체 숫자 인식하기

6장. 특징 추출

```
26         except Exception as e:  
27             print(e)  
28             pass  
29             cv2.imshow("Image", bin_img)  
30             if cv2.waitKey(1)&0xFF == 27: # ESC  
31                 break  
32         else:  
33             print("No Frame")  
34             break  
35     else :  
36         print("Camera not opened")  
37  
38 cap.release()  
39 cv2.destroyAllWindows()
```

숫자의 크기, 굵기 등에 민감하고, 전체적으로 인식율이 떨어집니다. test 데이터셋을 이용해서 평가한 평가 점수를 너무 신뢰하면 안됩니다. test 데이터셋은 train 데이터셋처럼 숫자의 크기와 굵기가 대부분 일정하기 때문에 정확도가 높게 나오는 것입니다. 실제 여러분이 직접 숫자를 썼을 경우에는 굵기, 크기 등이 다르므로 예측 정확도가 현저하게 떨어집니다.

