Computer Vision

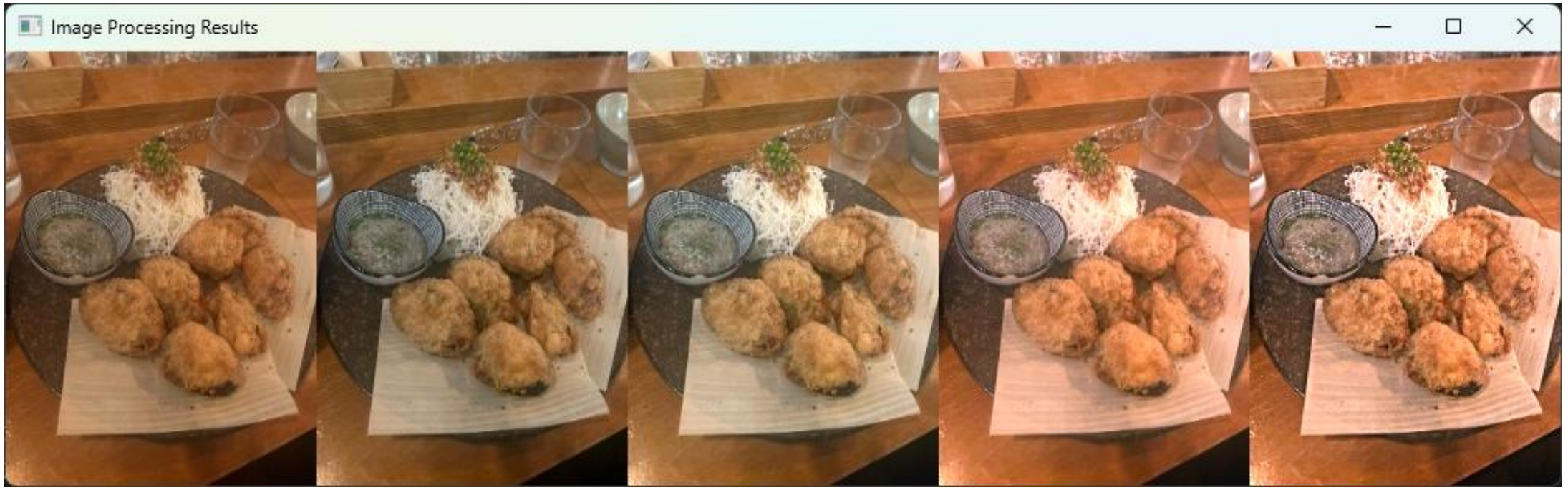
Lecture 05 에지와 이미지 분할

황선희

동양미래대학교
DONGYANG MIRAE UNIVERSITY

실습 5. 음식사진 필터 만들기

- food.jpg 이미지 또는, 직접 촬영한 음식 사진을 더 맛있어 보이는 사진으로 변환하기
- 실습함에 원본 → 변환과정 → 최종이미지 순서로 이어붙인(cv2.hconcat([img1, img2,]) 이미지 파일 제출하기 (cv2.imwrite로 저장) - **변환중 이미지 수는 자유롭게 조절**



원본 변환중 변환중 변환중 최종

실습 5. 음식사진 필터 만들기

- 1. 콘트라스트 스트레칭

```python
# 1. 콘트라스트 스트레칭
def contrast_stretching(img):
    # 채널별로 처리 (BGR 컬러 이미지)
    result = np.zeros_like(img)
    for i in range(3):  # B, G, R 각각의 채널에 대해
        min_val = np.min(img[:, :, i])
        max_val = np.max(img[:, :, i])
        # 각 채널별로 최소, 최대값에 기반해 선형적으로 대비 조정
        result[:, :, i] = ((img[:, :, i] - min_val) / (max_val - min_val) * 255).astype(np.uint8)
    return result
```

실습 5. 음식사진 필터 만들기

- 2. 밝기 및 대비 조정
- 3. 선명도 향상

```python
# 2. 색상 보정
def adjust_brightness_contrast(img, alpha=1.1, beta=10):
    # alpha: 대비 조정 계수 (1.0보다 크면 대비 증가)
    # beta: 밝기 증가 값 (양수면 밝기 증가, 음수면 밝기 감소)
    return cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

# 3. 선명도 향상 (Unsharp Masking)
def sharpen_image(img):
    # 가우시안 블러 적용 후 원본 이미지와 차이를 계산
    blurred = cv2.GaussianBlur(img, (9, 9), 10.0)
    sharp = cv2.addWeighted(img, 1.5, blurred, -0.5, 0)
    return sharp
```

실습 5. 음식사진 필터 만들기

- 4. Red(빨강) 색 강화

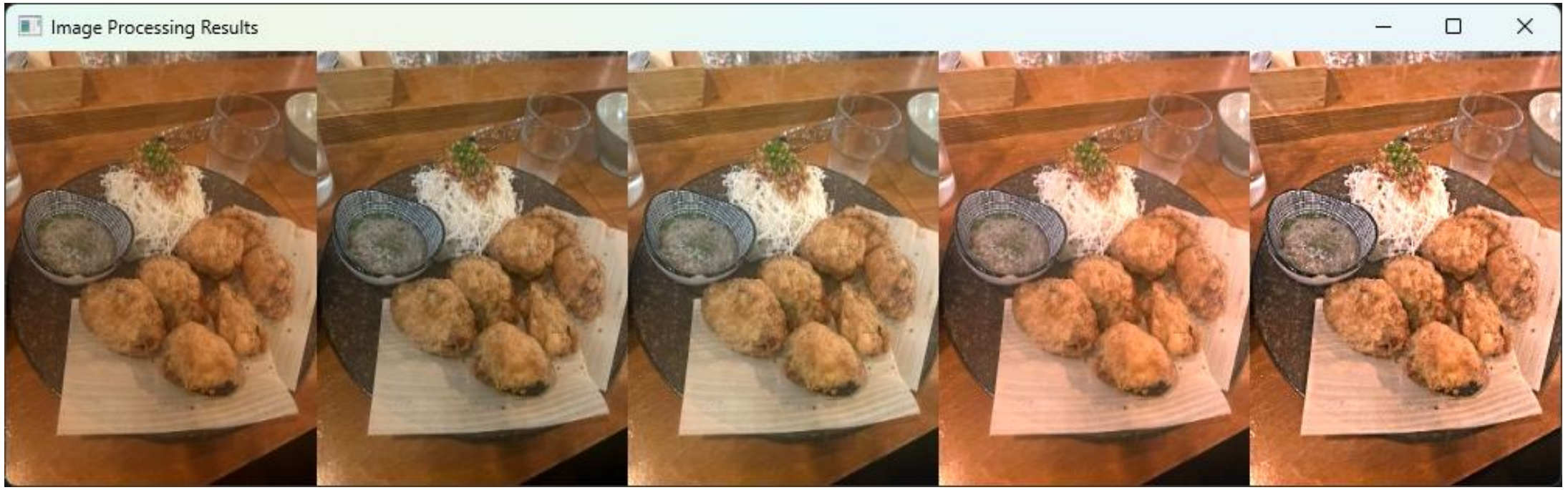
```python
# 4. Red 채널 강화
def boost_red_channel(img, red_boost=1.2):
    # BGR 이미지에서 R 채널에만 boost 적용
    img_copy = img.copy()
    img_copy[:, :, 2] = np.clip(img_copy[:, :, 2] * red_boost, 0, 255)
    return img_copy
```

실습 5. 음식사진 필터 만들기

```
# 음식 사진 보정 절차
# Step 1: 콘트라스트 스트레칭
contrast_stretched = contrast_stretching(image)

# Step 2: 색상 보정 (밝기를 덜하고 대비만 약간 증가)
color_corrected = adjust_brightness_contrast(contrast_stretched, alpha=1.1, beta=10)

# Step 3: Red 채널 강화
red_boosted = boost_red_channel(color_corrected, red_boost=1.2)

# Step 4: 선명도 향상
sharpened_image = sharpen_image(red_boosted)

result = cv2.hconcat([image, contrast_stretched, color_corrected, red_boosted,
sharpened_image])
cv2.imshow('Image Processing Results', result)

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

실습 5. 음식사진 필터 만들기

- food.jpg 이미지 또는, 직접 촬영한 음식 사진을 더 맛있어 보이는 사진으로 변환하기
- 실습함에 원본 → 변환과정 → 최종이미지 순서로 이어붙인(cv2.hconcat([img1, img2,]) 이미지 파일 제출하기 (cv2.imwrite로 저장) - **변환중 이미지 수는 자유롭게 조절**



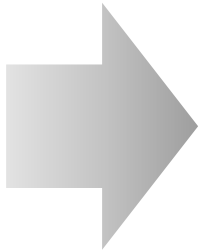
원본 변환중 변환중 변환중 최종

실습 6. 이미지 보간법 적용

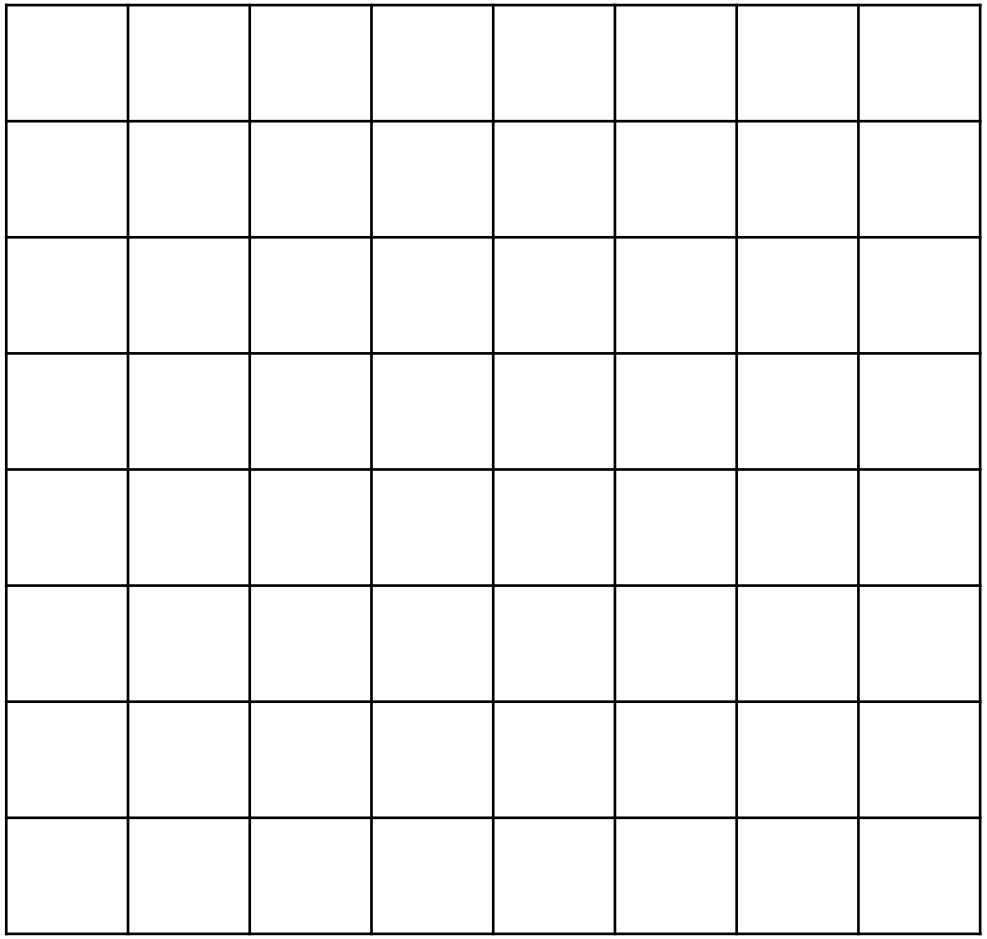
- 4x4의 numpy array를 만들어서, 세 가지 이미지 보간법에 따른 결과를 출력해보기
  - `arr = np.array([[10, 20, 30, 40],[50, 60, 70, 80],[90, 70, 80, 100],[110, 120, 130, 140]], dtype=np.uint8)`
  - 코드 작성 전, 결과를 계산하여 예측해보기
  - 8x8 크기로 변환하기
  - 보간법: NN, Bilinear, Bicubic
  - 실습함에 제출 불필요

10	20	30	40
50	60	70	80
90	70	80	100
110	120	130	140

Numpy array



보간법 적용



실습 6. 이미지 보간법 적용

- Nearest Neighbor (NN)

10	20	30	40
50	60	70	80
90	70	80	100
110	120	130	140

```
array([[ 10,  10,  20,  20,  30,  30,  40,  40],
       [ 10,  10,  20,  20,  30,  30,  40,  40],
       [ 50,  50,  60,  60,  70,  70,  80,  80],
       [ 50,  50,  60,  60,  70,  70,  80,  80],
       [ 90,  90,  70,  70,  80,  80, 100, 100],
       [ 90,  90,  70,  70,  80,  80, 100, 100],
       [110, 110, 120, 120, 130, 130, 140, 140],
       [110, 110, 120, 120, 130, 130, 140, 140]], dtype=uint8)
```

실습 6. 이미지 보간법 적용

- Bilinear

10	20	30	40
50	60	70	80
90	70	80	100
110	120	130	140

```
array([[ 10,  12,  17,  22,  27,  32,  37,  40],
       [ 20,  22,  27,  32,  37,  42,  47,  50],
       [ 40,  42,  47,  52,  57,  62,  67,  70],
       [ 60,  61,  62,  65,  70,  76,  82,  85],
       [ 80,  77,  71,  70,  75,  82,  91,  95],
       [ 95,  92,  86,  85,  90,  97, 106, 110],
       [105, 106, 107, 110, 115, 121, 127, 130],
       [110, 112, 117, 122, 127, 132, 137, 140]], dtype=uint8)
```

실습 6. 이미지 보간법 적용

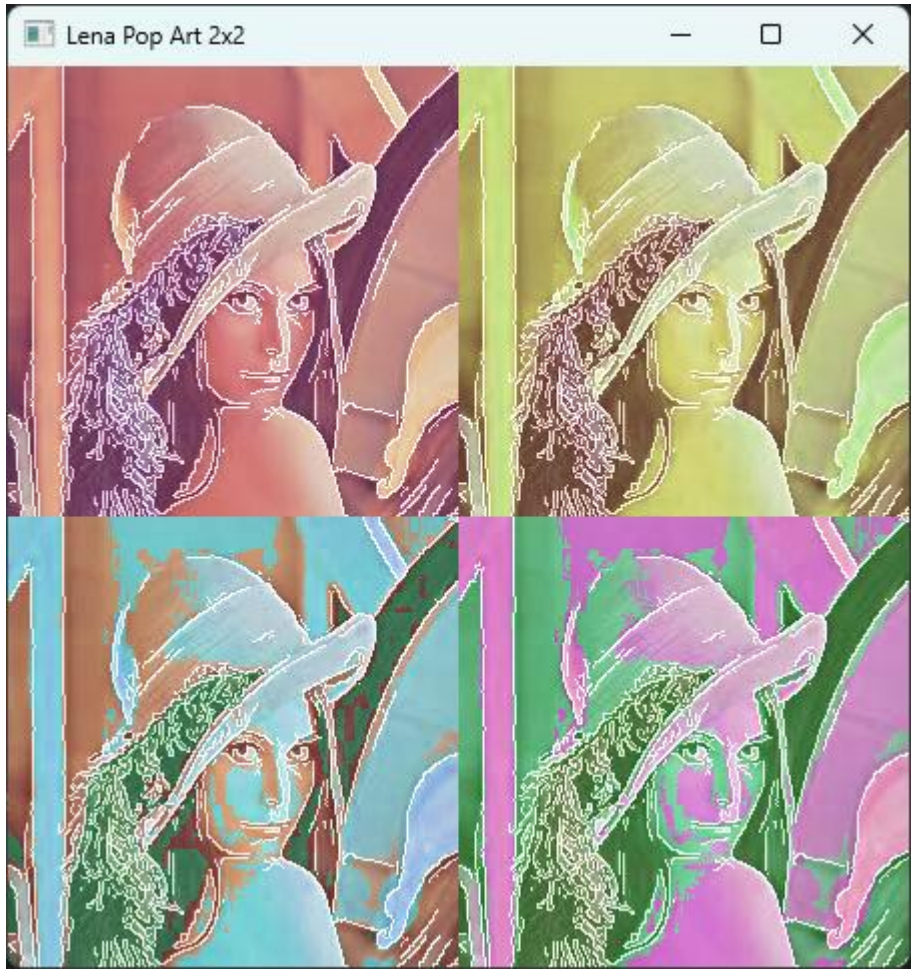
- Bilcubic

10	20	30	40
50	60	70	80
90	70	80	100
110	120	130	140

```
array([[  5,   8,  12,  19,  23,  29,  34,  37],
       [ 16,  20,  25,  32,  36,  42,  47,  49],
       [ 35,  39,  46,  53,  57,  63,  67,  70],
       [ 62,  63,  63,  67,  71,  79,  85,  89],
       [ 82,  76,  67,  64,  69,  79,  89,  95],
       [100,  95,  85,  82,  88,  98, 107, 113],
       [107, 107, 107, 111, 116, 123, 129, 133],
       [111, 115, 121, 129, 133, 138, 143, 145]], dtype=uint8)
```

실습 7. 이미지 효과 적용하기

- 레나(lena.jpg) 이미지를 활용하여 다음 예시와 유사한 형태의 이미지 제작해보기
  - 실습함에 생성된 결과 이미지 파일만 제출
  - 2x2로 이미지 이어붙이기



실습 7. 이미지 효과 적용하기

- 네온사인 효과 (가장자리 극대화)

```python
# 1. 네온사인 효과 함수 정의
def neon_effect(img):
    # 가장자리 검출
    edges = cv2.Canny(img, 100, 200)
    edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

    # 네온 효과: 가장자리 위로 색상 추가
    neon_image = cv2.addWeighted(img, 0.6, edges_colored, 0.4, 0)

    # 밝기와 대비를 좀 더 강조하여 네온 느낌을 줌
    bright_neon = cv2.convertScaleAbs(neon_image, alpha=1.3, beta=40)

    return bright_neon
```

실습 7. 이미지 효과 적용하기

- 팝아트 효과 (네온효과 적용 이미지에, 색상 변경, HSV 컬러공간에서 H값만 조절)

```
# 2. 팝아트 스타일 효과 함수 (색상 변환)
def pop_art_effect(img, color_shift):
    hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Hue 값에 색상 변화 적용
    hsv_image[:, :, 0] = (hsv_image[:, :, 0] + color_shift) % 180  # 0~180 범위에서 색상 변경

    # HSV에서 다시 BGR로 변환
    return cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)
```

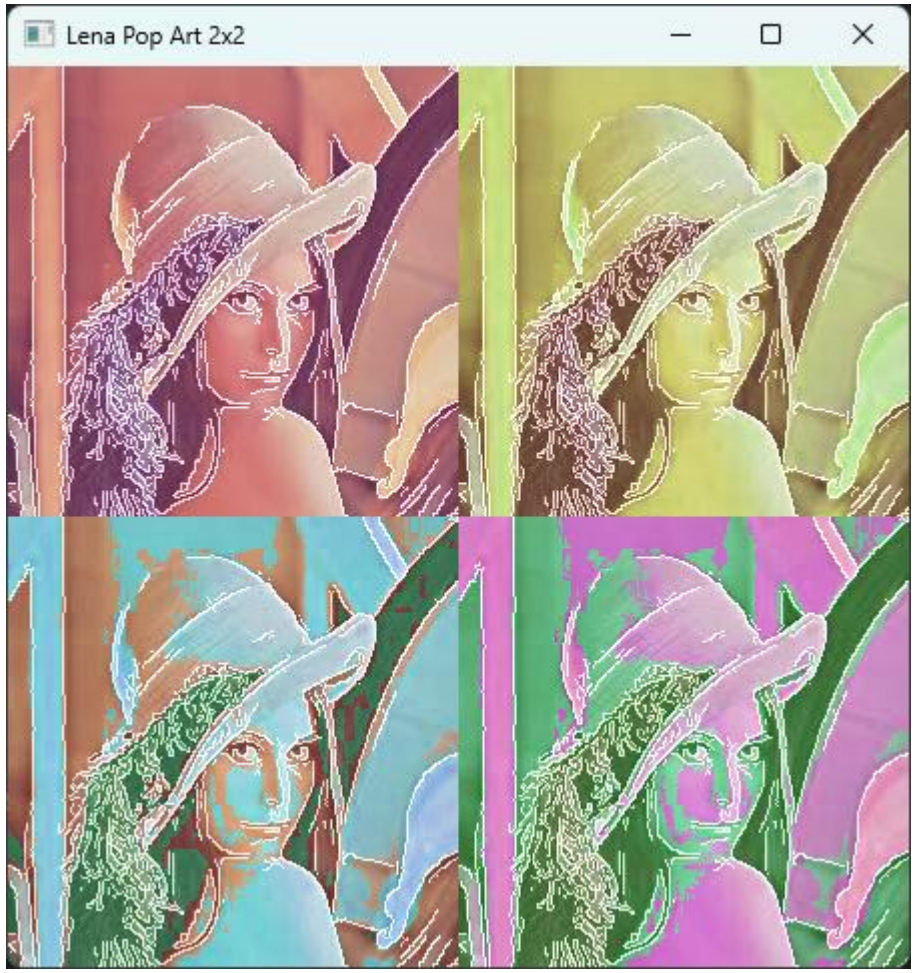
실습 7. 이미지 효과 적용하기

- 네온사인 효과 → 팝아트 효과를 위한 색상 변환

```
# 4. 네온사인 효과 적용
neon_image = neon_effect(image)

# 5. 팝아트 색상 변환 3가지
pop_art_1 = pop_art_effect(neon_image, 30)  # Hue 30도 변환
pop_art_2 = pop_art_effect(neon_image, 90)  # Hue 90도 변환
pop_art_3 = pop_art_effect(neon_image, 150)  # Hue 150도 변환

# 6. 2x2 배열로 결합
top_row = cv2.hconcat([neon_image, pop_art_1])
bottom_row = cv2.hconcat([pop_art_2, pop_art_3])
pop_art_grid = cv2.vconcat([top_row, bottom_row])

# 7. 결과 시각화
cv2.imshow('Lena Pop Art 2x2', pop_art_grid)
```

실습 7. 이미지 효과 적용하기

- 레나(lena.jpg) 이미지를 활용하여 다음 예시와 유사한 형태의 이미지 제작해보기
  - 실습함에 생성된 결과 이미지 파일만 제출
  - 2x2로 이미지 이어붙이기



목차

1. 에지 검출 개요

- 에지(Edge) 검출(Detection)과 영역 분할은 컴퓨터 비전 초창기부터 중요한 연구 주제
  - 이미지 분석, 의료영상 처리, 얼굴의 특징 추출 등 다양한 영역에 활용

(a) 원래 영상 (b) 영역 영상(사람이 분할)

그림 4-1 영역 분할을 위한 BSDS 데이터셋

1. 에지 검출 개요

- 에지 검출 알고리즘
  - 물체 내부는 명암이 서서히, 경계는 급격히 변하는 특성을 활용하여 경계 부분을 검출

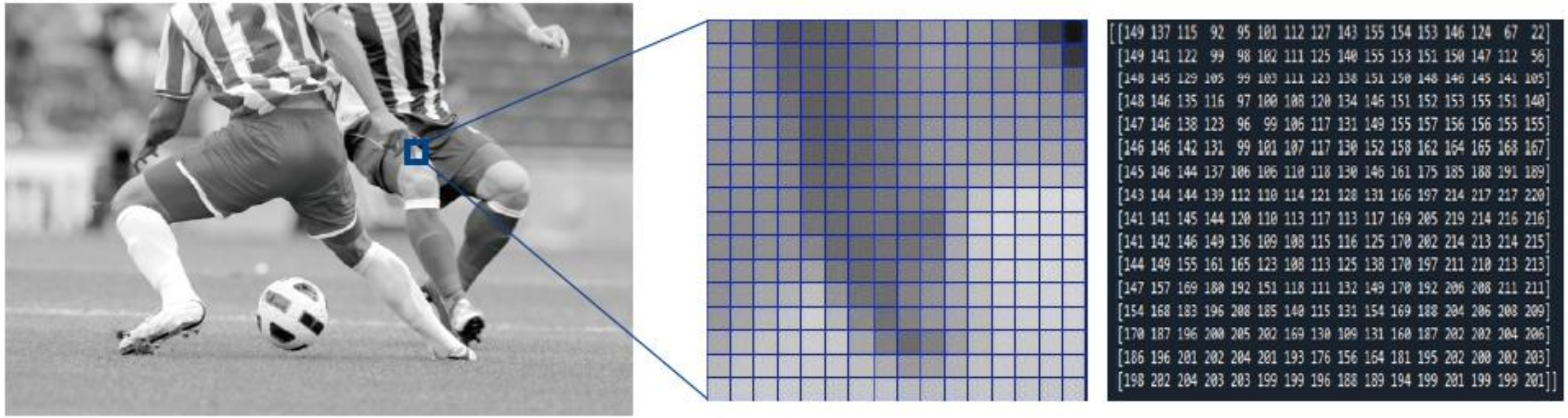
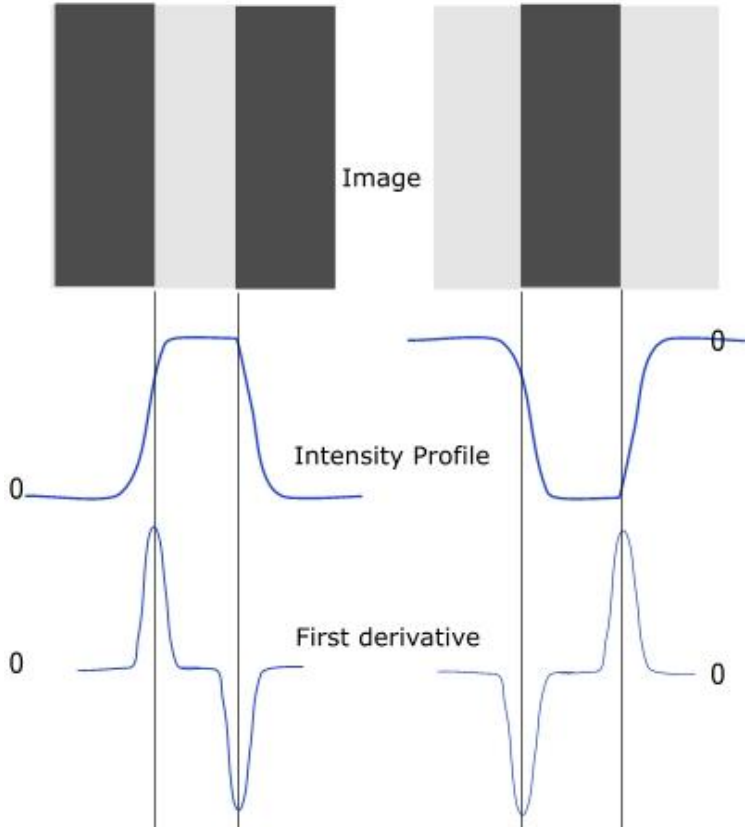


그림 4-2 명암 변화를 확인하기 위해 영상 일부를 확대

2. 미분연산을 통한 에지검출

- **1차 미분**을 통해 이미지 픽셀 값들 사이에 기울기가 발생한 구간을 에지검출에 활용
  - 이미지는 연속이 아니므로 미분이 불가능하여 근사치를 사용
  - 실제 영상에 대해 1차 미분을 적용하기 위해 [1 -1]의 필터를 컨볼루션 연산에 사용



	0	1	2	3	4	5	6	7	8	9
원래 영상 f	2	2	2	2	2	2	5	5	5	5
에지 영상 f'	0	0	0	0	0	3	0	0	0	-

필터 u= [-1 1] 로 컨볼루션

(b) 디지털 영상의 미분(필터 u로 컨볼루션)

더 정확한 에지 추출을 위해 개선된 필터

u_x=

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

u_y=

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

(a) 프레윗(Prewitt) 연산자

u_x=

-1	0	1
-2	0	2
-1	0	1

u_y=

-1	-2	-1
0	0	0
1	2	1

Sobel x **Sobel y**

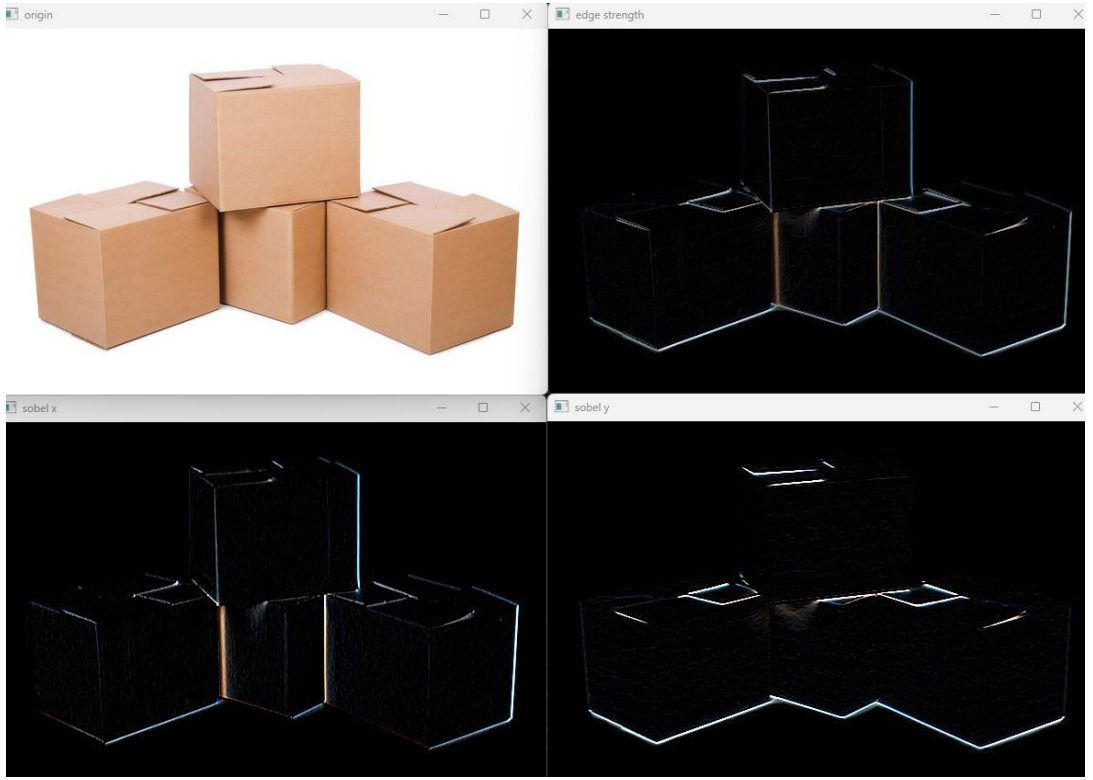
(b) 소벨(Sobel) 연산자

그림 4-6 에지 연산자

2. 미분연산을 통한 에지검출

- 소벨(Sobel) 필터를 사용한 에지검출 결과
  - 함수에 이미지, 출력데이터타입(-1은 원본과 동일), x축 미분차수, y축 미분차수, 커널사이즈 순 입력

```
import cv2

# 이미지 불러오기
img = cv2.imread('img/box.jpg')

# Sobel 에지 검출
sobelx = cv2.Sobel(img, -1, 1, 0, ksize=3) # x방향
sobely = cv2.Sobel(img, -1, 0, 1, ksize=3) # y방향
edge_strength =
cv2.addWeighted(sobelx,0.5,sobely,0.5,0)
cv2.imshow('origin', img)
cv2.imshow('sobel x', sobelx)
cv2.imshow('sobel y', sobely)
cv2.imshow('edge strength', edge_strength)

cv2.waitKey()
```

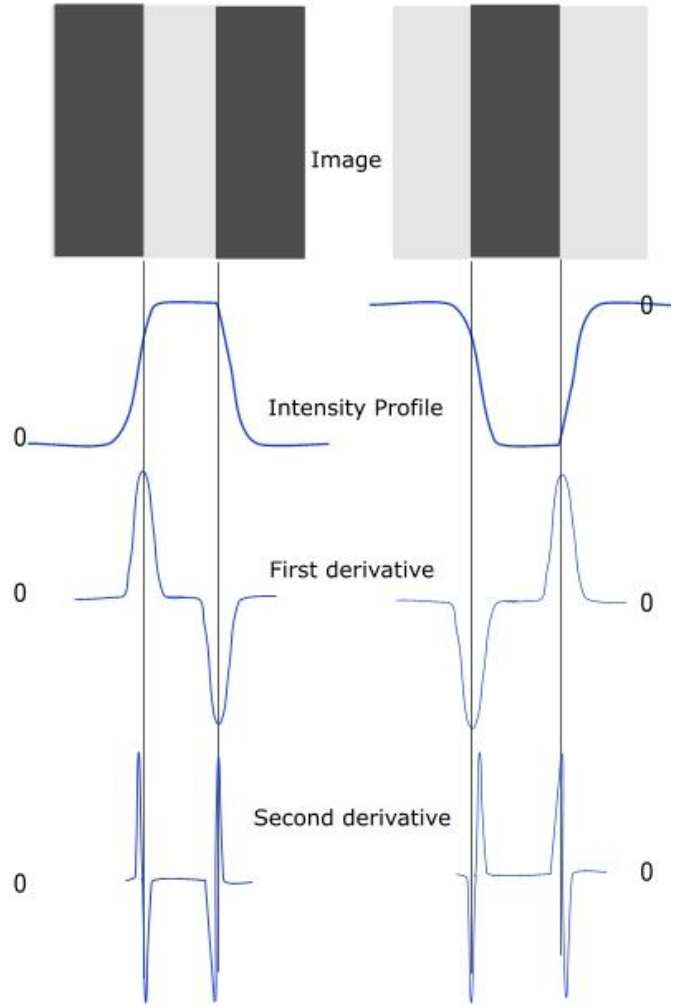


실습 8. Lena 이미지에 Sobel 필터 적용하기

- lena.jpg 이미지에 필터 적용하기
  - 실습파일에 소벨필터 적용 결과 이미지 추가하기

2. 미분연산을 통한 에지검출

- **2차 미분**을 통해 이미지의 기울기의 변화량 정보를 에지검출에 활용



1차 미분 결과는 두꺼운 에지구간(에지여부 불명확한 구간) 나타남

	0	1	2	3	4	5	6	7	8	9	0	1
원래 영상 f	3	3	3	3	5	7	7	7	7	3	3	3

필터 $u=$ | -1 | 1 | 로 컨볼루션

1차 미분 f'	0	0	0	2	2	0	0	0	-4	0	0	-

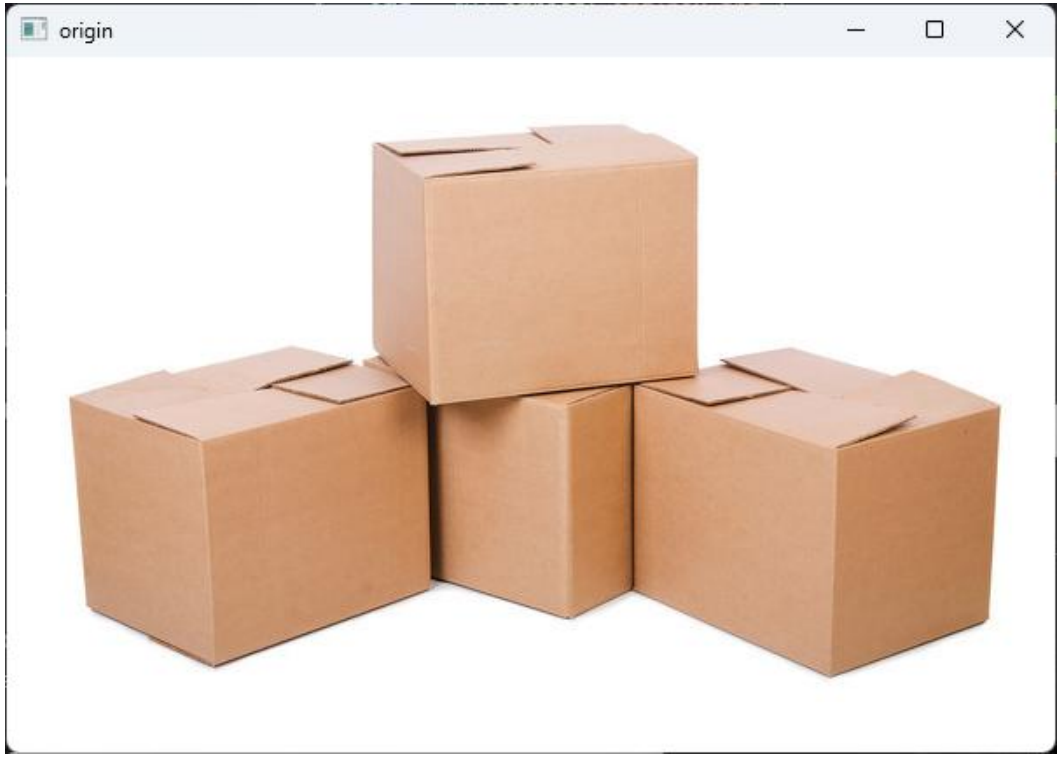
필터 $u=$ | -1 | 1 | 로 컨볼루션

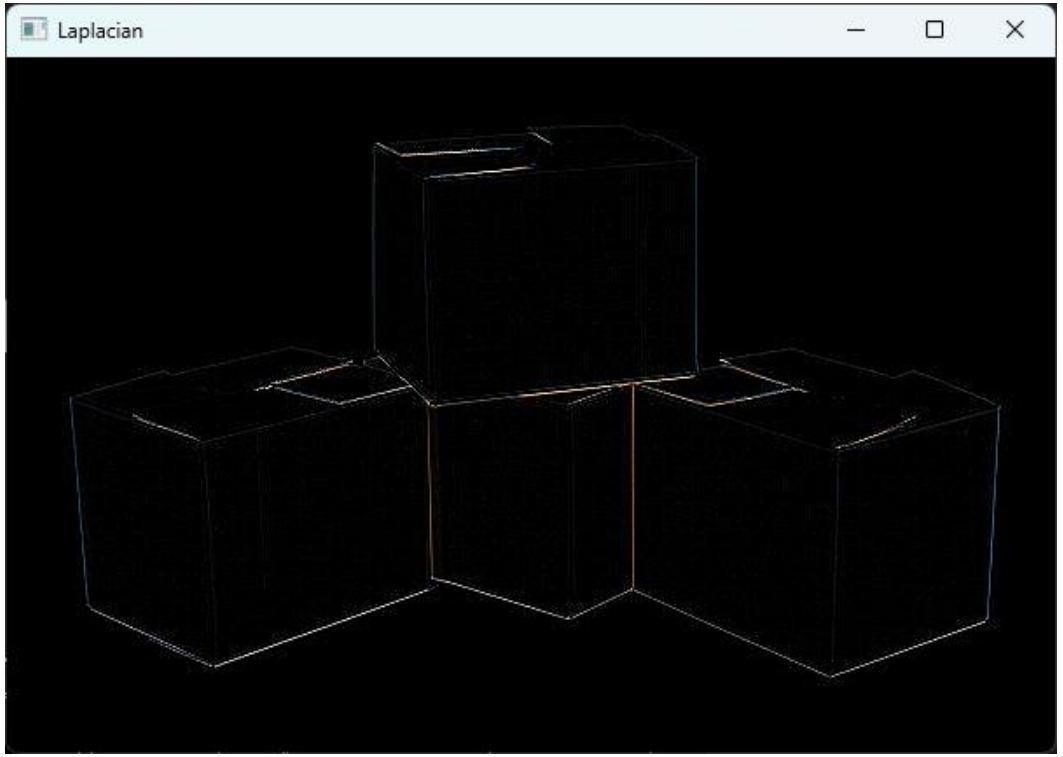
2차 미분 f''	0	0	2	0	-2	0	0	-4	4	0	-	-

2. 미분연산을 통한 에지검출

- 대표 알고리즘인 라플라시안(Laplacian) 연산을 통해 2차 미분을 적용한 에지추출
  - cv2.Laplacian 함수의 두번째 인자(-1)는 출력 이미지의 정밀도(ddepth) (cv2.CV_8U; 8bit unsigned int)

```
laplacian = cv2.Laplacian(img, cv2.CV_8U)
```





3. 캐니(Canny) 에지검출

- 에지를 효과적으로 검출하면서 잡음을 줄일 수 있는 기법
  - 낮은 오류율, 에지 위치 정확도, 단일 에지(하나의 점으로 표현)로 표현할 수 있는 좋은 에지 검출기를 목표로 생성된 기술
  - 가우시안 스무딩 → 그레디언트 계산 → 비최대 억제 → 이중 임계값 사용

(무작위로 발생되는 에지를 줄이기 위함) (소벨 필터를 활용) (NMS; Non-maximum suppression) (에지 판단을 위한 두 임계값)

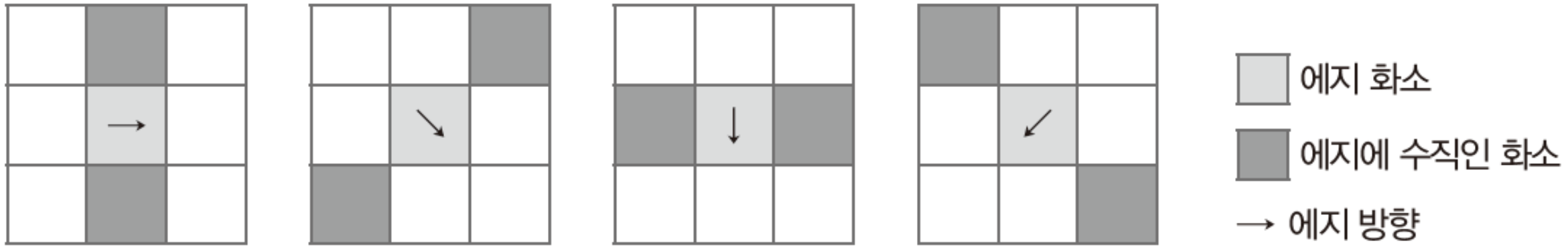


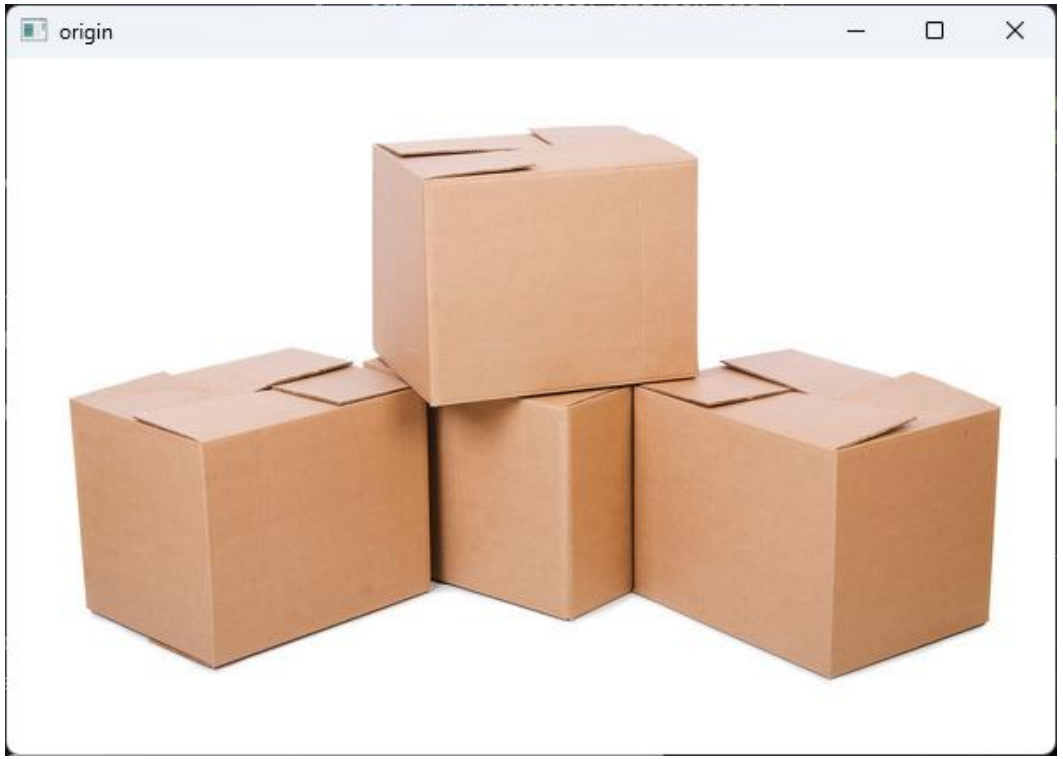
그림 4-8 비최대 억제

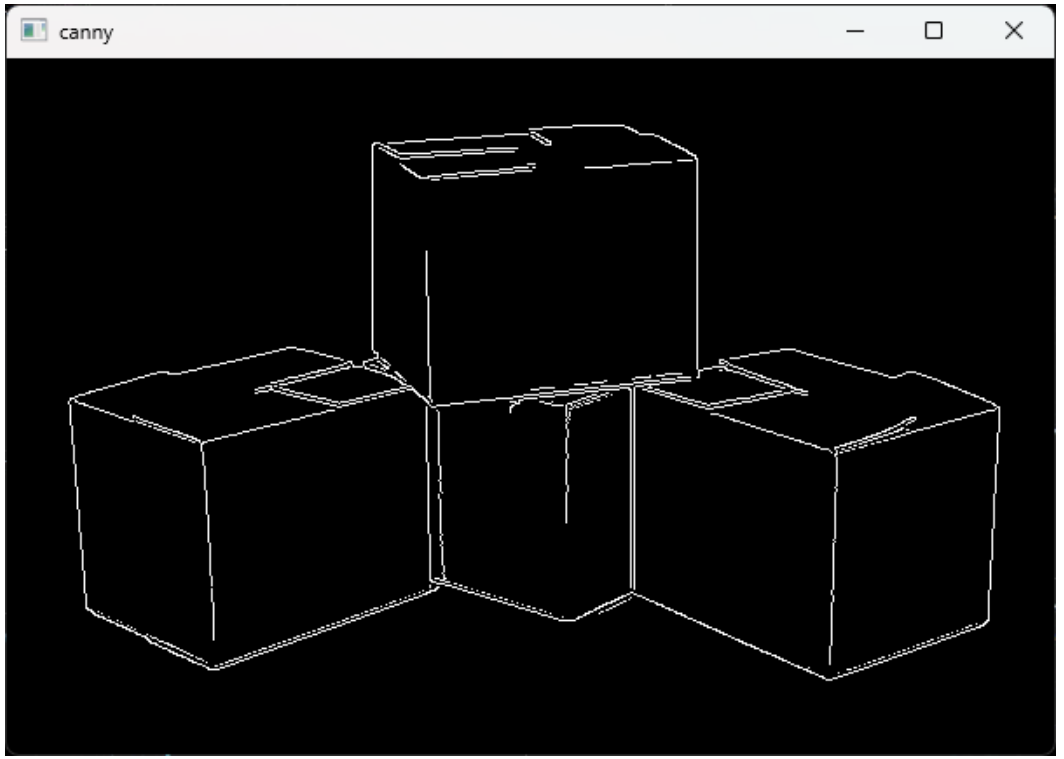
날카로운 에지 생성을 위해
(두꺼운 에지 생성을 피하기 위해)
로컬(좁은 영역)에서 최대값인지 확인

3. 캐니(Canny) 에지검출

- 캐니에지 적용 결과 → 이진(Binary) 이미지로 결과가 출력됨
  - 이미지, 임계치1(낮은 임계치), 임계치2(높은임계치) 순으로 입력

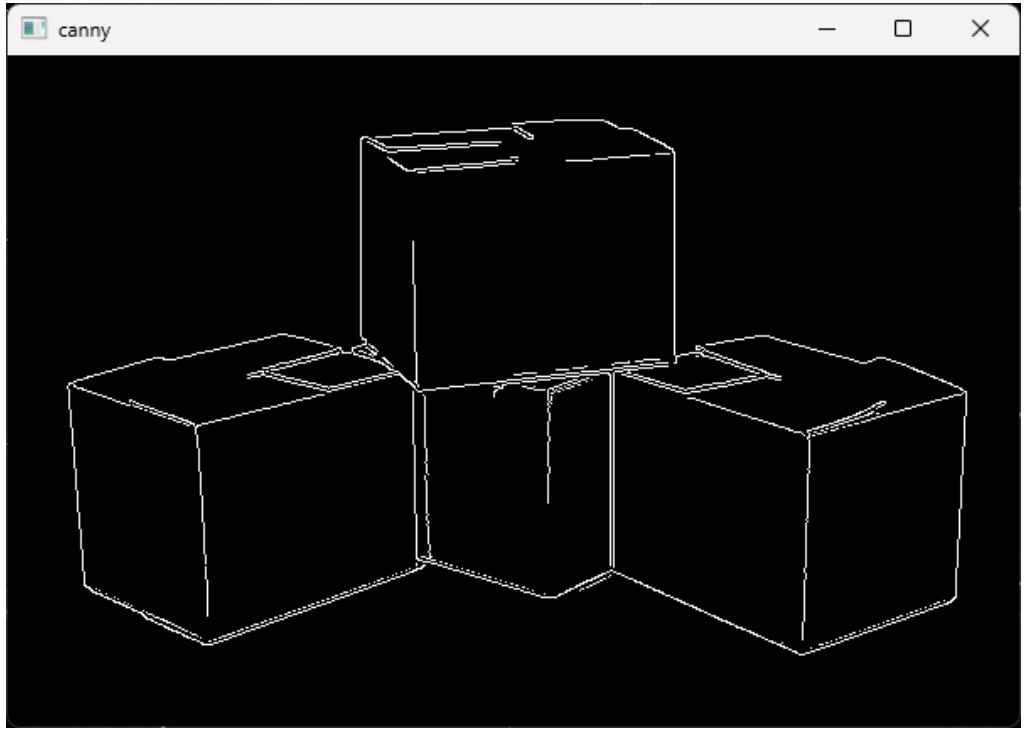
```
canny = cv2.Canny(img, 100, 150)
```

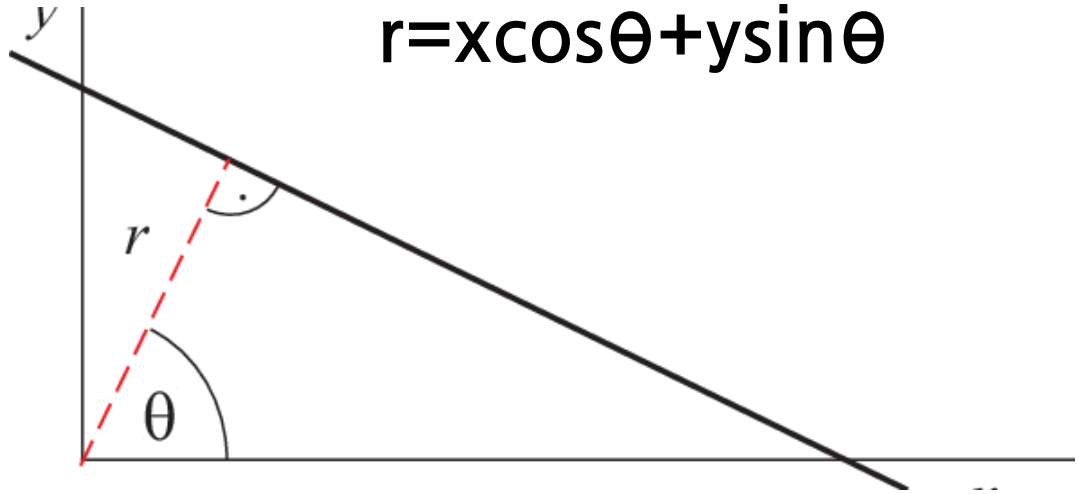




4. 직선 검출

- 허프변환을 통한 직선검출 (에지를 검출한 뒤 에지 영상에서 직선 검출)
  - 에지에서 추출된 모든 점에 대해, r(rho, 직선과 원점 사이의 최소거리), theta(직선이 원점과 이루는 각도)를 기준으로 직선 후보를 추출하여 직선이 많이 교차되는 영역을 직선으로 판단

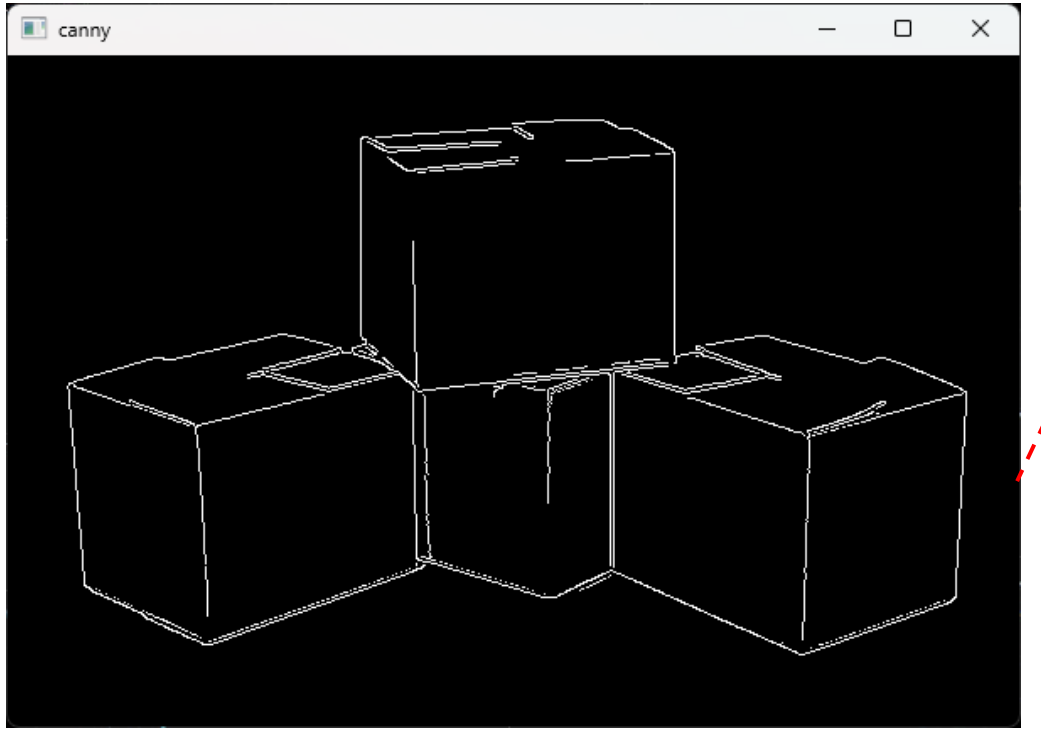


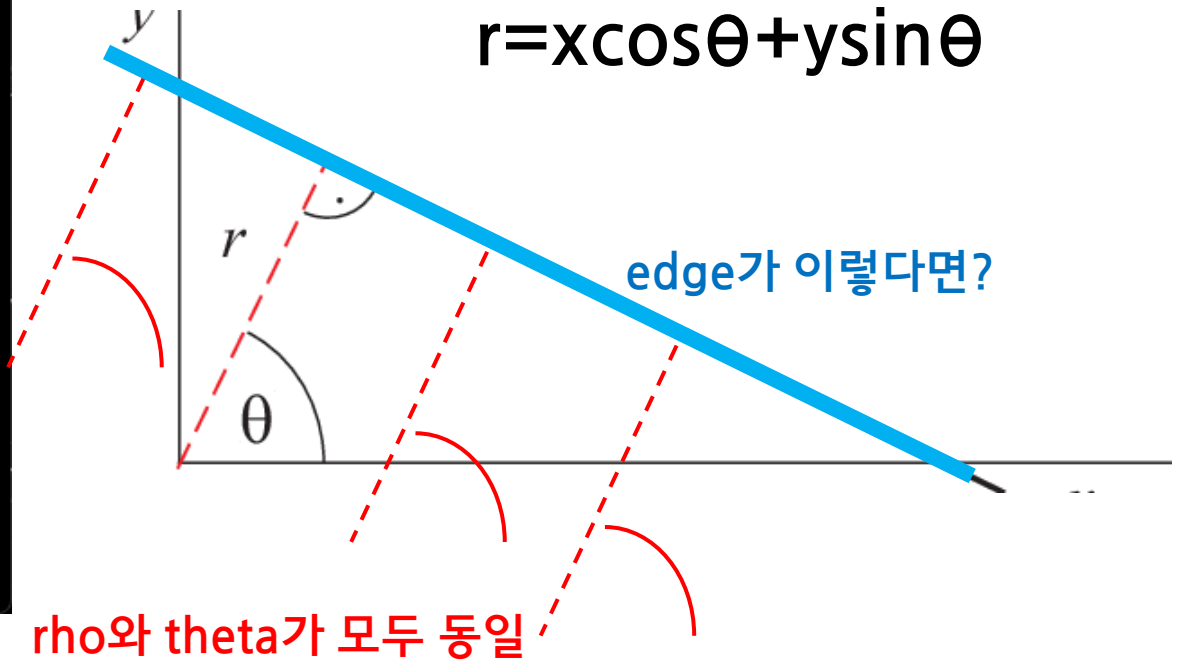


$$r = x\cos\theta + y\sin\theta$$

4. 직선 검출

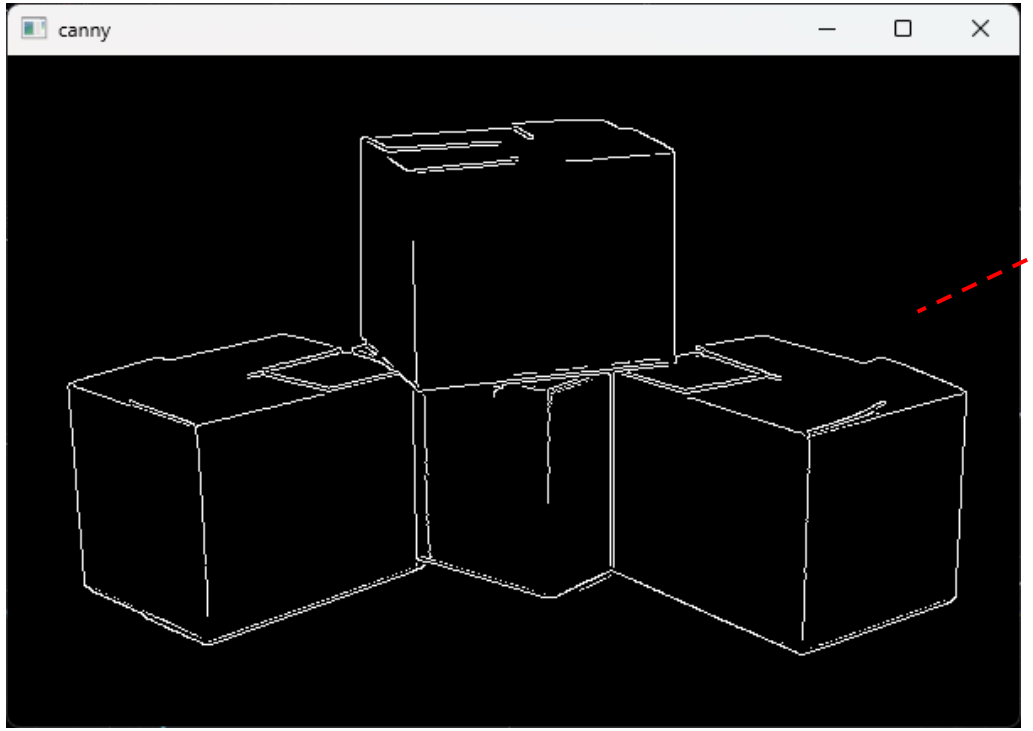
- 허프변환을 통한 직선검출 (에지를 검출한 뒤 에지 영상에서 직선 검출)
  - 에지에서 추출된 모든 점에 대해, r(rho, 직선과 원점 사이의 최소거리), theta(직선이 원점과 이루는 각도)를 기준으로 직선 후보를 추출하여 직선이 많이 교차되는 영역을 직선으로 판단

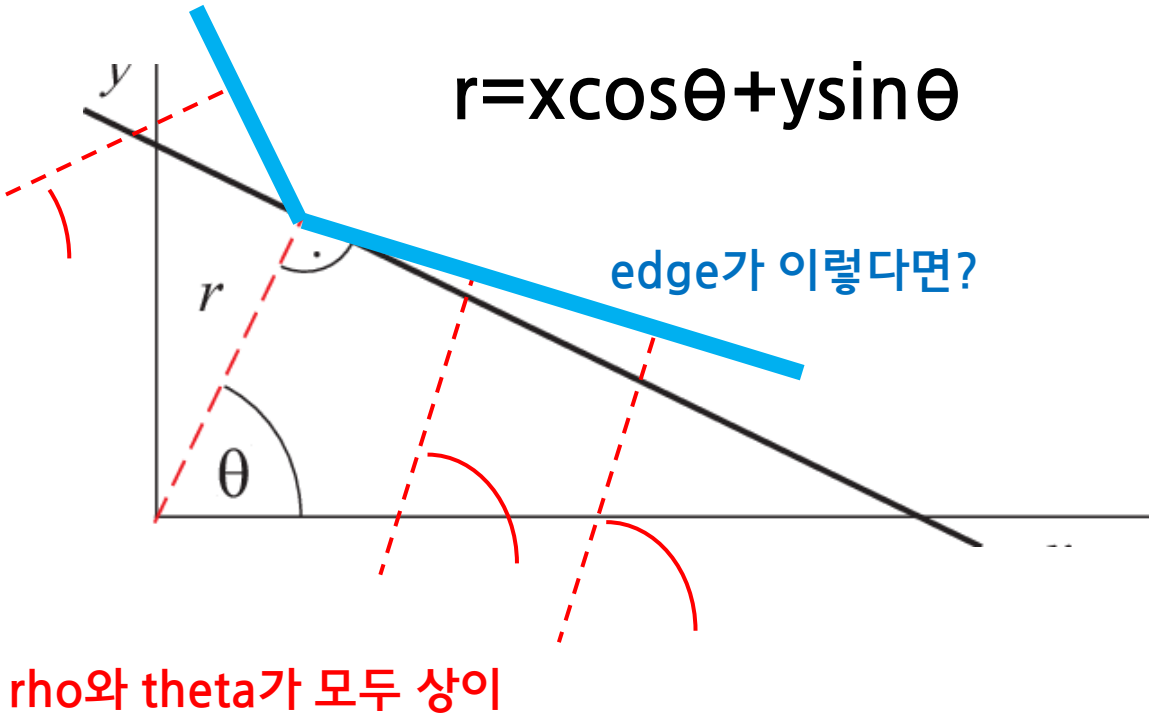




4. 직선 검출

- 허프변환을 통한 직선검출 (에지를 검출한 뒤 에지 영상에서 직선 검출)
  - 에지에서 추출된 모든 점에 대해, r(rho, 직선과 원점 사이의 최소거리), theta(직선이 원점과 이루는 각도)를 기준으로 직선 후보를 추출하여 직선이 많이 교차되는 영역을 직선으로 판단



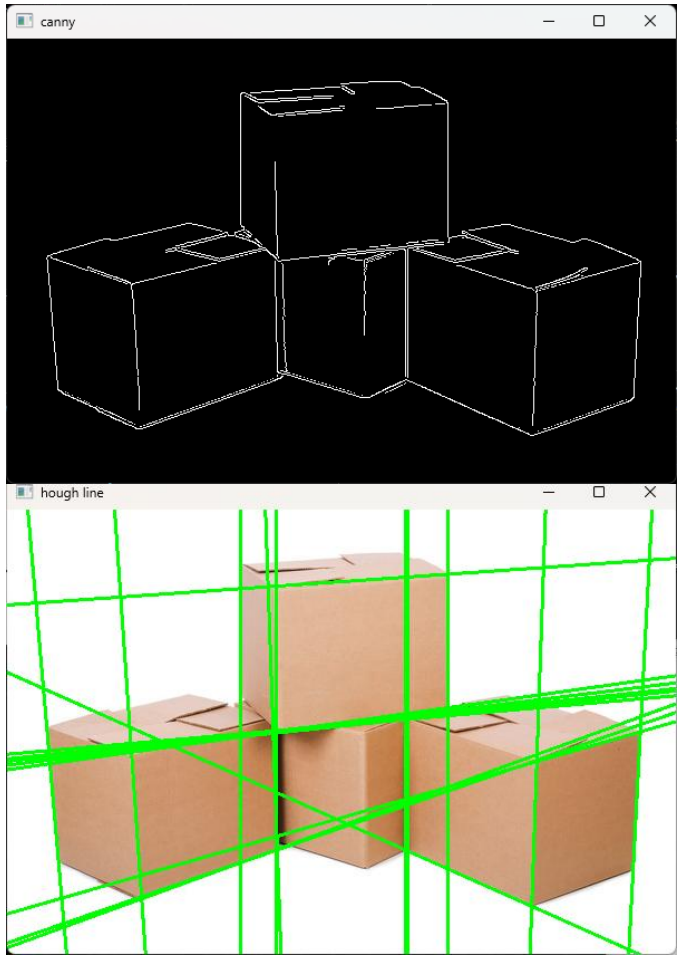


rho와 theta가 모두 상이

4. 직선 검출

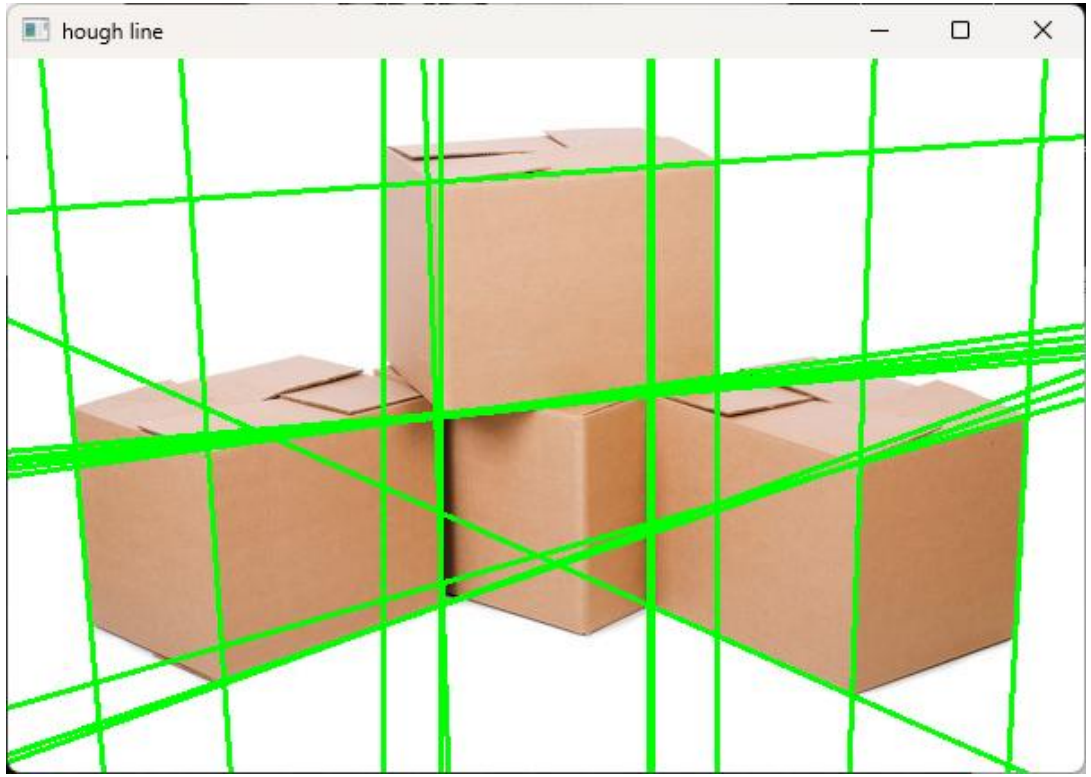
- 허프변환 적용 결과 (캐니에지 영상에 적용)
  - HoughLines함수에 에지이미지, r(0~1), theta, 임계치(직선으로 판단할 임계치) 순으로 입력

```
lines = cv2.HoughLines(canny, 1, np.pi/180, 80)

# 검출된 직선 그리기
for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho

    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
```



실습 9. 모니터 사진의 직선 검출하기

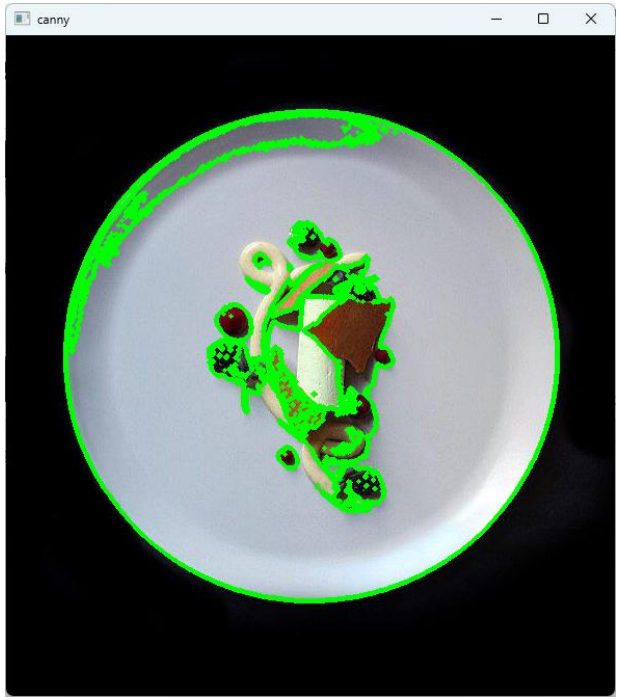
- 휴대폰으로 모니터 화면을 촬영하여, 모니터의 직선이 검출되도록 코드를 작성하기
  - 이미지에 직선이 그려진 형태의 결과물을 실습파일에 첨부



5. 컨투어 검출

- 이진 영상의 경계선(흰픽셀) 영역을 검출하는 기술 (곡선, 직선에 무관하게 경계면을 검출)
  - findContours 함수에 이진이미지, 컨투어 찾는 방법, 근사방법 순으로 입력

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
```



6. 이미지 분할 개요

- 이미지 분할(Image Segmentation)은 이미지에서 서로 다른 객체/영역을 구분하는 기법
  - 이진화 알고리즘을 통한 영역 분할(이미지를 대상과 배경으로 분리), 군집화(Clustering), 대화식 분할(Grabcut), 딥러닝 기반 영역 분할 등의 방법이 존재함

(a) Image (b) Semantic segmentation

Semantic Segmentation
카테고리 마다
서로 다른 픽셀 레이블을 할당

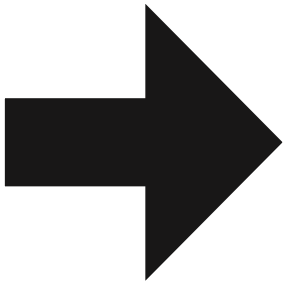
Instance Segmentation
정해진 카테고리 내에서,
서로다른 개체를 다른 레이블로 할당

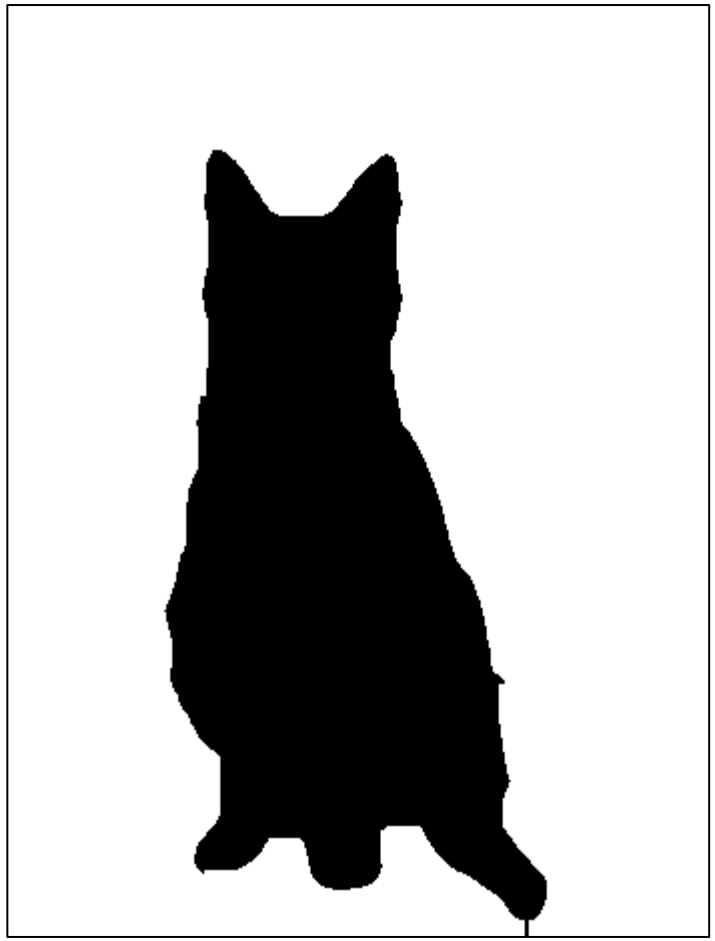
(c) Instance segmentation (d) Panoptic segmentation

Panotic Segmentation
카테고리, 개체 별
서로 다른 픽셀 레이블 할당

대표적인 딥러닝 기반 이미지 분할 예시
https://encord.com/blog/guide-to-semantic-segmentation/

7. 이진화 기반 영역 분할





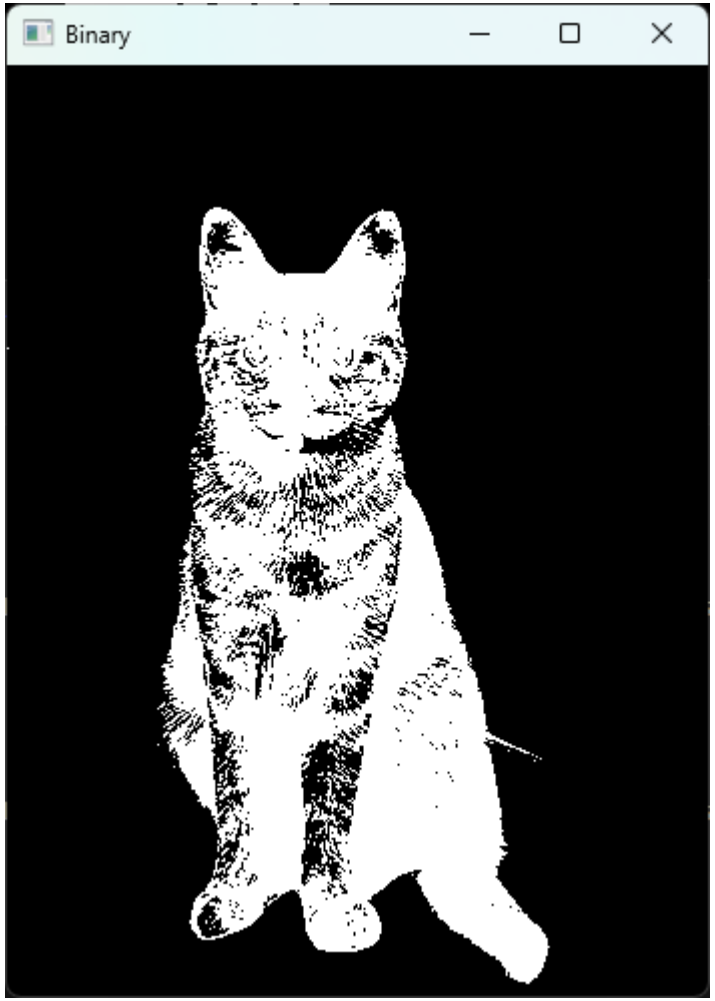
7. 이진화 기반 영역 분할

- 이미지 이진화(Binary Thresholding)는 그레이스케일 이미지를 0과 255로 구분하여 이진화된 이미지를 생성하는 과정
  - retval, threshold_image = cv2.threshold(src, thresh, maxval, type)

  - src: 입력 이미지. 반드시 그레이스케일 이미지여야 함
  - thresh: 임계값. 이 값을 기준으로 픽셀 값이 변환됨
  - maxval: 임계값을 넘는 픽셀에 할당할 최대값 (일반적으로 255 할당)
  - type: 이진화 방식
    - cv2.THRESH_BINARY: 임계값을 넘으면 최대값(255), 아니면 0으로 설정
    - cv2.THRESH_BINARY_INV: 임계값을 넘으면 0, 아니면 최대값(255)으로 설정

실습 10. 고양이 이미지를 이진화하기

- 고양이(cat.jpg) 이미지를 화면과 유사하도록 이진화하기
  - 이진화에 사용한 파라메터 값과 결과 이미지를 실습 파일에 입력하기



8. 연결요소와 모폴로지 연산

- 연결요소 분석은 이미지에서 서로 연결된 픽셀들의 그룹을 찾는 과정으로, 일반적으로 이진화된 이미지에서 사용되며, 흰색 또는 검은색으로 연결된 픽셀들이 각각 독립적인 객체로 인식됨

- 연결요소 분석 방법
  - N연결성 내 픽셀이 닿아 있는지 확인
  - 4-연결성: 상하좌우 4 방향 연결여부
  - 8-연결성: 대각선 포함 8 방향 연결여부

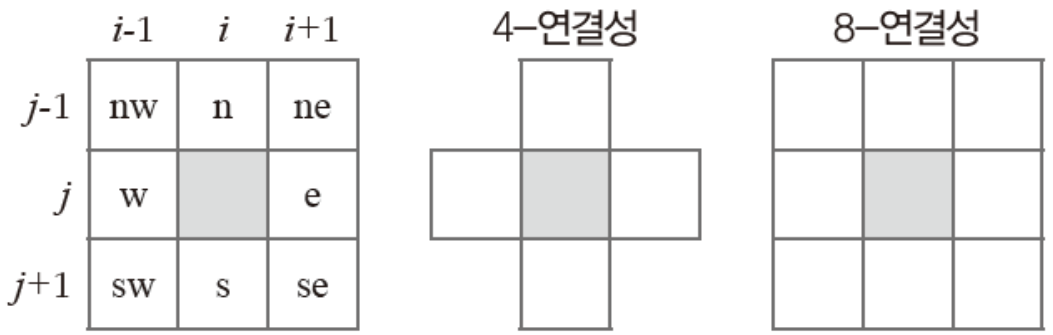


그림 3-10 화소의 연결성

[예시 3-2] 연결 요소

[그림 3-11]에서 (a)는 입력 이진 영상이고, (b)와 (c)는 각각 4-연결성과 8-연결성으로 찾은 연결 요소다. 연결 요소는 고유한 정수 번호로 구분한다.

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	0
0	0	0	1	1	0	1	0

(a) 입력 이진 영상

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

(b) 4-연결성으로 찾은 연결 요소

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

(c) 8-연결성으로 찾은 연결 요소

그림 3-11 연결 요소 찾기

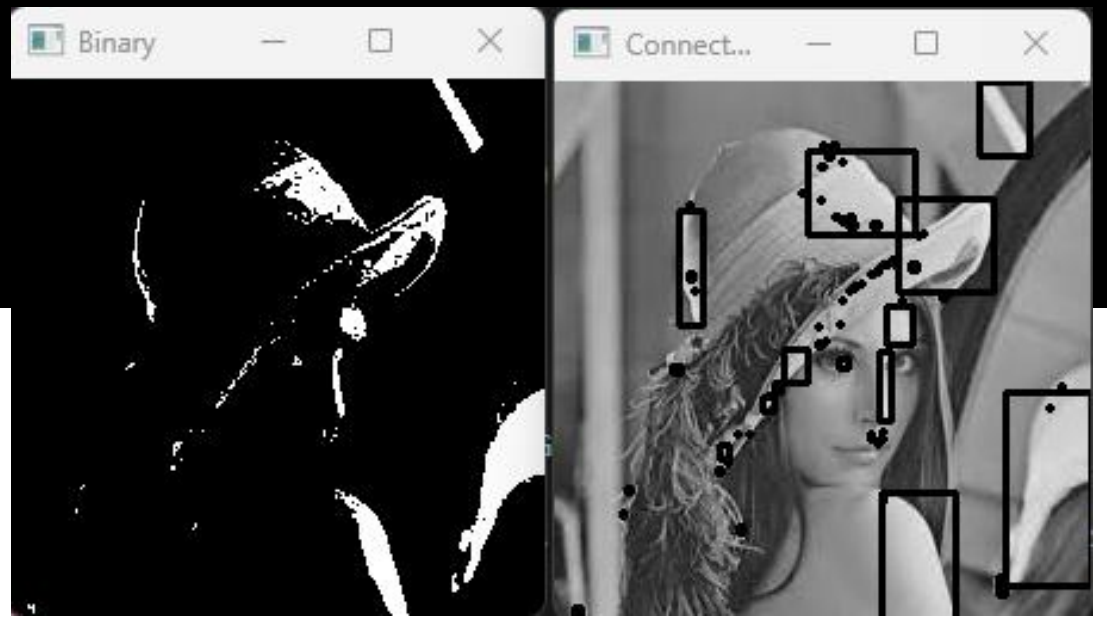
8. 연결요소와 모폴로지 연산

- cv2.connectedComponentsWithStats 함수를 사용하여 연결된 대상 검출
  - 이진(binary) 이미지에 함수를 적용
  - connectivity 값으로 4 또는 8을 할당 (N연결요소를 의미)
  - cnt; 객체가 몇 개인지, labels; 객체에 맞게 할당된 label, stats; 객체마다 bbox 정보, entroids; 무게중심 좌표

```
cnt, labels, stats, centroids = cv2.connectedComponentsWithStats(binary_image, connectivity=4)
```

8. 연결요소와 모폴로지 연산

```python
import cv2
img = cv2.imread('lena.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, binary = cv2.threshold(img, 200, 255, cv2.THRESH_BINARY)
cnt, labels, stats, centroids = cv2.connectedComponentsWithStats(binary, connectivity=8)

for i in range(1, cnt):
    (x, y, w, h, area) = stats[i]
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('Binary', binary)
cv2.imshow('Connected Components', img)

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



8. 연결요소와 모폴로지 연산

- 모폴로지 연산(Morphological Operations)은 이미지를 처리할 때 모양이나 구조를 기반으로 픽셀 값을 변환하는 기법
  - 구조 요소(structuring element)를 이용하여 영역의 모양을 조작 (직사각형, 십자형, 타원형 등)

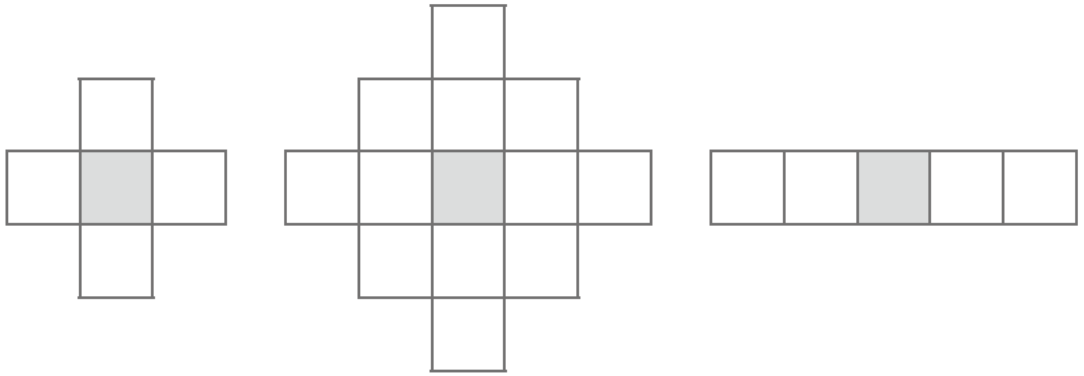
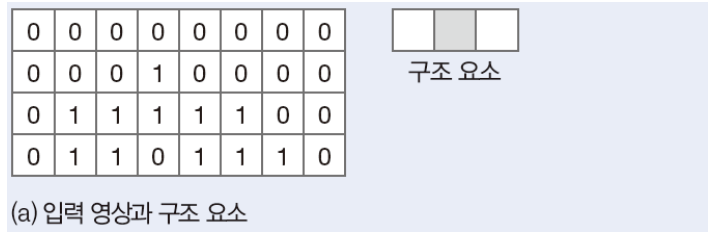


그림 3-12 모폴로지가 사용하는 구조 요소

- 주요 모폴로지 연산
  - 팽창(dilation, dilate): 객체가 확장되며, 작은 구멍이나 경계선이 채워짐
  - 침식(erosion, erode): 객체가 줄어들며, 작은 노이즈가 제거됨
  - 열림(opening): 작은 노이즈가 제거되며, 객체는 원래 크기로 유지됨
  - 닫힘(closing): 작은 구멍이 채워지고, 객체의 경계선이 부드럽게 처리됨

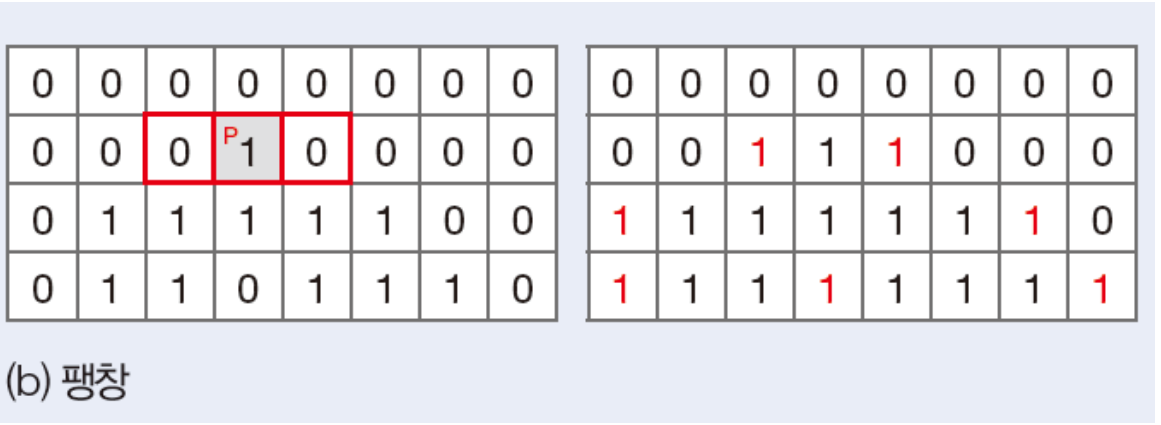
8. 연결요소와 모폴로지 연산

- 팽창(Dilate) 연산
  - 팽창은 객체의 경계선을 따라 픽셀을 추가하는 연산으로, 구조 요소 내에 하나라도 1이 있으면 중앙 픽셀이 1로 설정됨
  - dilated = cv2.dilate(binary_image, kernel, iterations=1)

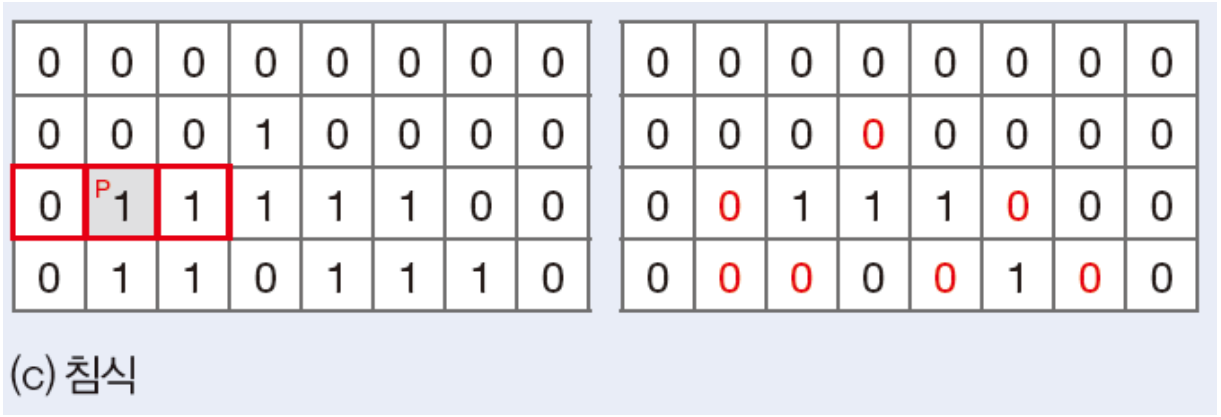


(a) 입력 영상과 구조 요소

- 침식(Erode) 연산
  - 침식은 객체의 경계선을 따라 픽셀을 제거하는 연산으로, 구조 요소 내에 모두가 1인 경우에만 중앙 픽셀이 유지되며, 그렇지 않으면 0이 됨
  - eroded = cv2.erode(binary_image, kernel, iterations=1)



(b) 팽창

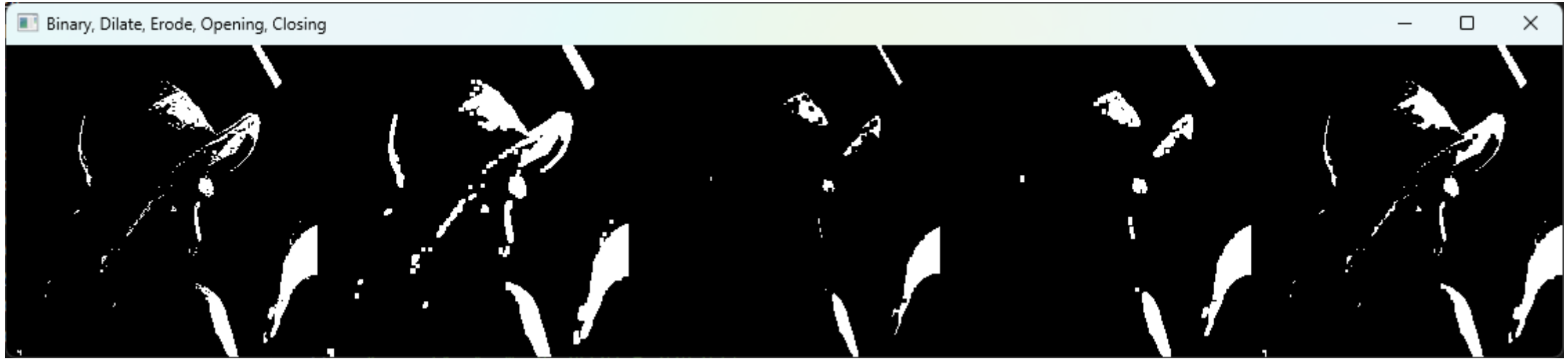


(c) 침식

8. 연결요소와 모폴로지 연산

- 열림(Opening) 연산
  - 침식 후에 팽창을 수행하는 연산으로, 객체의 모양을 유지하면서 노이즈를 제거하는 데 효과적임
  - opened = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)

- 닫힘(Closing) 연산
  - 팽창 후에 침식을 수행하는 연산으로, 객체 내부의 작은 구멍을 메우거나 객체의 경계선을 부드럽게 만
  - closed = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)



Binary이미지 Dilate(팽창)결과 Erode(침식)결과 Opening(열림)결과 Close(닫힘)결과

9. 슈퍼픽셀 알고리즘

- 슈퍼픽셀 생성을 위한 SLIC(Simple Linear Iterative Clustering) 알고리즘
  - 슈퍼픽셀(Superpixel)은 원본 이미지의 픽셀들을 클러스터링하여 유사한 특성을 가진 픽셀들을 그룹화한 영역을 의미
  - 슈퍼픽셀을 생성하기 위한 SLIC는 K-means 군집화에 기반하여, 인접한 픽셀들 사이에서 유사성을 계산하고 그들을 슈퍼픽셀로 묶는 알고리즘
  - 초기 클러스터 중심 설정 → 픽셀 할당 및 거리 계산 → 클러스터 중심 업데이트 → 반복

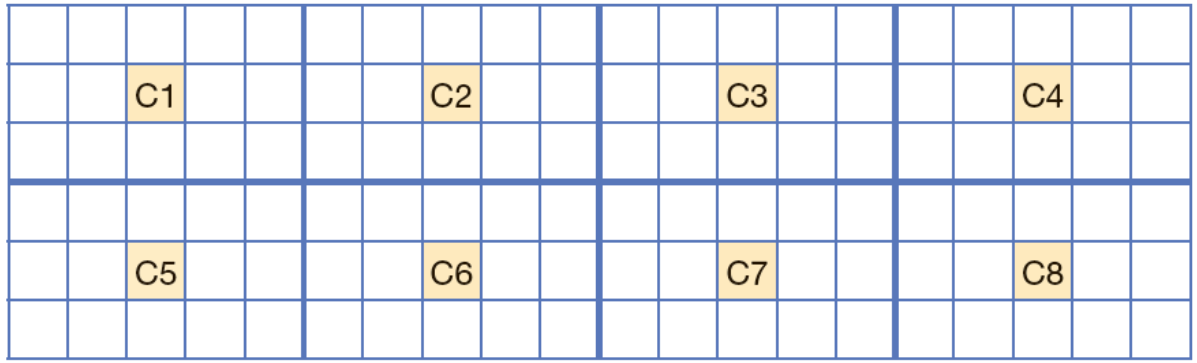
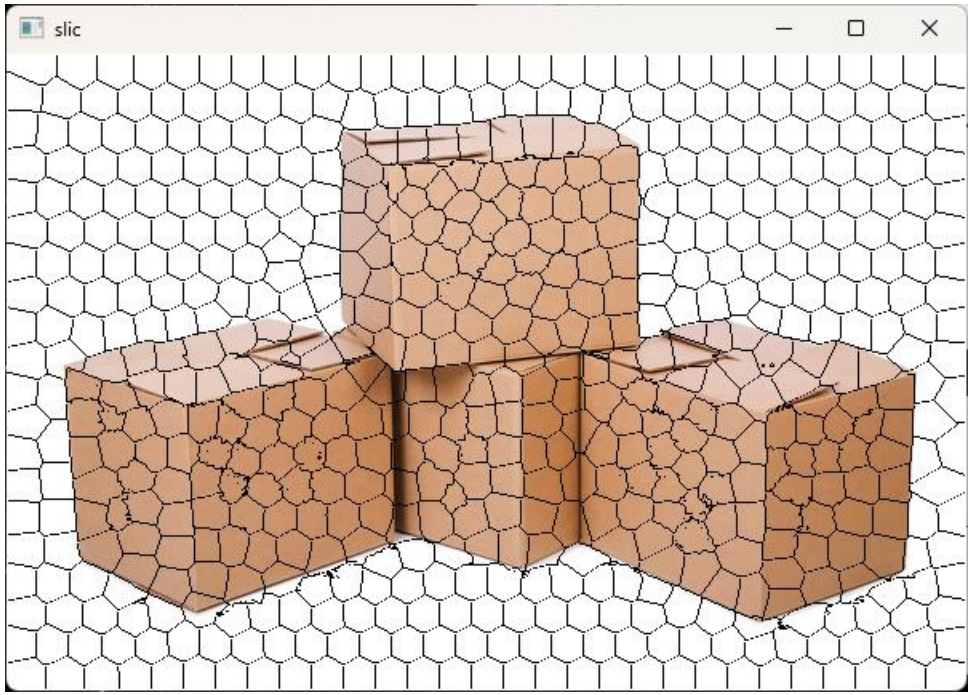


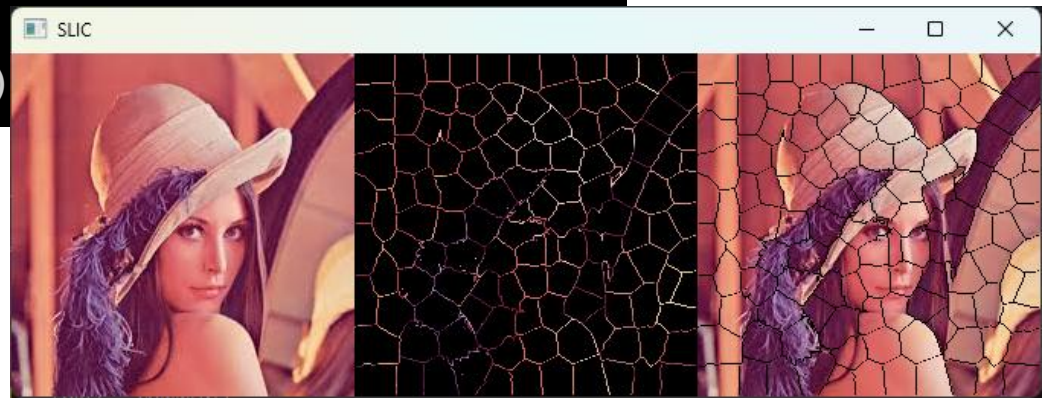
그림 4-15 SLIC 알고리즘의 초기 군집 중심



9. 슈퍼픽셀 알고리즘

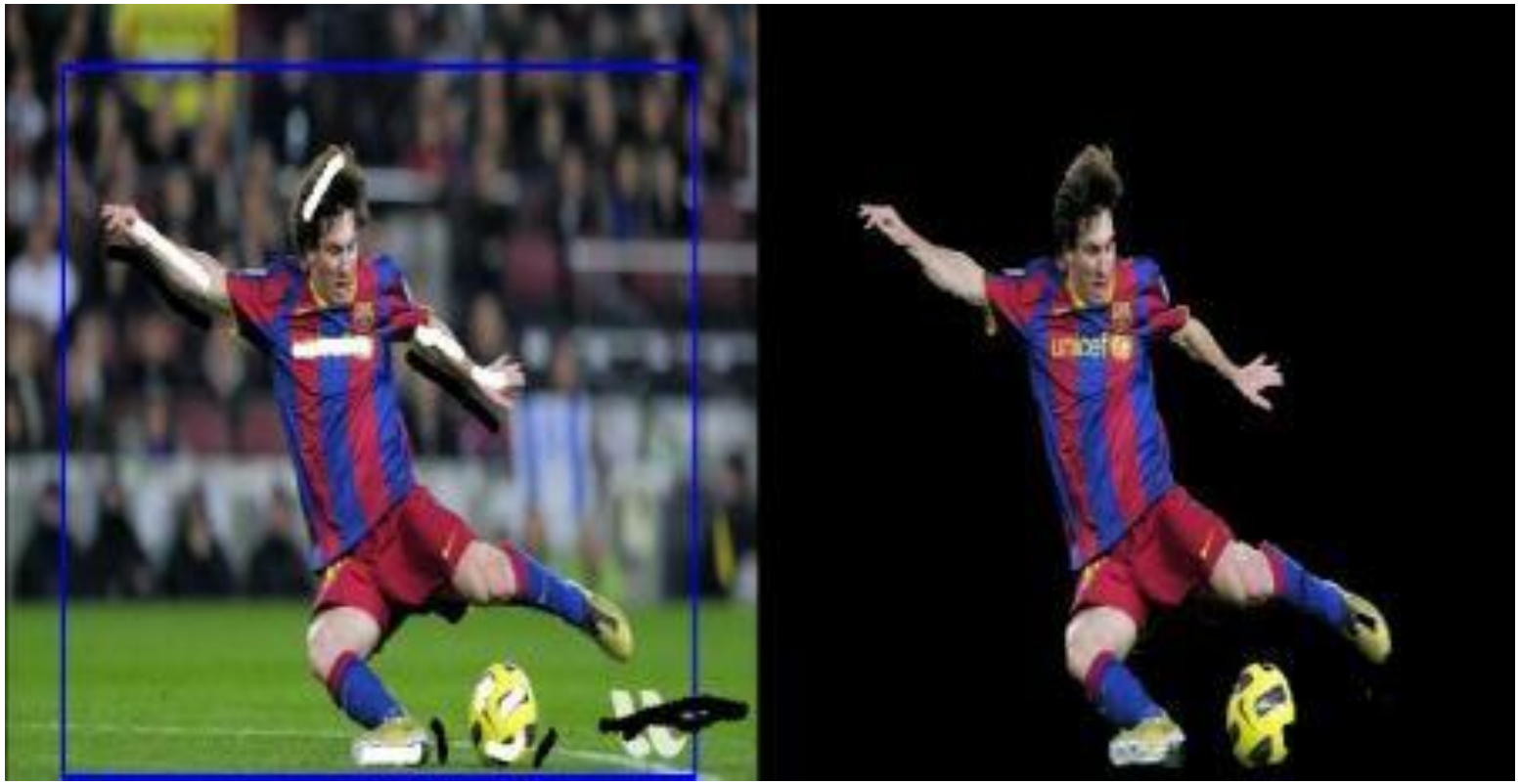
- SLIC 알고리즘 적용을 위해, pip install opencv-contrib-python로 추가모듈 설치 필요함

```
import cv2
img = cv2.imread('lena.jpg')

slic = cv2.ximgproc.createSuperpixelSLIC(img, region_size=20, ruler=10.0)
slic.iterate(10) # 반복 횟수 설정
# 슈퍼픽셀의 마스크를 가져오기
mask_slic = slic.getLabelContourMask(thick_line=True)
# 결과를 시각화하기 위해 마스크 색 변경
mask_slic_inv = cv2.bitwise_not(mask_slic)
image_result = cv2.bitwise_and(img, img, mask=mask_slic_inv)
image_contours = cv2.bitwise_and(img, img, mask=mask_slic)

res = cv2.hconcat([img, image_contours, image_result])
```



10. 그랩컷(Grabcut)

- 그랩컷은 사용자가 제공한 최소한의 정보(대략적인 객체의 경계)를 기반으로 이미지에서 객체와 배경을 분리하는 대화식 분할 알고리즘
  - 대화식 분할: 사용자가 직접적으로 분할 과정에 참여하며, 일반적으로 마우스 이벤트 같은 사용자 인터페이스를 통해 전경과 배경을 지정하는 등의 상호 작용을 통해 이미지 분할을 진행하는 작업



10. 그랩컷(Grabcut)

- 그랩컷 기능은 PPT에 내장됨

10. 그랩컷(Grabcut)

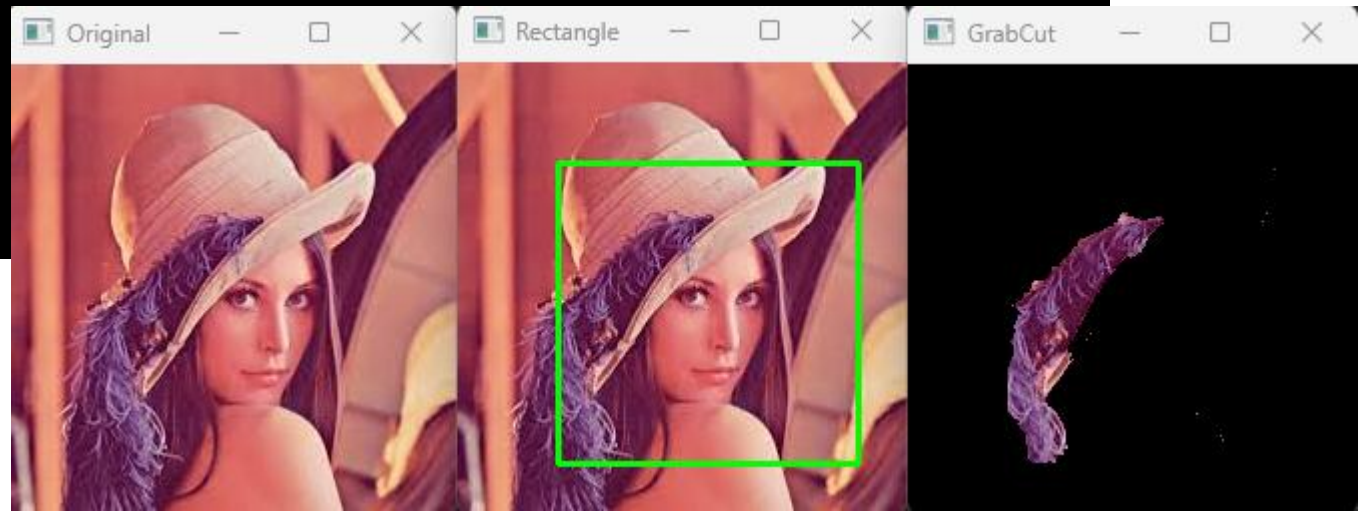
- 그랩컷을 적용하기 위해, 초기 마스크와 배경, 전경모델을 생성해야 함
  - 배경, 전경모델의 크기가 65인 이유: (가중치(1) + 평균벡터(3) + 공분산행렬(9)) * 가우시안 분포 수(5)
- 사용자가 임의로 전경영역을 임의로 지정해야 함

```
import cv2
import numpy as np
img = cv2.imread('lena.jpg')
print(img.shape)

# 초기 마스크 생성
mask = np.zeros(img.shape[:2], np.uint8)
# 배경과 전경 모델 생성
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)
# 전경이 될 영역을 직사각형으로 지정
rect = (50, 50, 150, 150) # 임의 조정된 영역
rect_img = cv2.rectangle(img.copy(), (50, 50), (200, 200), (0, 255, 0), 2)
cv2.imshow('Rectangle', rect_img)
```

10. 그랩컷(Grabcut)

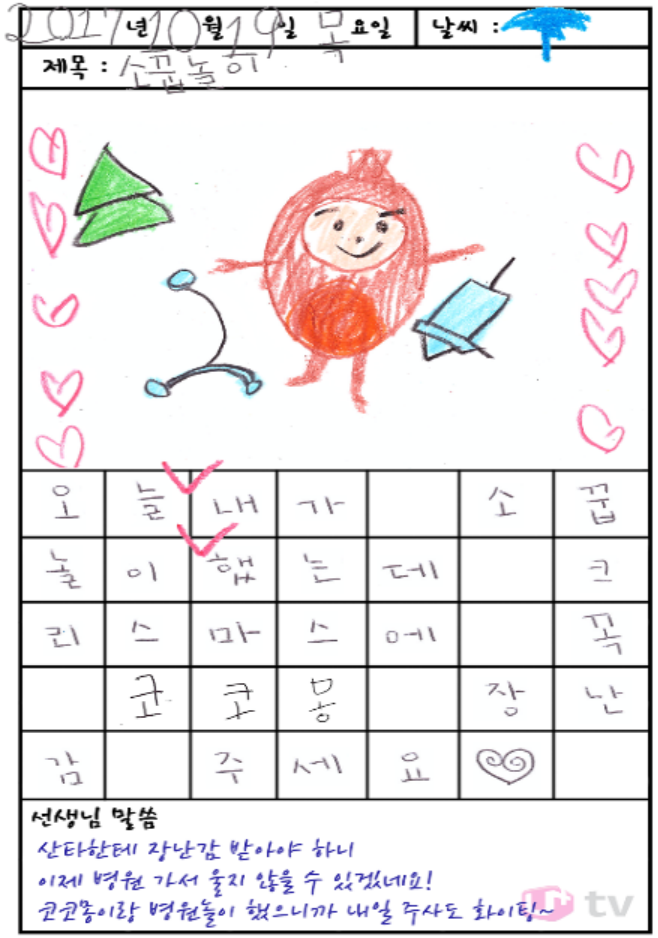
- grabCut 함수를 통해 영역 분할 (숫자 5는 반복 횟수를 의미)

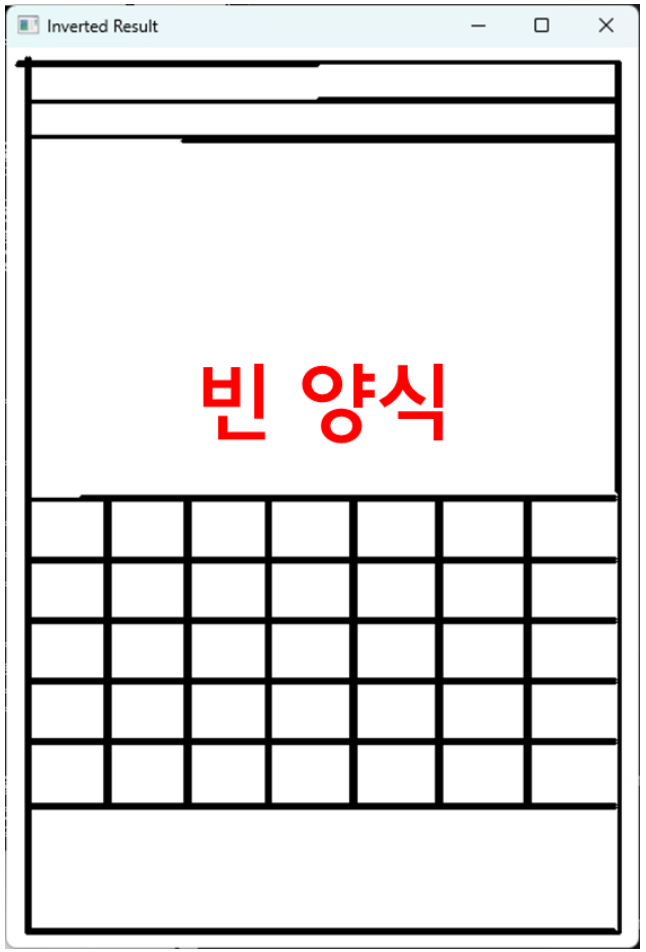
```
# GrabCut 알고리즘 실행
cv2.grabCut(img, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)
# 마스크를 2D 이미지로 변환
mask_2d = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
# 마스크를 3D 이미지로 확장하고 원본 이미지에 적용
mask_color = mask_2d[:, :, np.newaxis]
img_result = img * mask_color
# 결과 출력
cv2.imshow('Original', img)
cv2.imshow('GrabCut', img_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



실습 11. 일기장 내용 숨기기

- 학습한 영상처리 기법을 활용하여, 일기장의 내용은 가리고 양식만 남겨보기
  - 생성된 결과 이미지를 실습파일에 추가하기

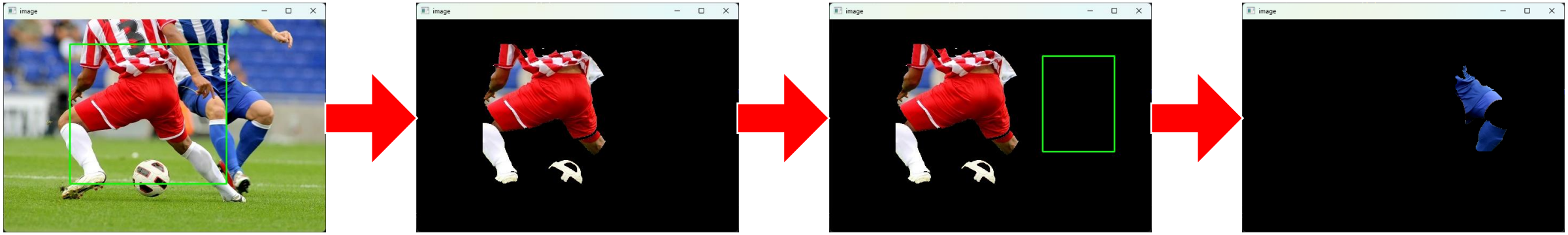




실습 12. 반복해서 이미지 분할을 수행하는 프로그램 만들기

- 마우스 콜백 함수와, 반복문을 활용하여 계속해서 Grabcut을 실행하는 프로그램 만들기
  - 소스코드를 실습파일에 작성하기

```
While True:
    cv2.imshow('image', img)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:  # ESC 키 누르면 종료
        break
    elif k == ord('r') and rect_over: # r 키 누르면 GrabCut 알고리즘 적용
        ...
```



사각형 그리고 r키 입력

사각형 그리고 r키 입력