CHAPTER 6
화소 처리

김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

6. 화소 처리

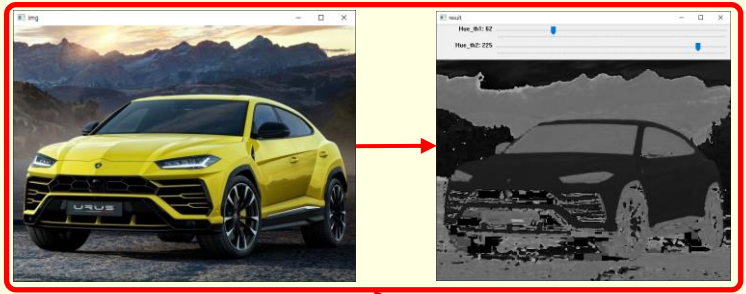
■ 6.4 컬러 공간 변환

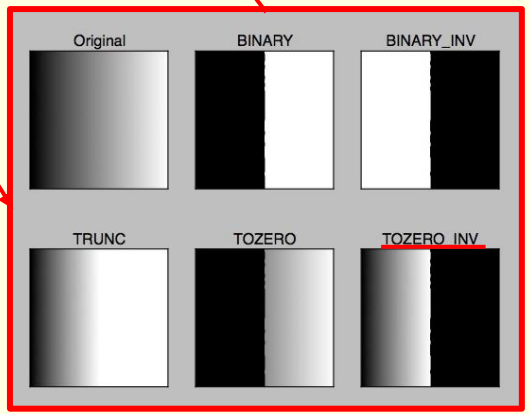
◆ 6.4.5 Hue 채널을 이용한 객체 검출

```
import numpy as np, cv2

img = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)
HSV_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hue_img = np.copy(HSV_img)[:, :, 0] # 색상 채널 컬러만 복사
th = [50, 100]

def onThreshold(value):
    th[0] = cv2.getTrackbarPos("Hue_th1", "result")
    th[1] = cv2.getTrackbarPos("Hue_th2", "result")

    _, result = cv2.threshold(hue_img, th[1], 255, cv2.THRESH_TOZERO_INV)
    cv2.imshow("result", result)


cv2.namedWindow("result")
cv2.createTrackbar("Hue_th1", "result", th[0], 255, onThreshold)
cv2.createTrackbar("Hue_th2", "result", th[1], 255, onThreshold)
cv2.imshow("img", img)
cv2.waitKey(0)
```

이진화

HSV
H = Hue(색상)
S = Saturation(채도)
V = Value(명도)

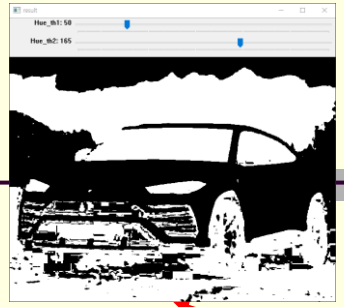


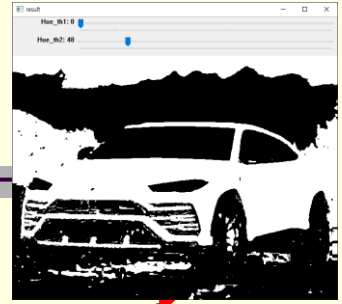


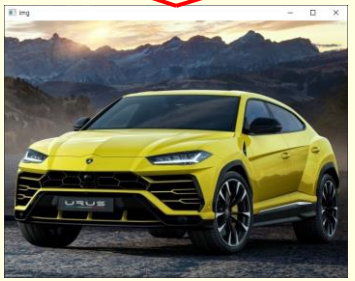
6. 화소 처리

■ 6.4 컬러 공간 변환

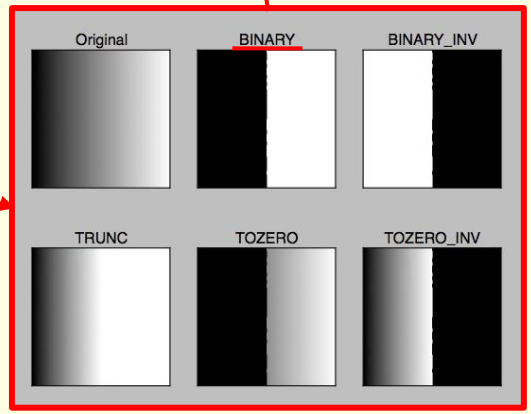
◆ 6.4.5 Hue 채널을 이용한 객체 검출







```
import numpy as np, cv2

img = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)
HSV_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hue_img = np.copy(HSV_img)[:, :, 0] # 색상 채널 컬러만 복사
th = [50, 100]

def onThreshold(value):
    th[0] = cv2.getTrackbarPos("Hue_th1", "result")
    th[1] = cv2.getTrackbarPos("Hue_th2", "result")

    _, result = cv2.threshold(hue_img, th[1], 255, cv2.THRESH_TOZERO_INV)
    cv2.threshold(result, th[0], 255, cv2.THRESH_BINARY, result)
    cv2.imshow("result", result)


cv2.namedWindow("result")
cv2.createTrackbar("Hue_th1", "result", th[0], 255, onThreshold)
cv2.createTrackbar("Hue_th2", "result", th[1], 255, onThreshold)
cv2.imshow("img", img)
cv2.waitKey(0)
```

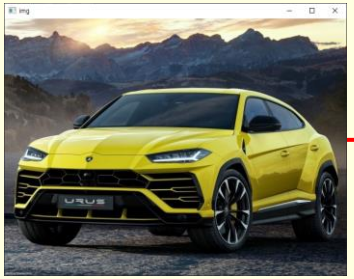


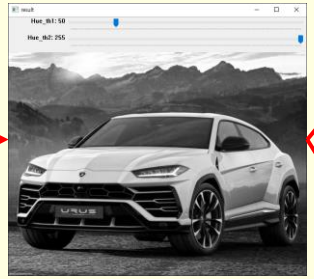
6. 화소 처리

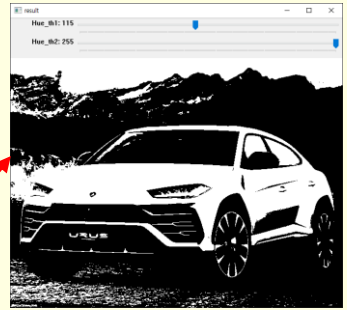
■ 6.4 컬러 공간 변환

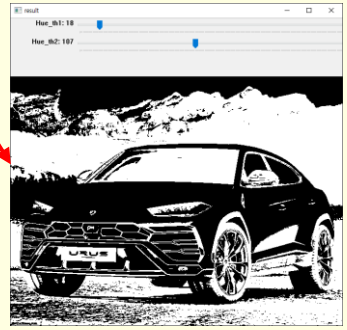
◆ 6.4.5 Hue 채널을 이용한 객체 검출

```
import numpy as np, cv2

...

hue_img = np.copy(HSV_img)[:, :, 2] # 명도 채널 컬러만 복사 했을 경우

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









CHAPTER 7
영역 처리

김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

7. 영역 처리

7.1 컬러 공간 변환

7.1.1 회선이용 블러링

```
import numpy as np, cv2
import matplotlib.pyplot as plt

# original image
img = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)

# matplotlib, gray scale
img =cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

그레이 스케일로 컬러채널 변환

```
# original image
cv2.imshow("original", img)
cv2.waitKey(0)
```

7. 영역 처리

■ 7.1 컬러 공간 변환

◆ 7.1.1 회선이용 블러링

```
...
cv2.imshow("original", img)

# blur convolution image
blur_mask = [1/9, 1/9, 1/9,
             1/9, 1/9, 1/9,
             1/9, 1/9, 1/9]
blur_mask = np.array(blur_mask, np.float32).reshape(3, 3)
blur_img = cv2.filter2D(img, cv2.CV_16S, blur_mask)
blur_img = cv2.convertScaleAbs(blur_img)

cv2.imshow("blur_img", blur_img)
cv2.waitKey(0)
```

1차원 -> 2차원으로 변환

```
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
```

예제 7.1.1 회선이용 블러링 – 01.bluring.py

```
01 import numpy as np, cv2
02
03 ## 회선 수행 함수 – 행렬 처리 방식(속도 면에서 유리)
04 def filter(image, mask):
05     rows, cols = image.shape[:2]
06     dst = np.zeros((rows, cols), np.float32)
07     ycenter, xcenter = mask.shape[0]//2, mask.shape[1]//2
08
09     for i in range(ycenter, rows - ycenter):
10         for j in range(xcenter, cols - xcenter):
11             y1, y2 = i - ycenter, i + ycenter + 1
12             x1, x2 = j - xcenter, j + xcenter + 1
13             roi = image[y1:y2, x1:x2].astype('float32')
14             tmp = cv2.multiply(roi, mask)
15             dst[i, j] = cv2.sumElems(tmp)[0]
16     return dst
17
18 ## 회선 수행 함수– 화소 직접 근접
19 def filter2(image, mask):
20     rows, cols = image.shape[:2]
21     dst = np.zeros((rows, cols), np.float32)          # 회선 결과 저장 행렬
22     ycenter, xcenter = mask.shape[0]//2, mask.shape[1]//2  # 마스크 중심 좌표
23
24     for i in range(ycenter, rows - ycenter):          # 입력 행렬 반복 순회
25         for j in range(xcenter, cols - xcenter):
26             sum = 0.0
27             for u in range(mask.shape[0]):            # 마스크 원소 순회
28                 for v in range(mask.shape[1]):
29                     y, x = i + u - ycenter, j + v - xcenter
30                     sum += image[y, x] * mask[u, v]   # 회선 수식
31             dst[i, j] = sum
32     return dst
```

마스크를 적용하는 함수

cv2.CV_16S는 **16**비트 **signed int**이다.
즉, **-32768 ~ 32767**까지 사용 가능

각각의 값을 절대값화 시키고,
정수화 시키는 함수

이미지는 **0~255**의 정수로만 이루어져 있기 때문에
이를 거쳐주어야 정상적인 이미지가 나온다.

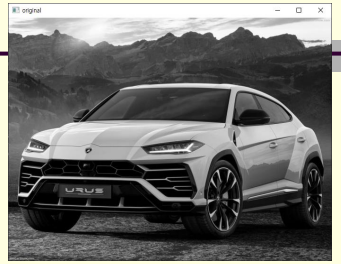
7. 영역 처리

■ 7.1 컬러 공간 변환

◆ 7.1.1 회선이용 블러링

```
...
cv2.imshow("original", img)

# blur convolution image
blur_mask = [1/9, 1/9, 1/9,
             1/9, 1/9, 1/9,
             1/9, 1/9, 1/9]
blur_mask = np.array(blur_mask, np.float32).reshape(3, 3)
blur_img = cv2.filter2D(img, cv2.CV_16S, blur_mask)
blur_img = cv2.convertScaleAbs(blur_img)

cv2.imshow("blur_img", blur_img)
cv2.waitKey(0)
```

7. 영역 처리



■ 7.1 컬러 공간 변환

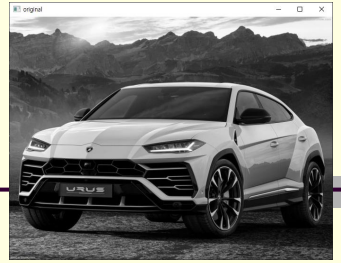
◆ 7.1.1 회선이용 블러링

```
...
cv2.imshow("original", img)

# blur convolution image
blur_mask = [1/9, 1/9, 1/9,
             1/9, 1/9, 1/9,
             1/9, 1/9, 1/9]
blur_mask = np.array(blur_mask, np.float32).reshape(3, 3)
blur_img = cv2.filter2D(img, cv2.CV_16S, blur_mask)
blur_img = cv2.convertScaleAbs(blur_img)

plt.figure(figsize=(12, 12))
pltx, plty = 2, 4
titles = ['blur_img']

for idx, title in enumerate(titles):
    plt.subplot(plty, pltx, idx+1)
    plt.axis('off')
    plt.title(title)
    plt.imshow(eval(title), cmap='gray')
plt.show()
cv2.waitKey(0)
```

앞으로 진행할 회선, 엣지 등의 이미지들을 한번에 출력하기 위해 미리 작성.

titles에 이미지 이름만 추가해주면 된다.

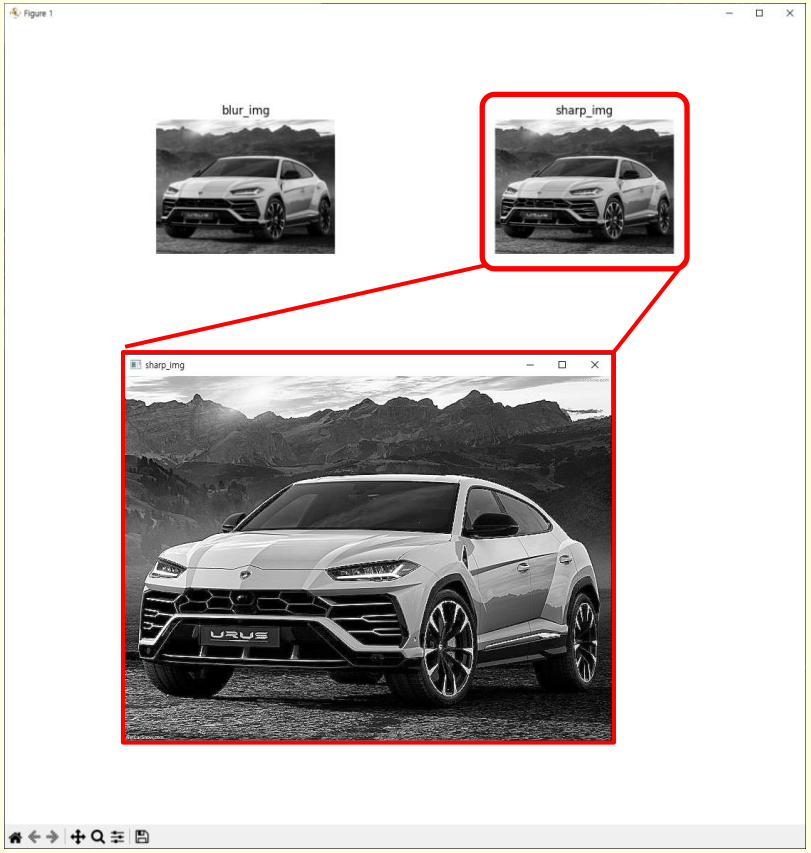
7. 영역 처리



7.1 컬러 공간 변환

◆ 7.1.2 회선이용 샤프닝

```
...
blur_img = cv2.convertScaleAbs(blur_img)

# sharp convolution image
sharp_mask = [0, -1, 0,
              -1, 5, -1,
              0, -1, 0]
sharp_mask = np.array(sharp_mask, np.float32).reshape(3, 3)
sharp_img = cv2.filter2D(img, cv2.CV_16S, sharp_mask)
sharp_img = cv2.convertScaleAbs(sharp_img)

plt.figure(figsize=(12, 12))
pltx, plty = 2, 4
titles = ['blur_img', 'sharp_img']

for idx, title in enumerate(titles):
    plt.subplot(plty, pltx, idx+1)
    plt.axis('off')
    plt.title(title)
    plt.imshow(eval(title), cmap='gray')
plt.show()
cv2.waitKey(0)
```



7. 영역 처리

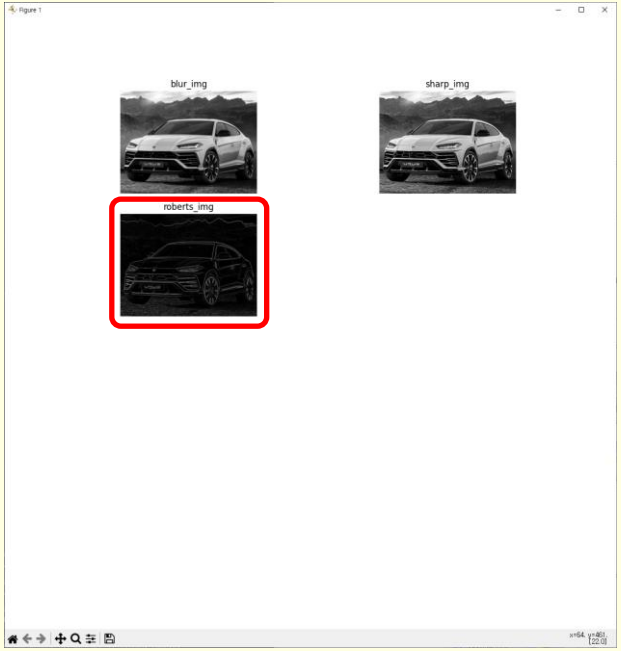
로버츠 마스크 특징:

대각선 방향으로 1과 -1을 배치하고, 나머지 값이 0이기 때문에 계산이 단순
따라서 차분의 크기가 작기 때문에 경계가 확실한 에지만을 추출
잡음에 매우 민감

7.2 에지 검출

7.2.3 로버츠 에지 검출

```
...
# 왼쪽 위에서 오른쪽 아래 대각선 방향으로 강조
roberts_lr_mask = [-1, 0, 0,
                    0, 1, 0,
                    0, 0, 0]
roberts_lr_mask = np.array(roberts_lr_mask, np.float32).reshape(3, 3)
roberts_lr_img = cv2.filter2D(img, cv2.CV_16S, roberts_lr_mask)
roberts_lr_img = cv2.convertScaleAbs(roberts_lr_img)

# 오른쪽 위에서 왼쪽 아래 대각선 방향으로 강조
roberts_rl_mask = [0, 0, -1,
                   0, 1, 0,
                   0, 0, 0]
roberts_rl_mask = np.array(roberts_rl_mask, np.float32).reshape(3, 3)
roberts_rl_img = cv2.filter2D(img, cv2.CV_16S, roberts_rl_mask)
roberts_rl_img = cv2.convertScaleAbs(roberts_rl_img)

# 두 이미지 결합
roberts_img = cv2.addWeighted(roberts_lr_img, 0.5, roberts_rl_img, 0.5, 0)
                    두 영상의 같은 위치에 존재하는 픽셀 값에 대하여 가중치 부여
...
titles = ['blur_img', 'sharp_img', 'roberts_img']
```



7. 영역 처리

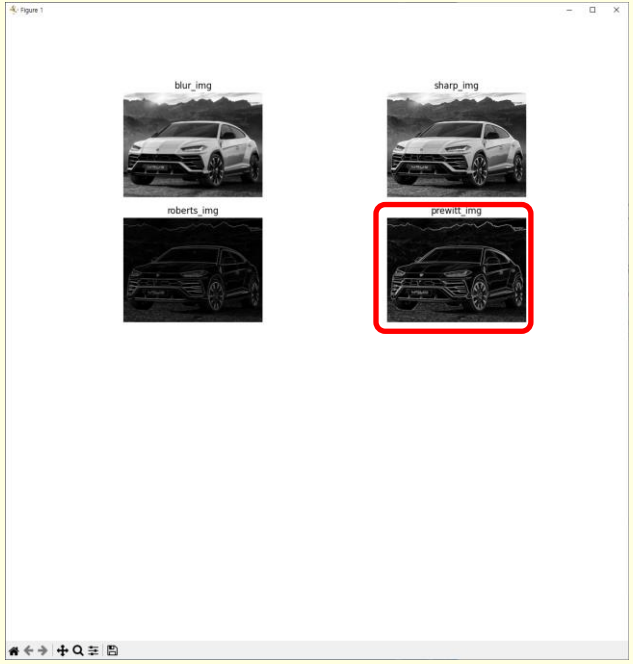
프리윗 마스크 특징:

로버츠 마스크의 단점을 보완하기 위해 고안됨
수직, 수평의 에지를 동등하게 찾는데 효과적이다.

7.2 에지 검출

7.2.4 프리윗 에지 검출

```
...
# 수직 방향 강조
prewitt_v_mask = [-1, 0, 1,
                  -1, 0, 1,
                  -1, 0, 1]
prewitt_v_mask = np.array(prewitt_v_mask, np.float32).reshape(3, 3)
prewitt_v_img = cv2.filter2D(img, cv2.CV_16S, prewitt_v_mask)
prewitt_v_img = cv2.convertScaleAbs(prewitt_v_img)

# 수평 방향 강조
prewitt_h_mask = [-1, -1, -1,
                   0, 0, 0,
                   1, 1, 1]
prewitt_h_mask = np.array(prewitt_h_mask, np.float32).reshape(3, 3)
prewitt_h_img = cv2.filter2D(img, cv2.CV_16S, prewitt_h_mask)
prewitt_h_img = cv2.convertScaleAbs(prewitt_h_img)

# 두 이미지 결합
prewitt_img = cv2.addWeighted(prewitt_v_img, 0.5, prewitt_h_img, 0.5, 0)

...
titles = ['blur_img', 'sharp_img', 'roberts_img', 'prewitt_img']
```



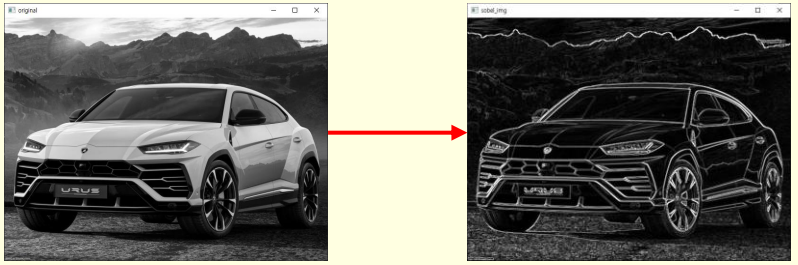
7. 영역 처리

소벨 마스크 특징:

소벨 마스크도 프리윗 마스크와 마찬가지로 수직, 수평을 검출한다.
추가로, 중심 계수의 차분 비중을 높였기 때문에 대각선도 잘 검출한다.

■ 7.2 에지 검출

◆ 7.2.5 소벨 에지 검출



```
...
# 수직 방향 강조
sobel_v_mask = [-1, 0, 1,
                -2, 0, 2,
                -1, 0, 1]
sobel_v_mask = np.array(sobel_v_mask, np.float32).reshape(3, 3)
sobel_v_img = cv2.filter2D(img, cv2.CV_16S, sobel_v_mask)
sobel_v_img = cv2.convertScaleAbs(sobel_v_img)

# 수평 방향 강조
sobel_h_mask = [-1, -2, -1,
                 0, 0, 0,
                 1, 2, 1]
sobel_h_mask = np.array(sobel_h_mask, np.float32).reshape(3, 3)
sobel_h_img = cv2.filter2D(img, cv2.CV_16S, sobel_h_mask)
sobel_h_img = cv2.convertScaleAbs(sobel_h_img)

# 두 이미지 결합
sobel_img = cv2.addWeighted(sobel_v_img, 0.5, sobel_h_img, 0.5, 0)

...
titles = [... , 'prewitt_img', 'sobel_img']
```

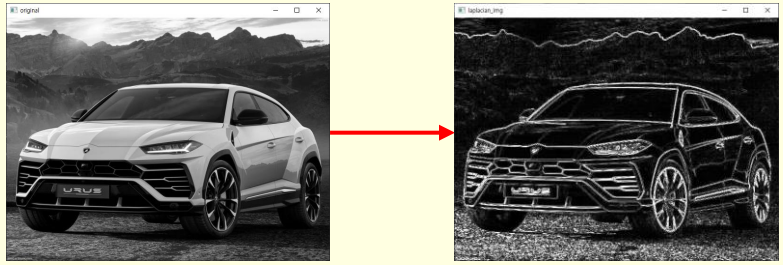
7. 영역 처리

라플라시안 마스크 특징:

대표적인 2차 미분 연산자
중심계수와 주변화소와 차분을 합하여 에지를 검출
주변화소에 잡음이 있으면 실제보다 더 많은 에지를 검출하는 경향이 있다.

7.2 에지 검출

7.2.6 라플라시안 에지 검출

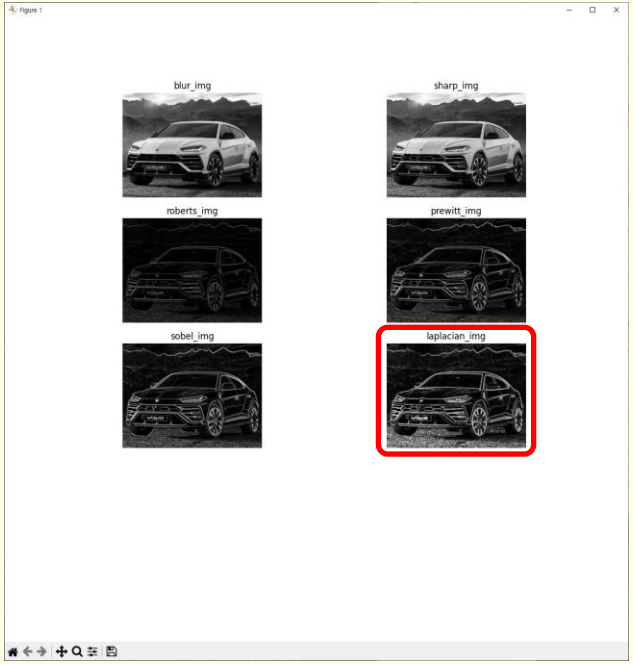


opencv에서 제공하는 라플라시안 함수

```
...
laplacian_img = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
laplacian_img = cv2.convertScaleAbs(laplacian_img)

...
titles = [... , 'sobel_img', 'laplacian_img']
```

ksize는 커널 사이즈를 의미함

```
# laplacian 4point edge image
laplacian_4p_mask = [ 0, 1, 0,
                      1, -4, 1,
                      0, 1, 0]
laplacian_4p_mask = np.array(laplacian_4p_mask, np.int16).reshape(3, 3) # 음수로 인해 int16 행렬 선언
laplacian_4p_img = cv2.filter2D(img, cv2.CV_16S, laplacian_4p_mask)
laplacian_4p_img = cv2.convertScaleAbs(laplacian_4p_img)

# laplacian 8point edge image
laplacian_8p_mask = [-1, -1, -1,
                     -1, 8, -1,
                     -1, -1, -1]
laplacian_8p_mask = np.array(laplacian_8p_mask, np.int16).reshape(3, 3) # 음수로 인해 int16 행렬 선언
laplacian_8p_img = cv2.filter2D(img, cv2.CV_16S, laplacian_8p_mask)
laplacian_8p_img = cv2.convertScaleAbs(laplacian_8p_img)

# laplacian merge edge image
laplacian_img = cv2.addWeighted(laplacian_4p_img, 0.5, laplacian_8p_img, 0.5, 0)
```



7. 영역 처리

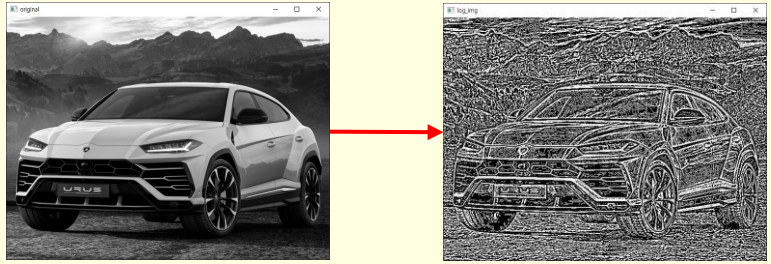
LoG 마스크 특징:
LoG: Laplacian of Gaussian

라플라시안은 잡음에 민감한 단점이 있다.
그래서 잡음을 먼저 제거하고 라플라시안을 수행하면
잡음에 강한 에지 검출이 가능하다.

7.2 에지 검출

7.2.7 LoG 에지 검출

```
...
# log edge image   잡음을 제거하기 위한 가우시안 블러처리
gaus_mask = cv2.GaussianBlur(img, (3, 3), 0, 0)
log_img = cv2.Laplacian(gaus_mask, cv2.CV_16S, 3).astype(np.uint8)
                잡음을 제거하고 라플라시안을 수행          음수 제거를 위함
...
titles = [... , 'laplacian_img', 'log_img']
```



7. 영역 처리

DoG 마스크 특징:
DoG: Difference of Gaussian

가우시안 스무딩 필터링의 차를 이용해서 에지를 검출하는 방법
두 개의 표준편차를 이용해서 가우시안 마스크를 만듬

7.2 에지 검출

7.2.7 LoG / DoG 에지 검출

```
...
# log edge image
gaus_mask = cv2.GaussianBlur(img, (3, 3), 0, 0)
log_img = cv2.Laplacian(gaus_mask, cv2.CV_16S, 3).astype(np.uint8)

# dog edge image
gaus2_mask = cv2.GaussianBlur(img, (9, 9), 0, 0)
dog_img = gaus_mask - gaus2_mask
```

두 가우시안 스무딩 필터링의 차 = **DoG**

```
...
titles = [... , 'log_img', 'dog_img']
```