CHAPTER 7
영역 처리

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

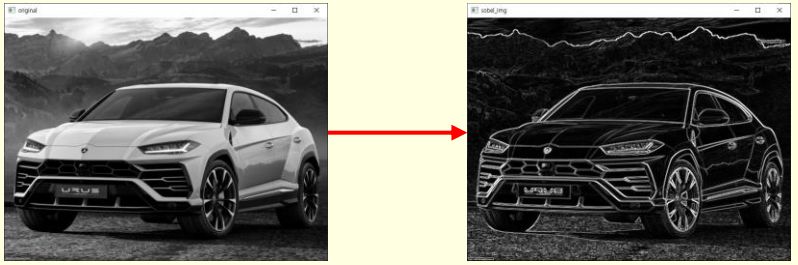
7. 영역 처리

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



```
...
# sobel vertical(수직)
sobel_v_mask = cv2.Sobel(img, cv2.CV_16S, 1, 0)
sobel_v_img = cv2.convertScaleAbs(sobel_v_mask)



# sobel horizontal(수평)
sobel_h_mask = cv2.Sobel(img, cv2.CV_16S, 0, 1)
sobel_h_img = cv2.convertScaleAbs(sobel_h_mask)



# 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

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

```
...
# sobel vertical(수직)
sobel_v_mask = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])
sobel_v_img = cv2.filter2D(img, cv2.CV_16S, sobel_v_mask)
sobel_v_img = cv2.convertScaleAbs(sobel_v_img)
```

=

```
...
# sobel vertical(수직)
sobel_v_mask = cv2.Sobel(img, cv2.CV_16S, 1, 0)
sobel_v_img = cv2.convertScaleAbs(sobel_v_mask)
```

```
# sobel horizontal(수평)
sobel_h_mask = np.array([[-1, -2, -1],
                         [0, 0, 0],
                         [1, 2, 1]])
sobel_h_img = cv2.filter2D(img, cv2.CV_16S, sobel_h_mask)
sobel_h_img = cv2.convertScaleAbs(sobel_h_img)
```

=

```
# sobel horizontal(수평)
sobel_h_mask = cv2.Sobel(img, cv2.CV_16S, 0, 1)
sobel_h_img = cv2.convertScaleAbs(sobel_h_mask)
```

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

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

cv2.Sobel을 사용하나, 직접 구현하나 결과가 같음을 볼 수 있다.
따라서, opencv에서 Sobel을 더 쉽게 사용할 수 있도록 제공한다.

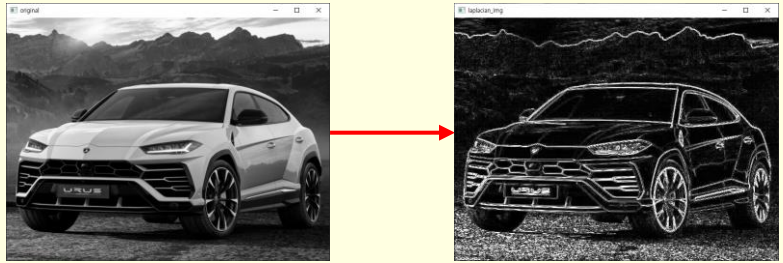
7. 영역 처리

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)
```

ksize는 커널 사이즈를 의미함

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

```
# 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)
```

라플라시안의 경우, 대비를 키우기 위해 **normalize**를 하기도 함.

```
lap_image = cv2.Laplacian(image, cv2.CV_16S, 1)
lap_image = cv2.normalize(lap_image, 0, 255, cv2.NORM_MINMAX).astype('uint8')
```

7. 영역 처리

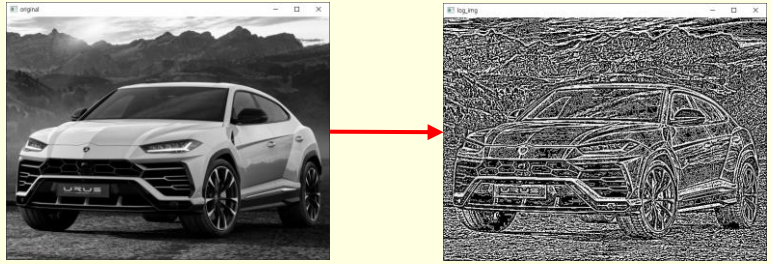
LoG 마스크 특징:
LoG: Laplacian of Gaussian

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

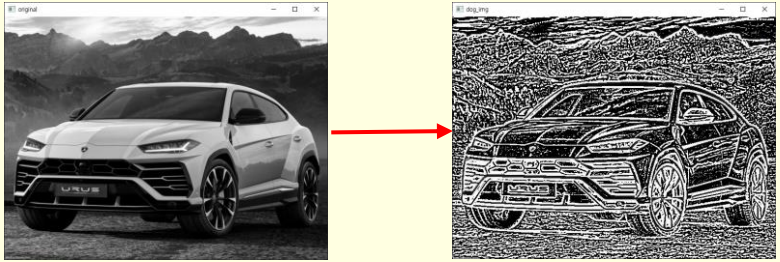
7.2.7 DoG 에지 검출

```
...
# 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']
```



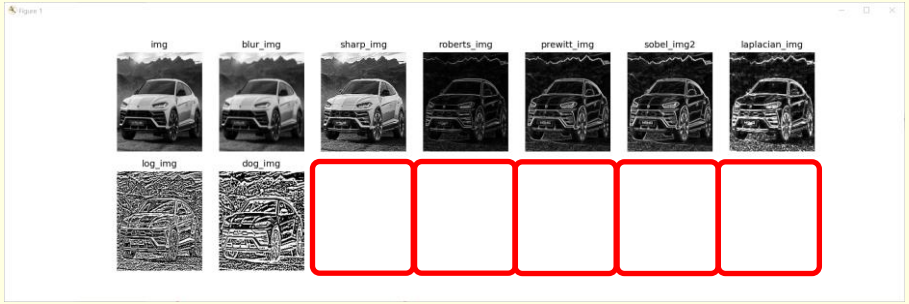
7. 영역 처리

Canny 마스크 특징:

영상 내에서 잡음(노이즈)가 에지로 검출되는 경우가 많다.
이 문제를 해결하기 위해 캐니 에지 검출 방법이 개발되었다.

7.2 에지 검출

7.2.8 캐니 에지 검출




```
...
plt.rcParams['toolbar'] = 'None'
plt.figure(figsize=(30, 8))
      7장씩 2줄로 표현
pltx, plty = 7, 2
titles = ['img', 'blur_img', 'sharp_img', 'roberts_img', 'prewitt_img', 'sobel_img', 'laplacian_img',
'log_img', 'dog_img']
for idx, title in enumerate(titles):
    plt.subplot(plty, pltx, idx+1)
    plt.axis('off')
    plt.title(title)
    plt.imshow(eval(title), aspect='auto', cmap='gray')

plt.show()
cv2.waitKey(0)
```

7. 영역 처리

Canny 마스크 특징:

영상 내에서 잡음(노이즈)가 에지로 검출되는 경우가 많다.
이 문제를 해결하기 위해 캐니 에지 검출 방법이 개발되었다.

■ 7.2 에지 검출

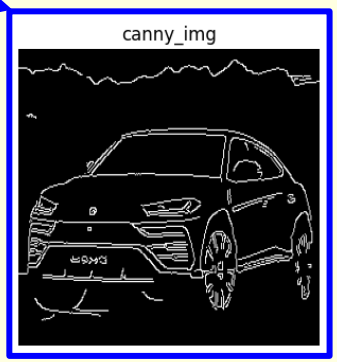
◆ 7.2.8 캐니 에지 검출

```
...
# canny edge image
canny_img = cv2.Canny(img, 400, 200)
```

image, threshold1, threshold2

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

1. 블러링을 통한 노이즈 제거 (가우시안 블러링)
2. 화소 기울기(gradiant)의 강도와 방향 검출 (소벨 마스크)
3. 비최대치 억제(non-maximum suppression)
4. 이력 임계값(hysteresis threshold)으로 에지 결정




```
## 캐니 에지 검출
gaus_img = cv2.GaussianBlur(image, (5, 5), 0.3)        # 표준 편차
Gx = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 1, 0, 3)      # x방향 마스크
Gy = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 0, 1, 3)      # y방향 마스크
sobel = cv2.magnitude(Gx, Gy)                          # 두 행렬 벡터 크기
        sobel = np.abs(Gx) + np.abs(Gy)
directs = cv2.phase(Gx, Gy) / (np.pi/4)                # 에지 기울기 계산 및 근사
directs = directs.astype(int) % 4                      # 8방향 → 4방향 축소
max_sobel = nonmax_suppression(sobel, directs)         # 비최대치 억제
hysteresis_th(max_sobel, 100, 150)                     # 이력 임계값
```

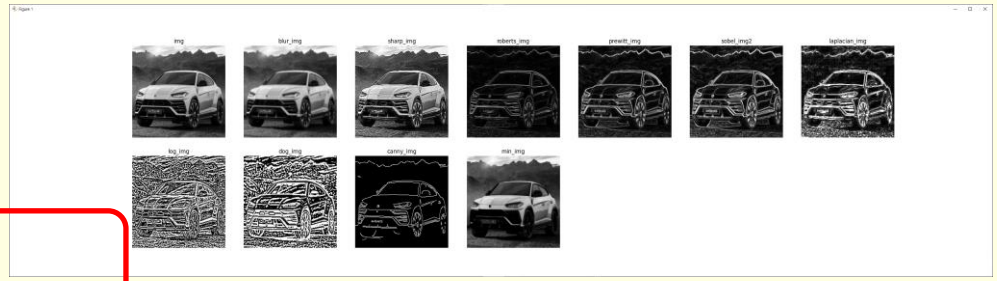
7. 영역 처리

최소값 필터링 특징:

입력 영상의 중심 화소에서 마스크로 씌워진 영역의 입력 화소들을 가져와서 계수를 구성하고 최솟값을 출력 화소로 결정하는 방법
최솟값은 0에 가깝기 때문에 **전체적으로 어두운 영상**이 된다.

■ 7.3 필터링

◆ 7.3.1 최소값 필터링



```
...
# minmax filter function
def minmax_filter(image, ksize, mode):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

    for i in range(center, rows - center):
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            mask = image[y1:y2, x1:x2]
            dst[i, j] = cv2.minMaxLoc(mask)[mode]
    return dst

# minfilter image
minfilter_img = minmax_filter(img, 3, 0) # 최솟값 필터링
```

=

```
# min filter image
min_img = cv2.erode(img, np.ones((3, 3), np.uint8), iterations=1)
...
titles = [... , 'canny_img', 'min_img']
```

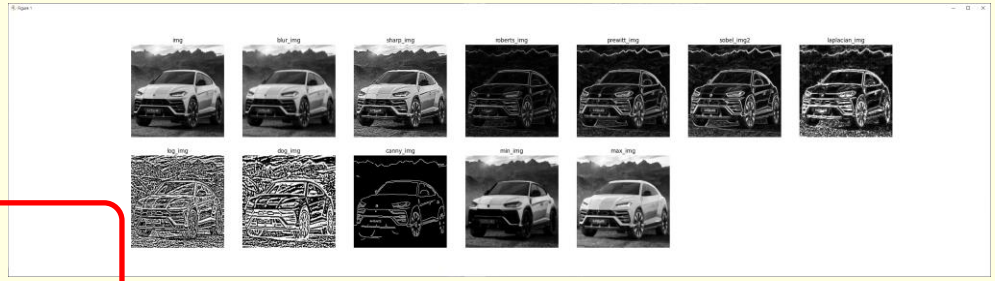
7. 영역 처리

최대값 필터링 특징:

입력 영상의 중심 화소에서 마스크로 씌워진 영역의 입력 화소들을 가져와서 계수를 구성하고 최대값을 출력 화소로 결정하는 방법
최대값은 255에 가깝기 때문에 **전체적으로 밝은 영상**이 된다.

■ 7.3 필터링

◆ 7.3.2 최대값 필터링



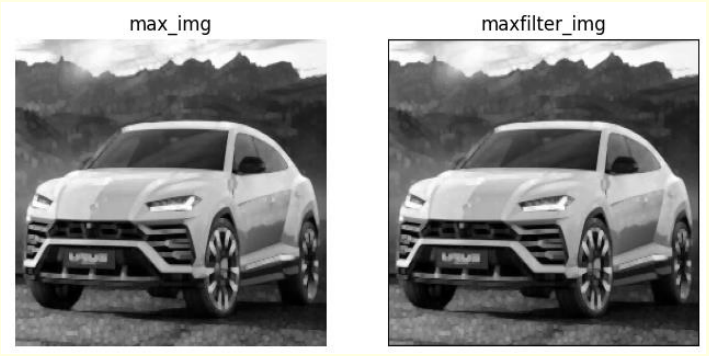
```
...
# minmax filter function
def minmax_filter(image, ksize, mode):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

    for i in range(center, rows - center):
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            mask = image[y1:y2, x1:x2]
            dst[i, j] = cv2.minMaxLoc(mask)[mode]
    return dst

# maxfilter image
maxfilter_img = minmax_filter(img, 3, 1) # 최대값 필터링
```

=

```
# max filter image
max_img = cv2.dilate(img, np.ones((3, 3), np.uint8), iterations=1)
...
titles = [... , 'min_img', 'max_img']
```



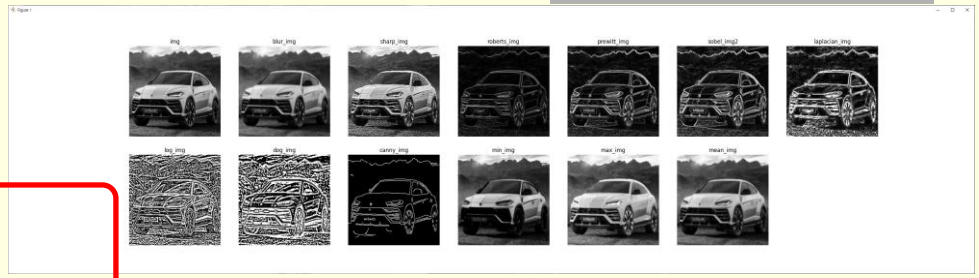
7. 영역 처리

평균값 필터링 특징:

입력 영상의 마스크로 씌워진 영역의 입력 화소들을 가져와서
화소들의 평균을 구하여 출력 화소로 지정하기 때문에 블러링의 효과가 나타난다.

■ 7.3 필터링

◆ 7.3.3 평균값 필터링



```python
# mean filter function
def mean_filter(image, ksize):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

    for i in range(rows):
        for j in range(cols):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            if y1 < 0 or y2 > rows or x1 < 0 or x2 > cols:
                dst[i, j] = image[i, j]
            else:
                mask = image[y1:y2, x1:x2]
                dst[i, j] = np.mean(mask)
    return dst

# meanfilter image
meanfilter_img = mean_filter(img, 3)
```

=

```python
# mean filter image
mean_img = cv2.blur(img, (3, 3))
```

7. 영역 처리

7.3 필터링

7.3.4 중간값 필터링

중간값 필터링 특징:

임펄스성 소금-후추 잡음을 잘 제거 해 주지만,
평균값 필터링에 비해 블러링 현상이 적다.
보통 명암도 영상에서 효과적이다.

```
# median filter image
median_img = cv2.medianBlur(img, 3)
```

=

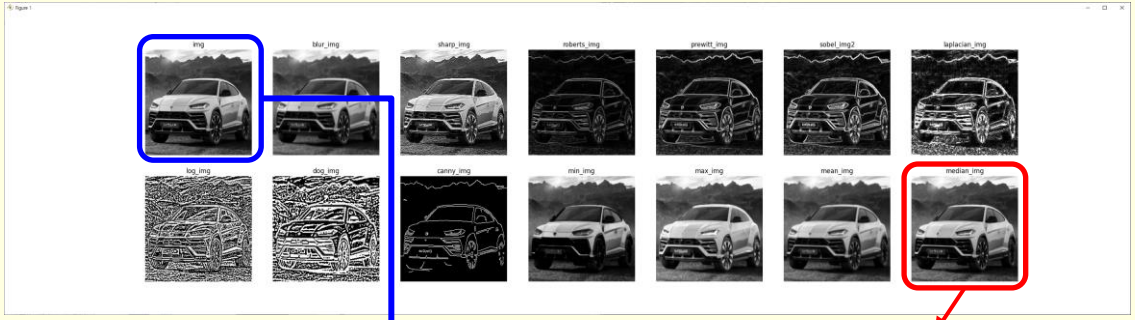
```
import numpy as np, cv2

def median_filter(image, size):                    # 미디언 필터링 함수
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2                            # 마스크 절반 크기

    for i in range(center, rows - center):         # 입력 영상 순회
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1    # 마스크 높이 범위
            x1, x2 = j - center, j + center + 1    # 마스크 너비 범위
            mask = image[y1:y2, x1:x2].flatten()   # 관심 영역 지정 및 벡터 변환

            sort_mask = cv2.sort(mask, cv2.SORT_EVERY_COLUMN)        # 정렬 수행
            dst[i, j] = sort_mask[sort_mask.size//2] # 출력 화소로 지정
    return dst

def salt_pepper_noise(img, n):                     # 소금 후추 잡음 생성 함수
    h, w = img.shape[:2]
    x, y = np.random.randint(0, w, n), np.random.randint(0, h, n)
    noise = img.copy()
    for (x, y) in zip(x, y):
        noise[y, x] = 0 if np.random.rand() < 0.5 else 255
    return noise
```



소금-후추 잡음의 예시

```
titles = ['img', 'blur_img', 'sharp_img', 'roberts_img', 'prewitt_img', 'sobel_img2',
'laplacian_img', 'log_img', 'dog_img', 'canny_img', 'min_img', 'max_img', 'mean_img', 'median_img']
```