Computer Vision

Lecture 04 영상처리 기술

황선희

동양미래대학교
DONGYANG MIRAE UNIVERSITY

영상처리 (Image Processing)

- 영상처리란?
  - 특정 목적을 달성하기 위해 원래 영상을 개선된 새로운 영상으로 변환하는 작업

- 영상처리 기술 사용 예시
  - 화질 개선 목적 (Super Resolution, Noise Removal, Deblur 등)
    - 구작 영상(무한도전 등) 화질 개선, 촬영된 이미지 노이즈 제거, 흔들린 영상 개선, 그림자/안개 제거 등
  - 컴퓨터 비전 기술 개발 시, 영상처리 기술을 이미지 전처리과정(Pre-processing)에 활용
    - 영상처리 된 이미지로부터, 차량의 번호판 식별/ 의료영상 내 병변 위치 찾기 등

(a) 안개 낀 도로 영상 (b) 히스토그램 평활화로 개선한 영상

그림 3-1 영상 처리로 화질 개선

프로그래밍 실습 문제 4 - 예시코드

```
import cv2

# 동영상 파일 또는 이미지 시퀀스에서 프레임 불러오기
cap = cv2.VideoCapture('movingobj01.mp4')

# 첫 번째 프레임을 읽고 배경으로 설정
ret, base_frame = cap.read()
if not ret:
    print("비디오를 읽을 수 없습니다.")
    cap.release()
    exit()

# 시퀀스 사진의 기본 베이스 이미지 생성 (초기 프레임)
sequence_image = base_frame.copy()
gray_base = cv2.cvtColor(base_frame, cv2.COLOR_BGR2GRAY)


...
```

프로그래밍 실습 문제 4 - 예시코드

```
# 각 프레임을 하나씩 읽어가면서 처리
frame_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    if frame_count % 15   == 0:  # 너무 많은 프레임을 합치지 않도록 간격을 둠


        ...


        sequence_image = cv2.add(bg, fg)

# 시퀀스 이미지 출력
cv2.imshow('Sequence Image', sequence_image)
cv2.imwrite('motion_output.jpg', sequence_image)

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

프로그래밍 실습 문제 4 - 예시코드

```
if frame_count % 15    == 0:  # 너무 많은 프레임을 합치지 않도록 간격을 둠
        # 그레이스케일로 변환하여 차이 검출
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # 두 이미지 간의 차이 계산
        diff = cv2.absdiff(gray_base, gray_frame)

        # 차이가 있는 영역을 찾아서 이진화
        _, mask = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)

        # 차이가 있는 영역만 합성 (색상 공간을 유지한 상태로)
        mask_inv = cv2.bitwise_not(mask)
        fg = cv2.bitwise_and(frame, frame, mask=mask)
        bg = cv2.bitwise_and(sequence_image, sequence_image, mask=mask_inv)

        # 모션 이미지 업데이트
        sequence_image = cv2.add(bg, fg)
```

목차

1. 영상 획득과 표현

- 우리가 보는 영상 데이터
  - 주로 모바일, 데스크탑, 태블릿PC, TV 등 디지털 디바이스를 통해 이미지 및 동영상 감상

- 디지털 영상이란?
  - 아날로그 신호(Continuous)로 표현되는 장면을 디지털(Discrete)화 하여 표현한 영상

- 디지털 영상활용 분야
  - 영상 감상에 활용
  - 컴퓨터 비전 기술 개발에 활용

1. 영상 획득과 표현

- 디지털 변환
  1. M*N 으로 샘플링 (sampling): 아날로그 영상을 디지털화 하기 위해, 일정 간격으로 나누어 수치화
  2. L 단계로 양자화 quantization: 샘플링을 통해 얻은 수치를 다시 정해진 수준으로 변환

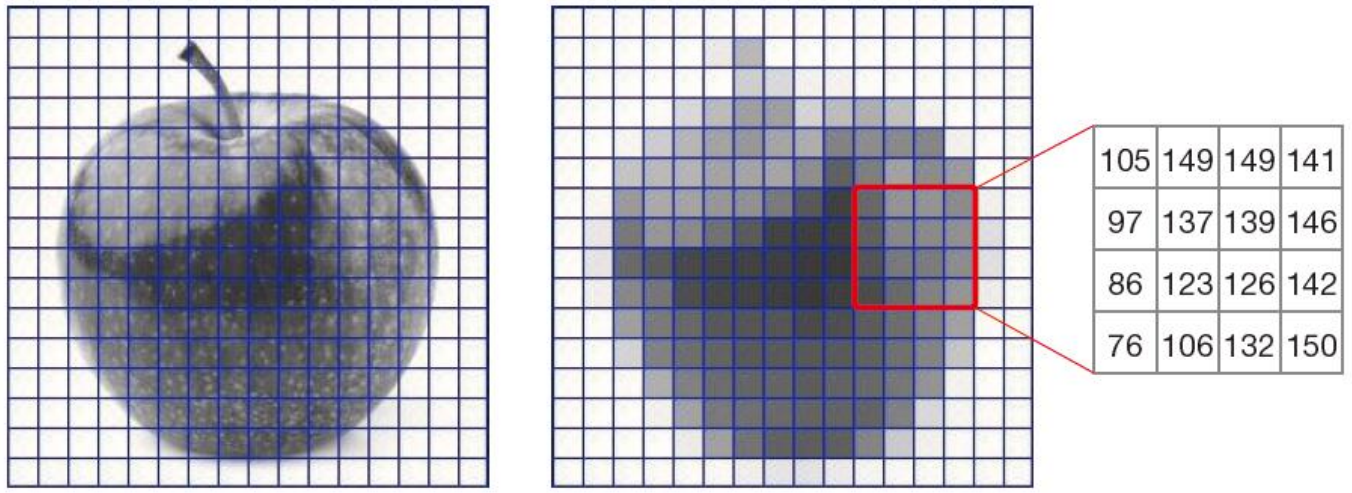
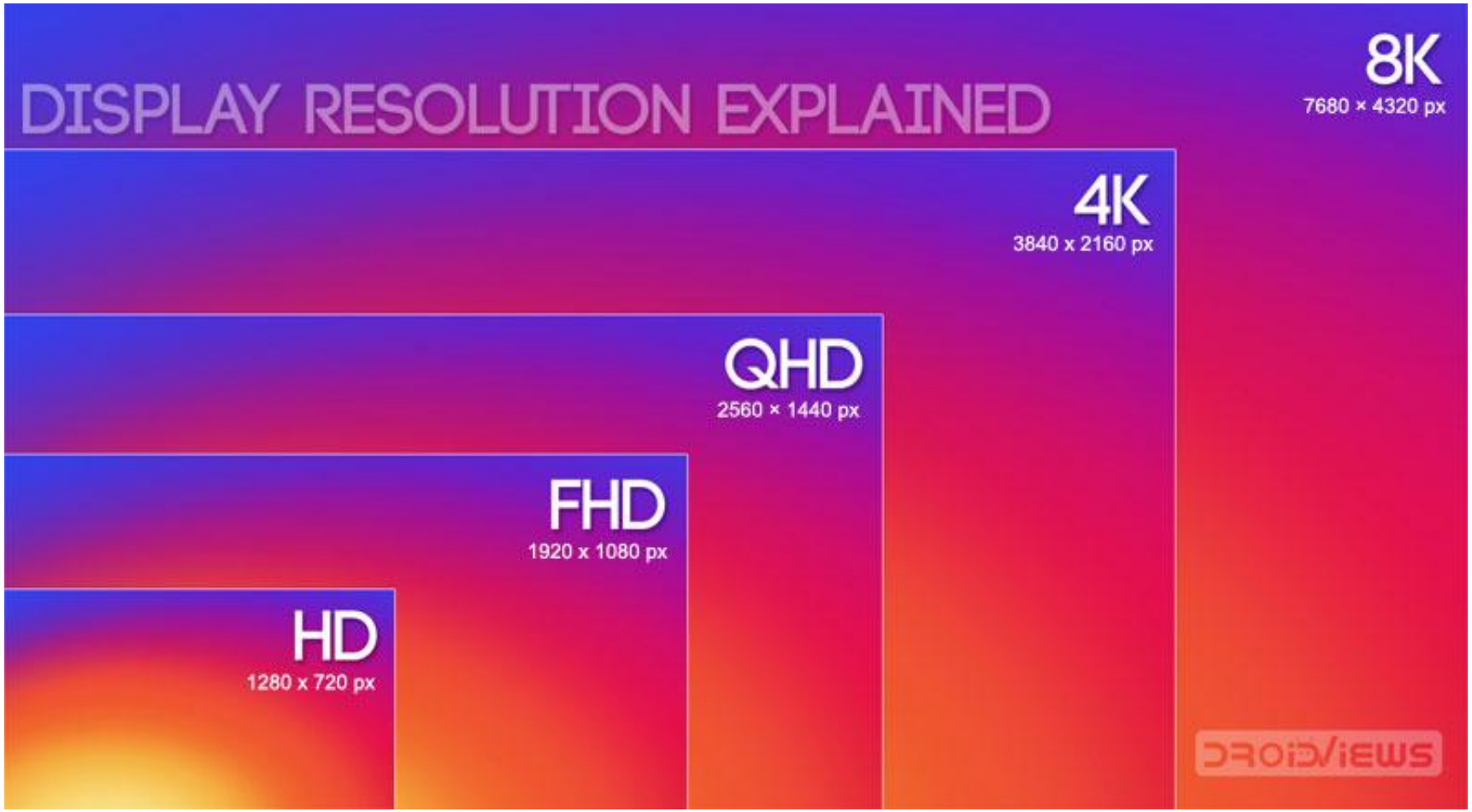


그림 3-3 피사체가 반사하는 빛 신호를 샘플링과 양자화를 통해 디지털 영상으로 변환

TIP 엄밀히 말해 해상도는 물리적 단위 공간에서 식별 가능한 점의 개수를 뜻한다. 예를 들어 인치 당 점의 개수를 뜻하는 dpi(dot per inch)는 해상도다. 이 책에서는 화소의 개수를 해상도라고 부른다.

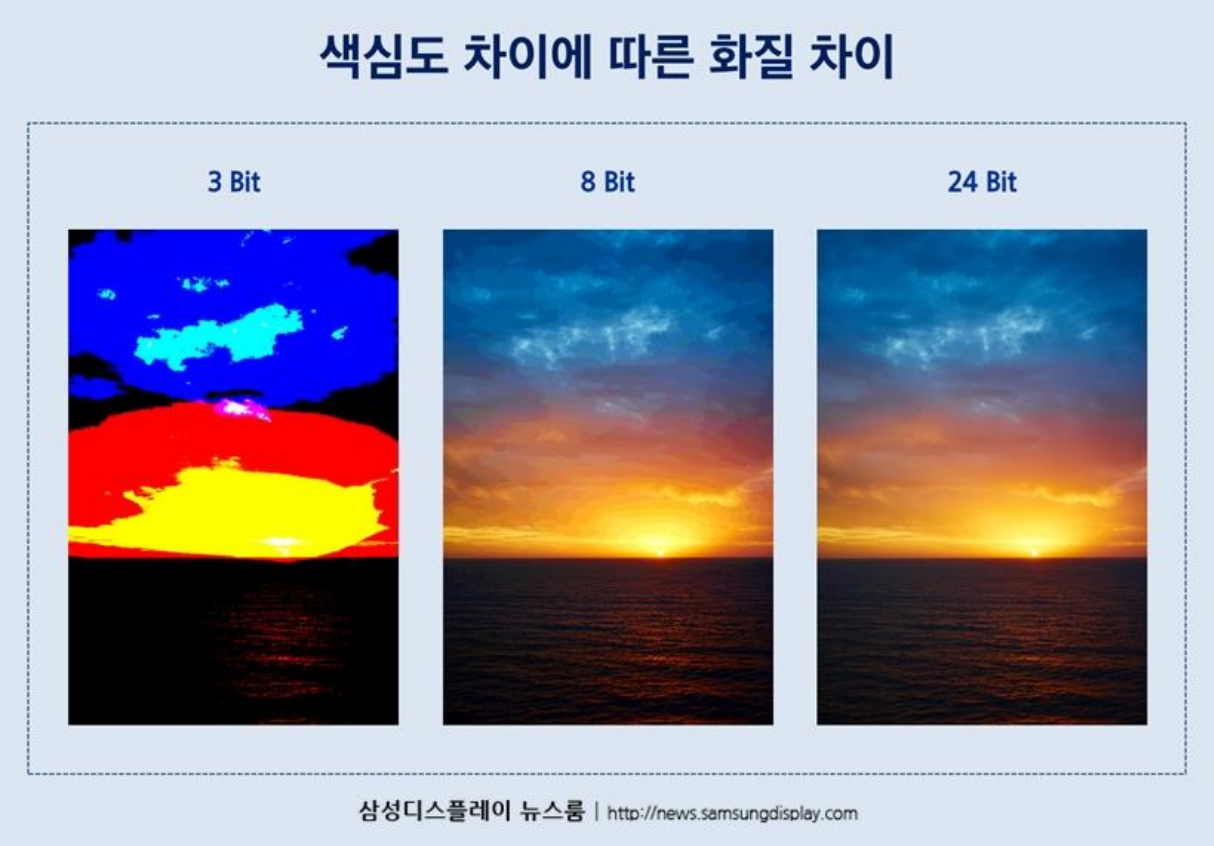
1. 영상 획득과 표현

- 디지털 영상의 해상도 표준
  - 해상도의 표준 규격은 가로 및 세로 픽셀을 기준으로 (샘플링 수준에 따른) 화소 수를 표기



1. 영상 획득과 표현

- 영상 내 색상의 깊이를 표현하는 색 심도 (Color Depth)
  - 이미지를 구성하는 픽셀당 표현 가능한 컬러 정보를 의미
  - 한 픽셀이 표현하는 bit 수에 따라 표현 가능한 컬러 범위가 상이함



색심도(Bit)에 따른 계조 표현

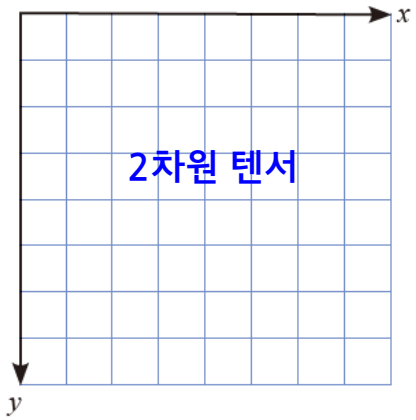
Bit	계조 표현력 (Sub-Pixel 별)	1 Pixel 별 표현 가능 색상의 수 * 계산 : Bit(R) x Bit(G) x Bit(B)
1 Bit = 2^1		$2^1 \times 2^1 \times 2^1 = 8$
2 Bit = 2^2		$2^2 \times 2^2 \times 2^2 = 64$
3 Bit = 2^3		$2^3 \times 2^3 \times 2^3 = 512$
6 Bit = 2^6		$2^6 \times 2^6 \times 2^6 = 262,144$
8 Bit = 2^8		$2^8 \times 2^8 \times 2^8 = 16,777,216$

1. 영상 획득과 표현

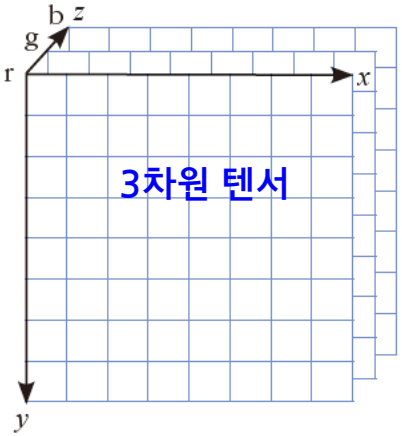
- 주사율 (Refresh Rate)
  - 동영상 레벨에서, 초당 표현 가능한 장면 수를 나타내는 수치 (인간의 눈 한계는 천차만별)

1. 영상 획득과 표현

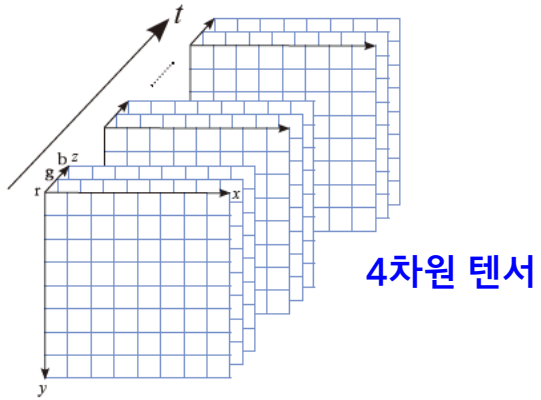
- 다양한 종류의 영상 데이터



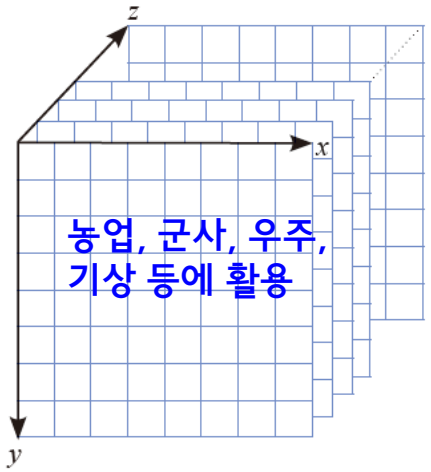
(a) 명암 영상



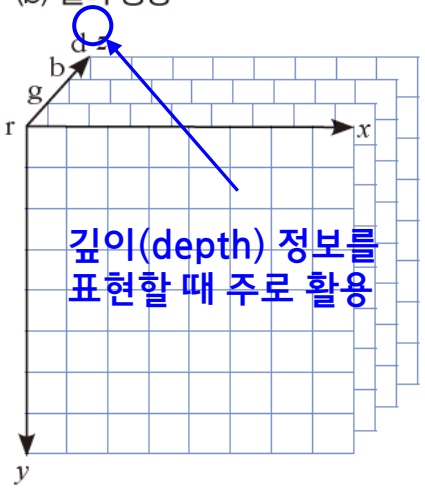
(b) 컬러 영상



(c) 컬러 동영상



(d) 다분광/초분광/MR/CT 영상



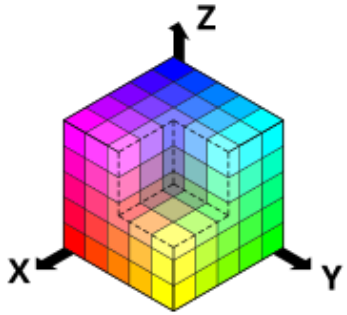
(e) RGB-D 영상

(f) 점 구름 영상 (Point Cloud)

그림 3-5 다양한 형태의 디지털 영상

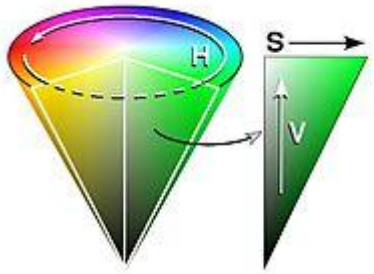
2. 영상 색 공간

- 색의 표현 및 재현방식을 정의하는 색 공간

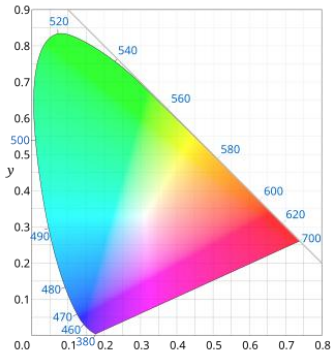


RGB 색 공간
빨강(Red), 녹색(Green), 파랑(Blue)

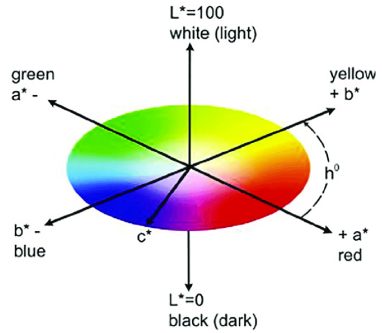
CMYK 색 공간 (인쇄용 4색)
옥색(Cyan), 자홍색(Magenta), 노랑(Yellow), 검정(Black)



HSV 색 공간
색상(Hue), 채도(Saturation), 명도(value)



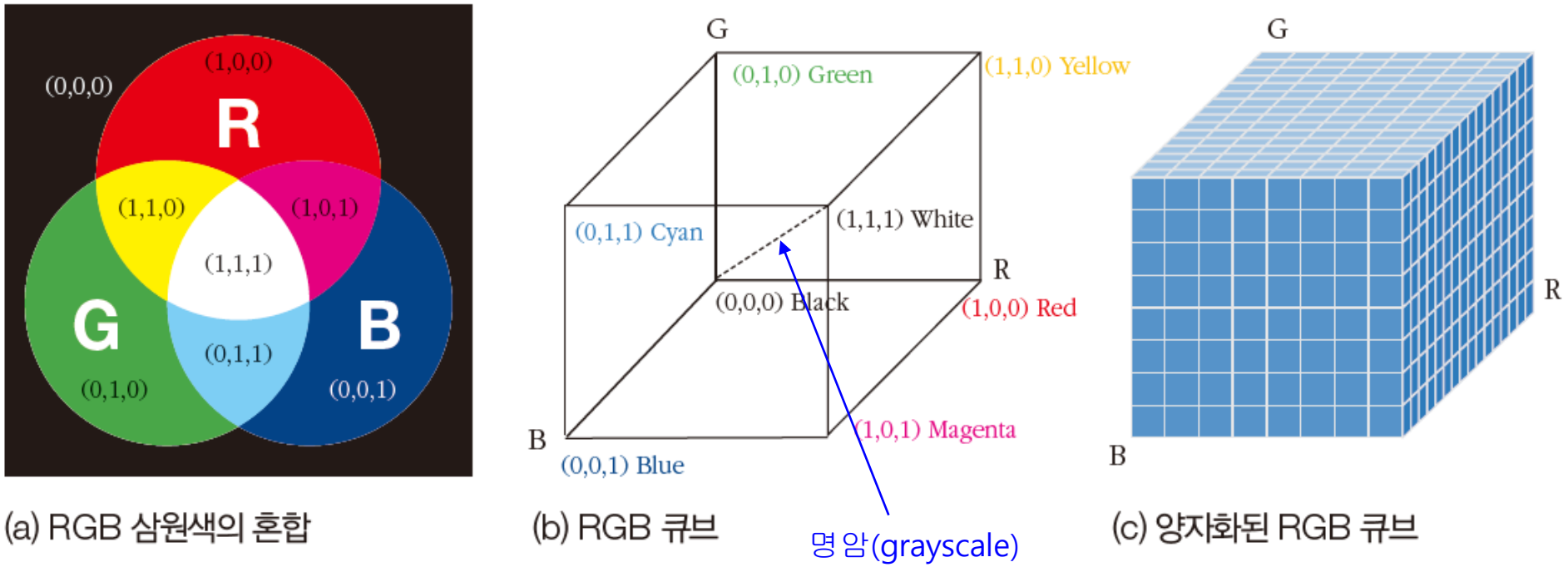
CIE 색 공간
CIE: 국제조명위원회



CIELAB 색 공간
밝기(L), 초록/빨강(a*), 노랑/파랑(b*)

2. 영상 색 공간

- RGB 컬러 모델
  - 빨강(Red), 초록(Green), 파랑(Blue)의 세 가지 색상을 기본 색상으로 사용하는 컬러 모델
  - 모니터, TV, 카메라 등 다양한 디스플레이 장치에서 주로 사용됨



(a) RGB 삼원색의 혼합 (b) RGB 큐브 (c) 양자화된 RGB 큐브

그림 3-6 RGB 컬러 공간

2. 영상 색 공간

- HSV 컬러 모델
  - 색상(Hue), 채도(Saturation), 명도(Value)의 세 가지 속성을 기본으로 사용하는 컬러 모델
  - 색상인식 관련 작업에 주로 사용됨 (그래픽, 디자인 등)

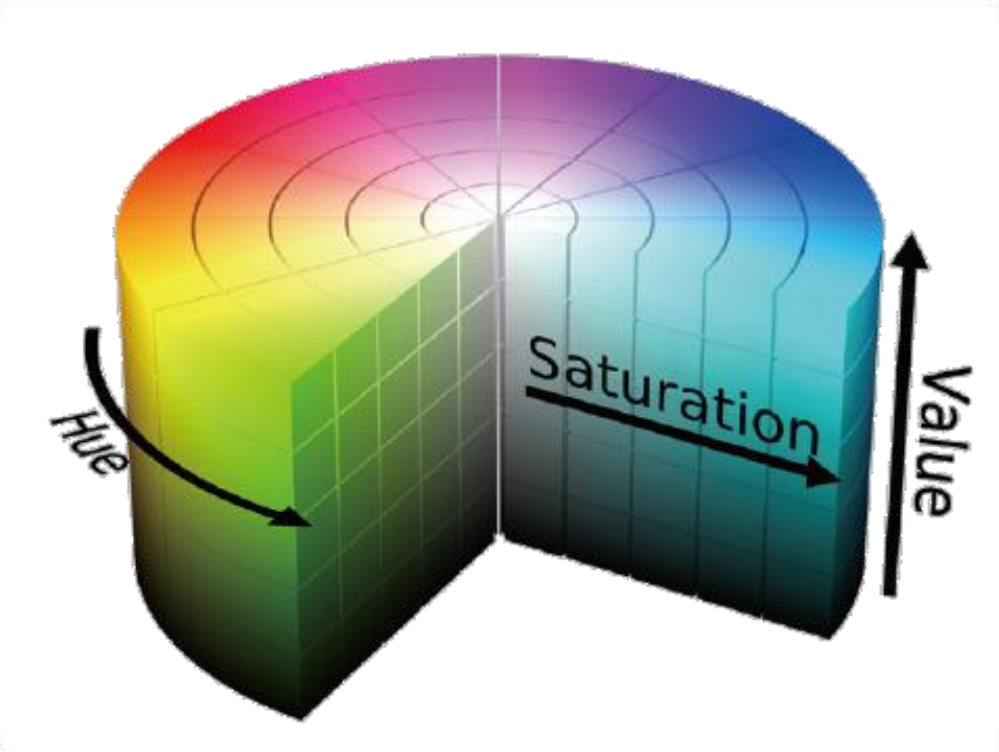
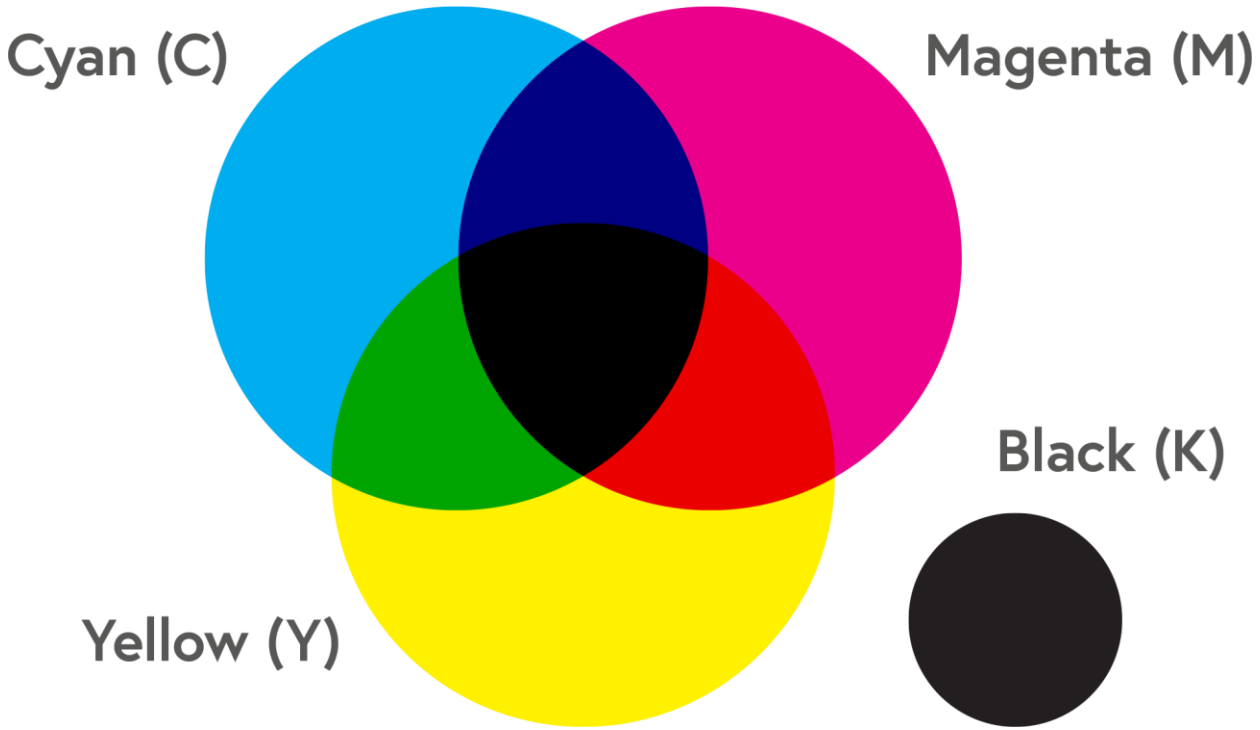


그림 3-7 HSV 컬러 모델

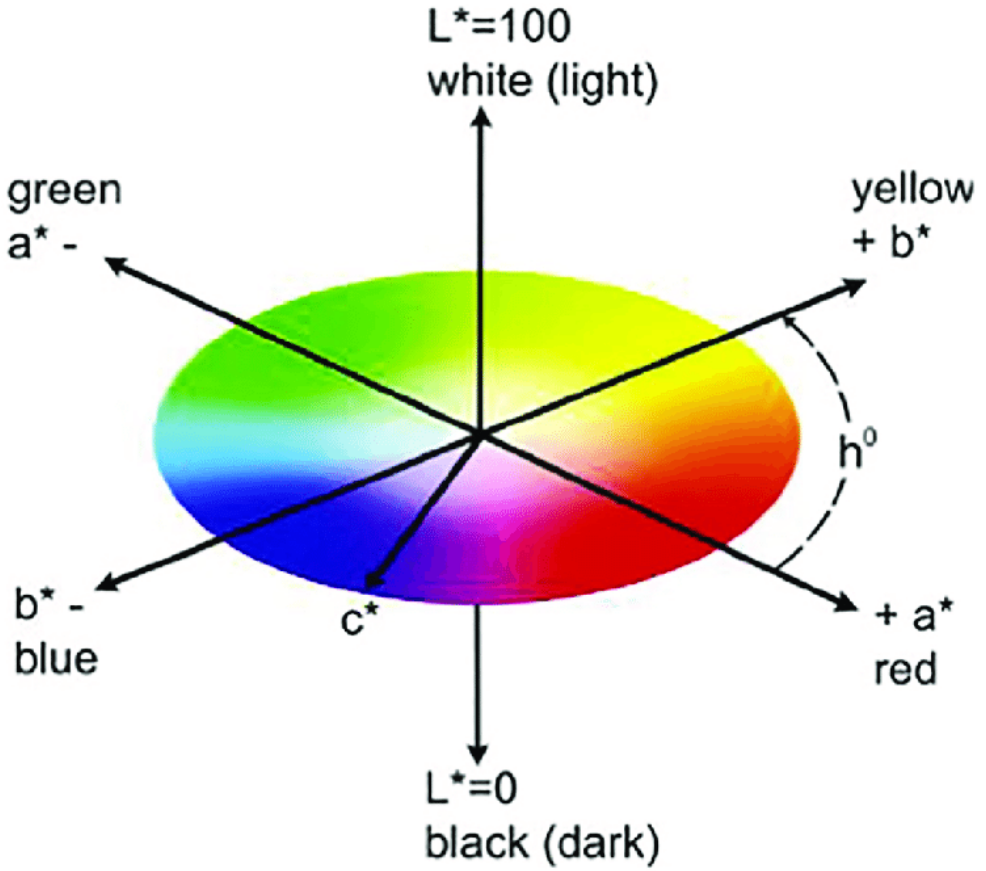
2. 영상 색 공간

- CMYK 컬러 모델
  - 시안(Cyan), 마젠타(Magenta), 노랑(Yellow), 키 블랙(Black)의 네 가지 색상을 기본 색상으로 사용하는 컬러 모델
  - 주로 인쇄 장치 등에 사용됨 (프린터기 잉크 컬러)



2. 영상 색 공간

- LAB 컬러 모델
  - CIE 1931 색 공간의 일부로, 모든 인간이 볼 수 있는 색상을 수학적으로 표현
  - L은 밝기를(Lightness), a는 Red/Green, b는 Blue/Yellow 값을 표현
  - 색상의 절대값을 나타내므로, 다양한 디스플레이 및 출력 장치 간의 색상 일관성을 표현하는데 사용됨



2. 영상 색 공간

- numpy의 슬라이싱 기능을 이용하여 RGB 채널별로 디스플레이

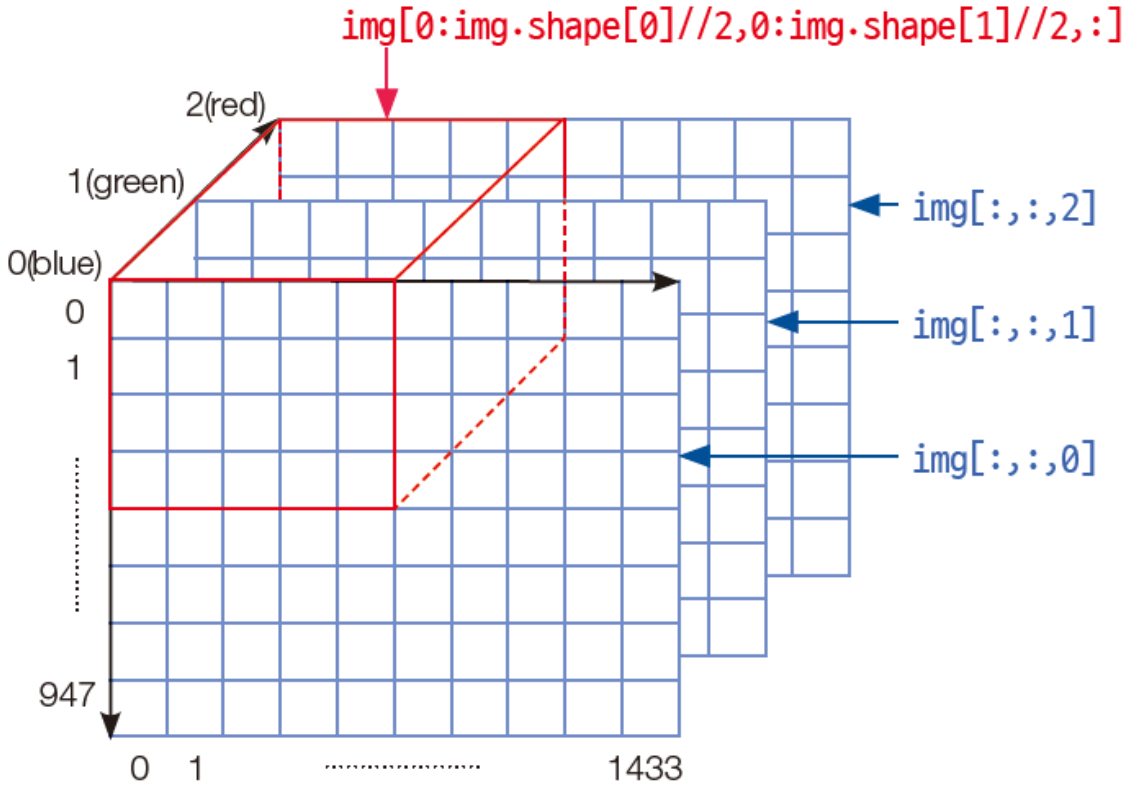


그림 3-8 numpy.ndarray의 슬라이싱을 이용한 영상 일부분 자르기([프로그램 3-1]의 10행)

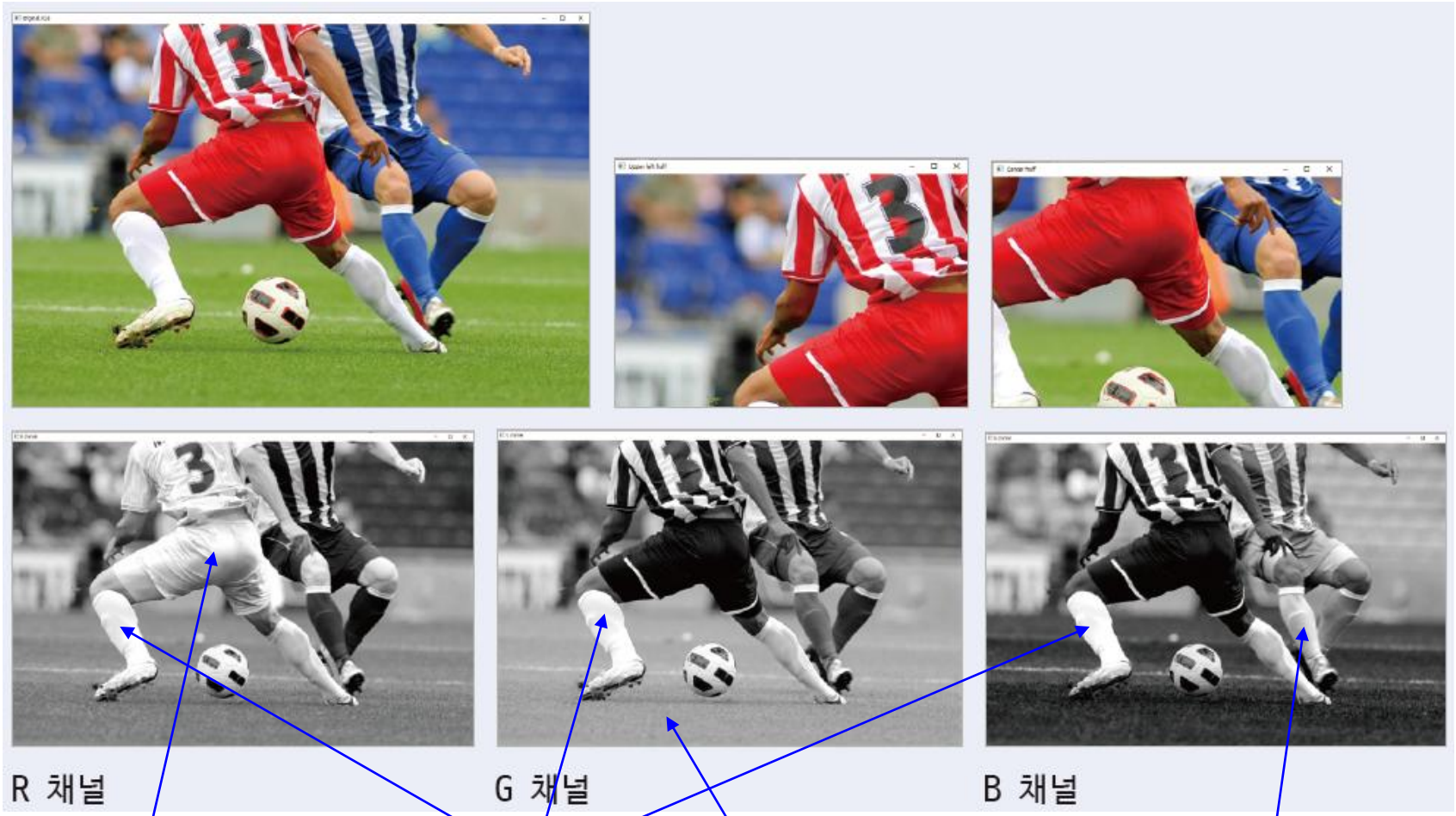
TIP 온라인 부록 A에서 이 책을 공부하는 데 필요한 최소한의 numpy 지식을 제공한다.

2. 영상 색 공간

```python
import cv2

# 이미지 읽기
image = cv2.imread('soccer.jpg')

# R, G, B 채널 분리
B, G, R = cv2.split(image)

# 각각의 채널을 별도로 보여주기
cv2.imshow("Original Image", image)
cv2.imshow("Red Channel", R)
cv2.imshow("Green Channel", G)
cv2.imshow("Blue Channel", B)

# 키 입력 대기
cv2.waitKey(0)

# 모든 창 닫기
cv2.destroyAllWindows()
```

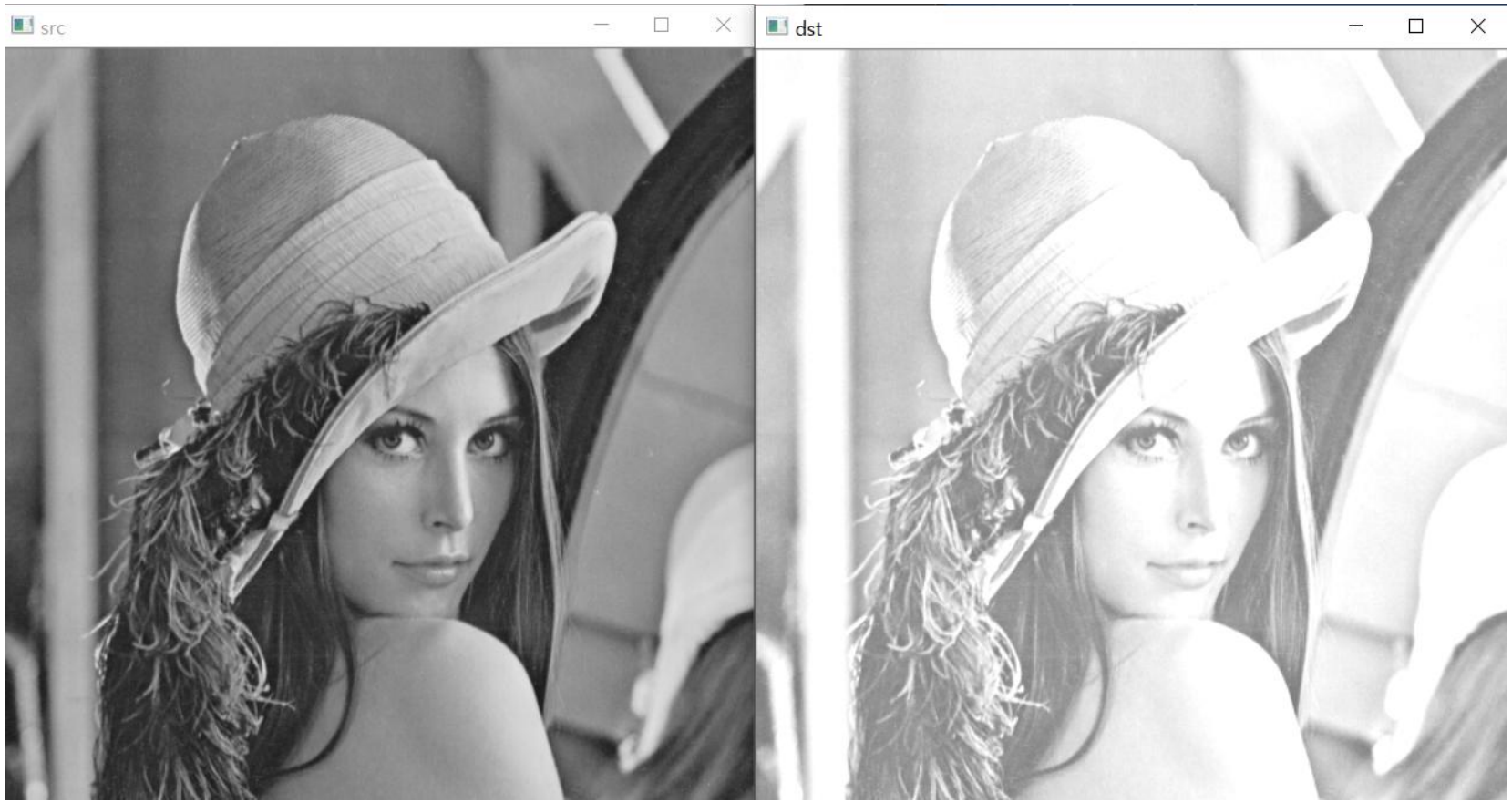
2. 영상 색 공간

- RGB 채널 별 밝게 보여지는 영역이 다르게 나타남



3. 영상 필터링

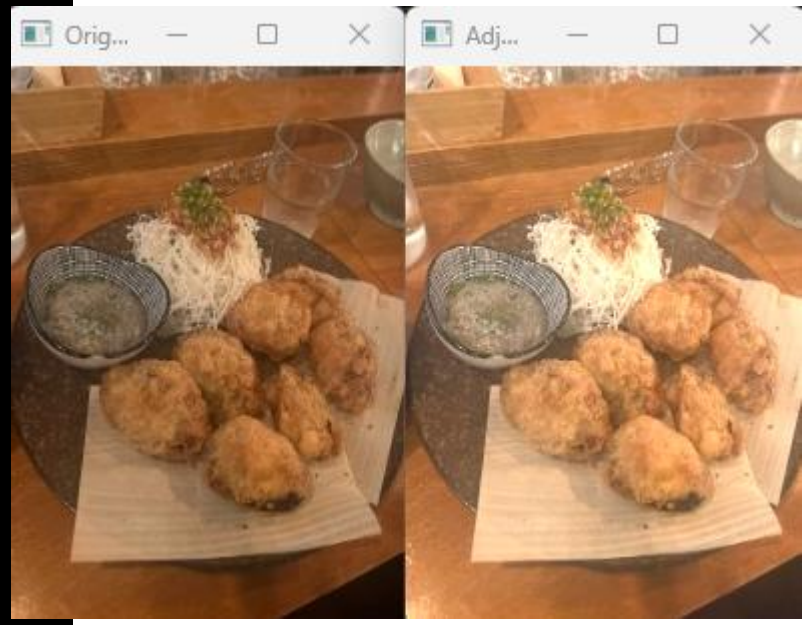
- 단순 덧셈 기반의 밝기 보정
  - cv2.add(이미지, (100))
  - 덧셈 연산 적용 시 픽셀 값이 255 가 넘어간다면? 모두 하얗게 나타나서, 흐려짐



3. 영상 필터링

- OpenCV 함수를 통한 밝기 보정 (convertScaleAbs)

```python
import cv2

image = cv2.imread('food.jpg')

# 대비와 밝기 조정 (대비 1.2배, 밝기 +30)
alpha = 1.2  # 대비 계수 (1보다 크면 대비 증가, 1보다 작으면 대비 감소)
beta = 30    # 밝기 증가 값 (양수면 밝기 증가, 음수면 밝기 감소)

# 대비와 밝기 조정 적용
adjusted_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

cv2.imshow('Original Image', image)
cv2.imshow('Adjusted Image (Contrast & Brightness)', adjusted_image)

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



3. 영상 필터링

- 합성곱(Convolutional) 연산을 통한 영상 필터링
  - 스무딩: 영상을 부드럽게 (Blur)
  - 샤프닝: 영상을 날카롭게
  - 엠보싱: 영상이 울퉁불퉁 하도록

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

(a) 스무딩 필터

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

(b) 샤프닝 필터

-1	0	0
0	0	0
0	0	1

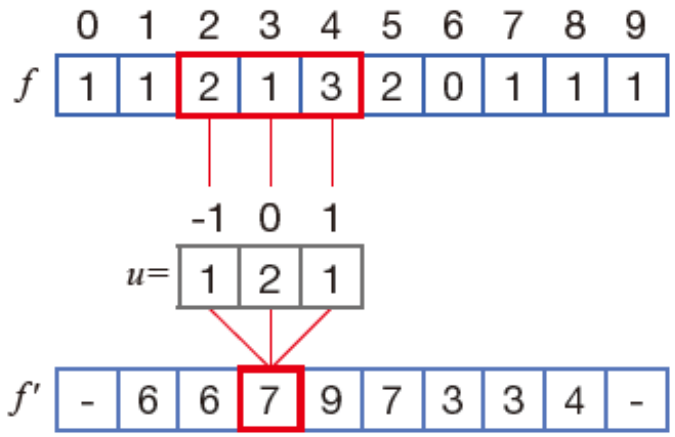
-1	-1	0
-1	0	1
0	1	1

(c) 엠보싱 필터

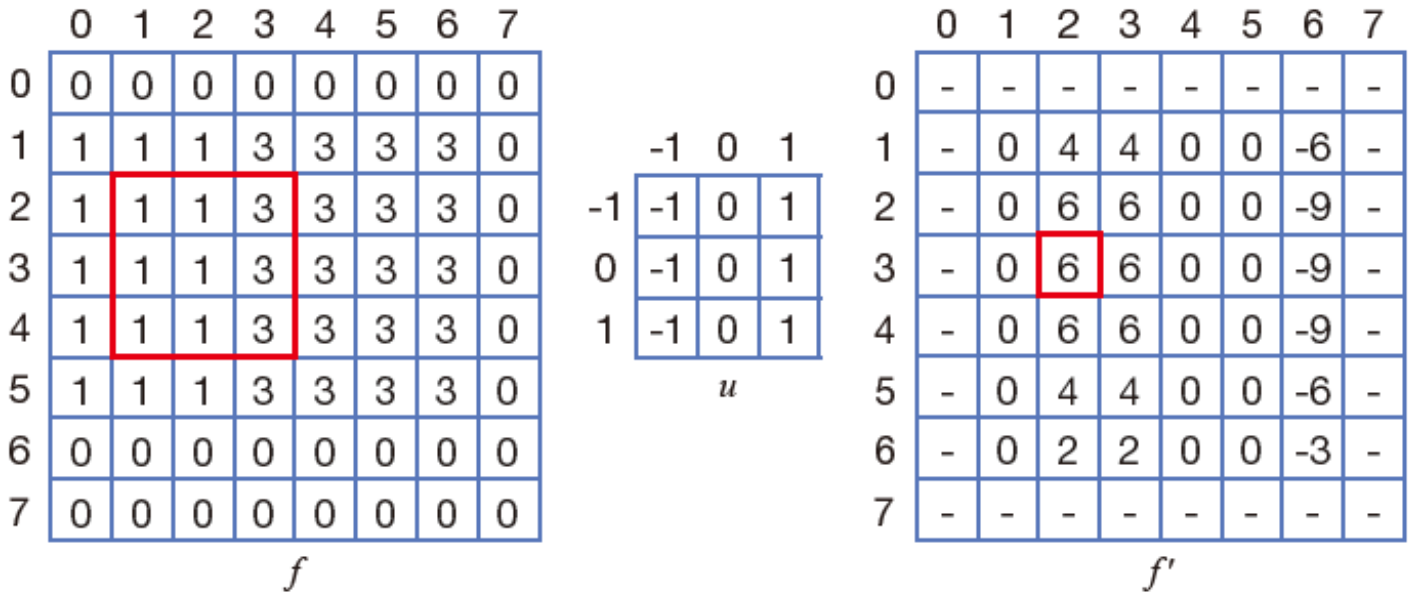
그림 3-17 잡음 제거와 대비 향상을 위한 필터

3. 영상 필터링

- 합성곱(Convolutional) 연산 적용 방법



(a) 1차원 영상에 컨볼루션 적용



(b) 2차원 영상에 컨볼루션 적용

그림 3-16 컨볼루션의 원리

3. 영상 필터링

```python
import cv2
import numpy as np

image = cv2.imread('food.jpg')

# 3x3 샤프닝 커널 정의
# 중앙 값이 양수, 주변 값이 음수로 설정되어 가장자리를 강조함
sharpen_kernel = np.array([[ 0, -1,  0],
                           [-1,  4, -1],
                           [ 0, -1,  0]])

# 샤프닝 필터 적용
sharpened_image1 = cv2.filter2D(image, -1, sharpen_kernel)
sharpen_kernel[1,1] = 5
sharpened_image2 = cv2.filter2D(image, -1, sharpen_kernel)
sharpen_kernel[1,1] = 9
sharpened_image3 = cv2.filter2D(image, -1, sharpen_kernel)

# 결과 시각화
result = cv2.hconcat([image, sharpened_image1, sharpened_image2, sharpened_image3])
cv2.imshow('Sharpening Results', result)
# 키 입력 대기
cv2.waitKey(0)
cv2.destroyAllWindows()
```

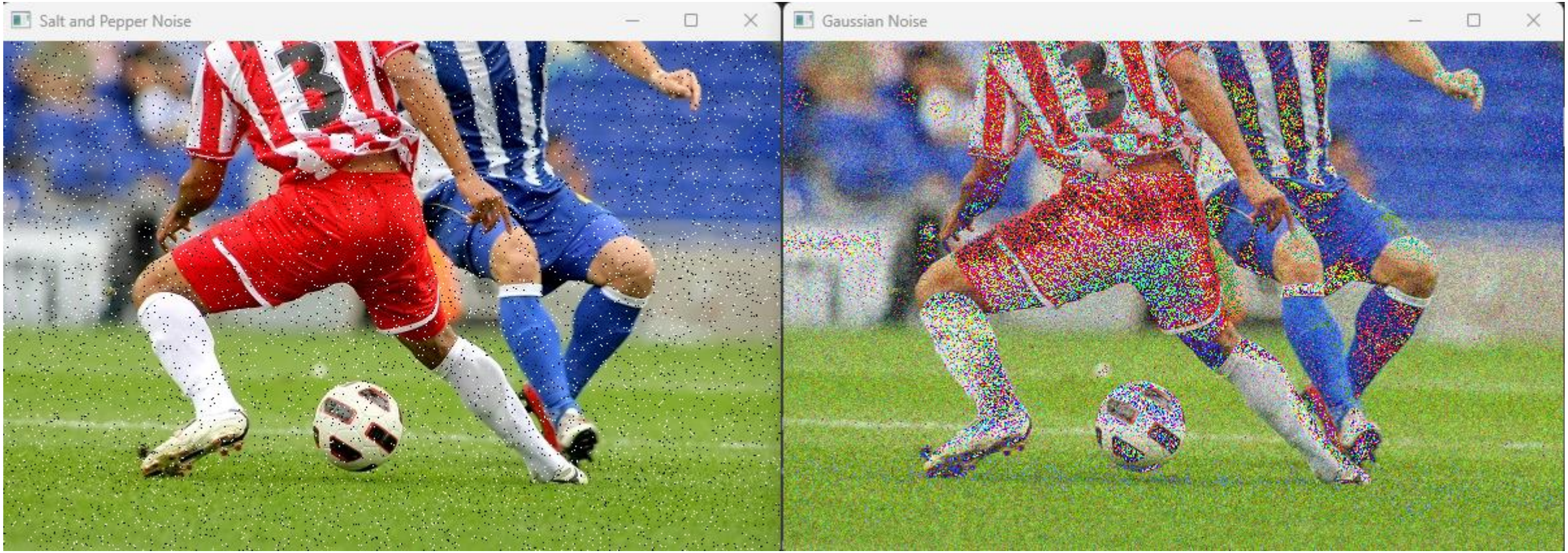
3. 영상 필터링

- 이미지 필터 적용 결과 (원본, 박스필터, 가우시안 블러, 샤프닝, 엠보싱 순)

```
import cv2
import numpy as np

image = cv2.imread('food.jpg')
box_filtered = cv2.blur(image, (9, 9))
gaussian_blurred = cv2.GaussianBlur(image, (9, 9), 10.0)
sharpened_image = cv2.addWeighted(image, 1.5, gaussian_blurred, -0.5, 0)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
emboss_kernel = np.array([[-1, -1, 0], [-1, 0, 1], [0, 1, 1]])
embossed = cv2.filter2D(gray_image, -1, emboss_kernel)
embossed = np.uint8(np.clip(embossed + 128, 0, 255))
result = cv2.hconcat([image, box_filtered, gaussian_blurred, sharpened_image, cv2.cvtColor(embossed,
cv2.COLOR_GRAY2BGR)])
cv2.imshow('Image Processing Results', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. 영상 필터링

- 노이즈 추가 기법
  - 소금후추(Salt and Pepper) 잡음: 검/흰 픽셀을 추가
  - 가우시안 노이즈(Gaussian Noise): 각 픽셀에 가우시안 분포에 따른 랜덤 노이즈를 더하는 방식



3. 영상 필터링

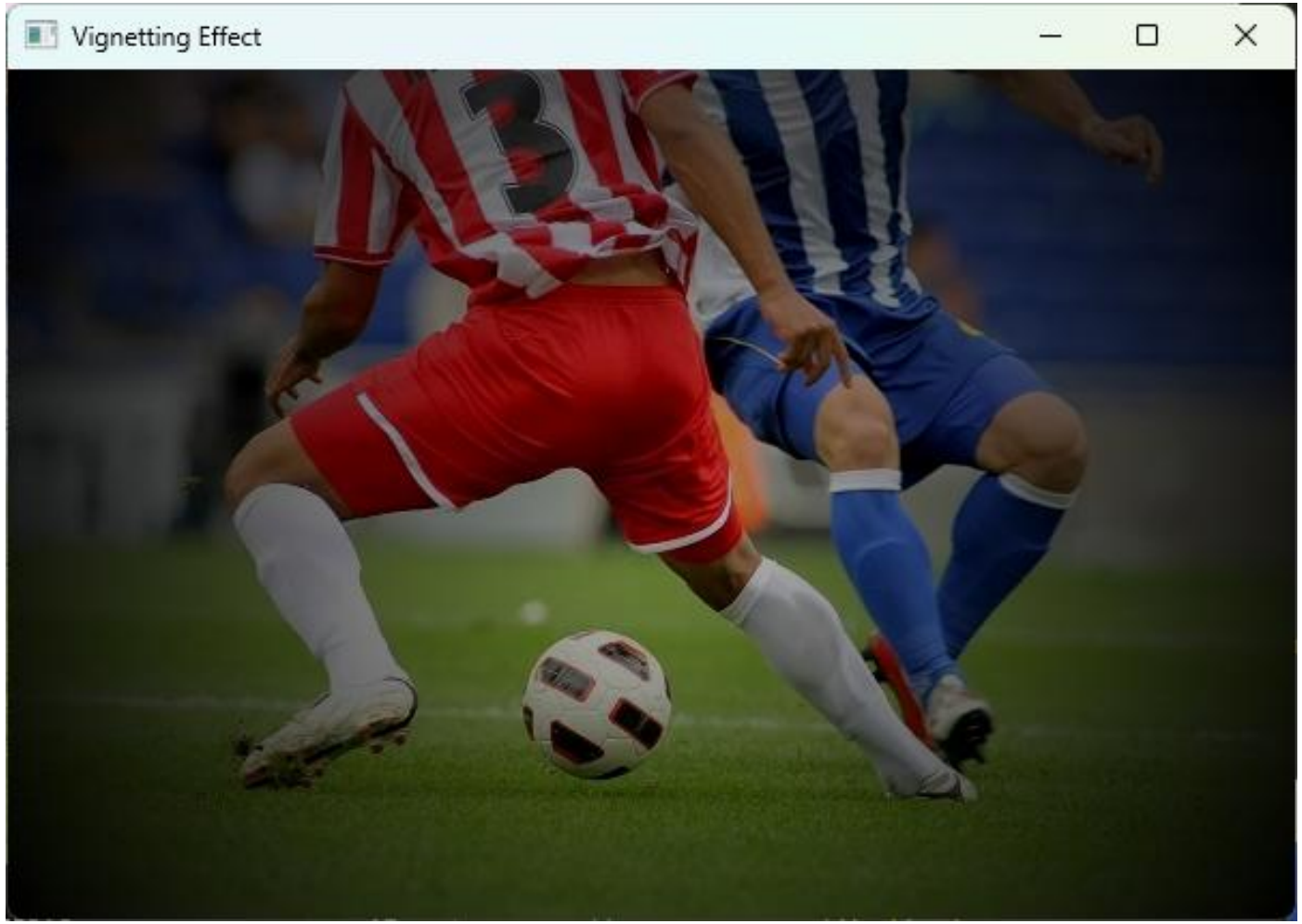
```
import cv2
import numpy as np

image = cv2.imread('soccer.jpg’)
# 1. 후추소금 노이즈 (Salt and Pepper Noise) 적용
def add_salt_and_pepper_noise(img, prob): # 확률에 따라 랜덤하게 픽셀을 흑백(0, 255)으로 설정
    noisy = img.copy()
    black = 0
    white = 255
    probs = np.random.rand(img.shape[0], img.shape[1])
    noisy[probs < prob] = black
    noisy[probs > 1 - prob] = white
    return noisy

salt_pepper_noise_img = add_salt_and_pepper_noise(image, 0.02)  # 2% 확률로 노이즈 추가

cv2.imshow('Original Image', image)
cv2.imshow('Salt and Pepper Noise', salt_pepper_noise_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. 영상 필터링

```
import cv2
import numpy as np
image = cv2.imread('soccer.jpg’)

# 2. 가우시안 노이즈 (Gaussian Noise) 적용
def add_gaussian_noise(img, mean=0, sigma=25):
    row, col, ch = img.shape
    gauss = np.random.normal(mean, sigma, (row, col, ch)).reshape(row, col, ch)
    noisy_image = img + gauss.astype(np.uint8)
    return np.clip(noisy_image, 0, 255).astype(np.uint8)

gaussian_noise_img = add_gaussian_noise(image)


cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Noise', gaussian_noise_img)

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

3. 영상 필터링

- 영상 비네팅 효과 (Vignetting Effect)
  - 이미지 가장자리 부분이 점점 어두워지는 효과로, 이미지 중심으로부터의 거리와 strength 값에 의해 가중치가 적용됨



3. 영상 필터링

```
def add_vignetting_effect(img, strength=200):
    rows, cols = img.shape[:2]
    # 중심점과 거리에 따라 가중치 적용
    X_result, Y_result = np.ogrid[:rows, :cols]
    center_x, center_y = cols / 2, rows / 2
    distance_from_center = np.sqrt((X_result - center_y)**2 + (Y_result - center_x)**2)
    max_distance = np.sqrt(center_x**2 + center_y**2)
    vignetting_mask = 1 - (distance_from_center / max_distance)
    # 가중치 적용 후 비네팅 효과 강화
    vignetting_mask = vignetting_mask * (strength / 255)
    vignetting_mask = np.clip(vignetting_mask, 0, 1)
    # 각 채널에 비네팅 효과 적용
    result = np.zeros_like(img)
    for i in range(3):  # B, G, R 채널에 동일한 마스크 적용
        result[:, :, i] = img[:, :, i] * vignetting_mask
    return result.astype(np.uint8)
vignetted_image = add_vignetting_effect(image)
```

3. 영상 필터링

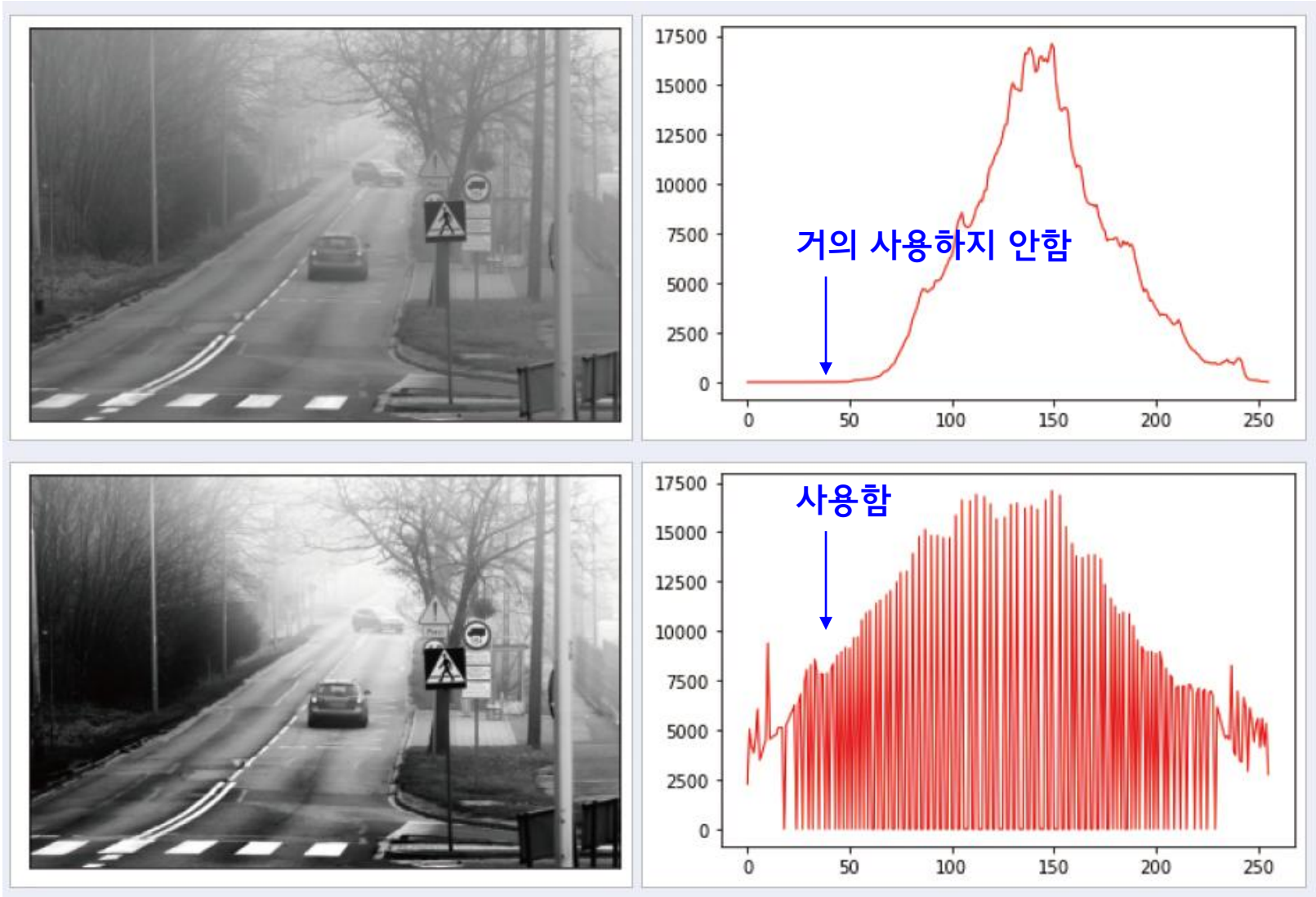
- 감마보정 (Gamma Correction)
  - 픽셀 마다 서로 다른 비율로 자연스럽게 밝기 조절
  - 픽셀에 같은 비율로 밝기 보정 시 (선형적 방법), 전체 밝기가 일괄 변환되어 부자연스러움

```python
import cv2
import numpy as np

g = float(input("감마 값 : "))
img = cv2.imread("soccer.jpg")

# 감마 변환 수행
out = img.copy()
out = out.astype(np.float32)
out = (out / 255) ** g * 255
out = out.astype(np.uint8)

cv2.imshow("original", img)
cv2.imshow("gamma", out)
cv2.waitKey(0)
```

[0-1] 사이의 픽셀값을 가지도록
normalization된 값에 감마연산 식 적용

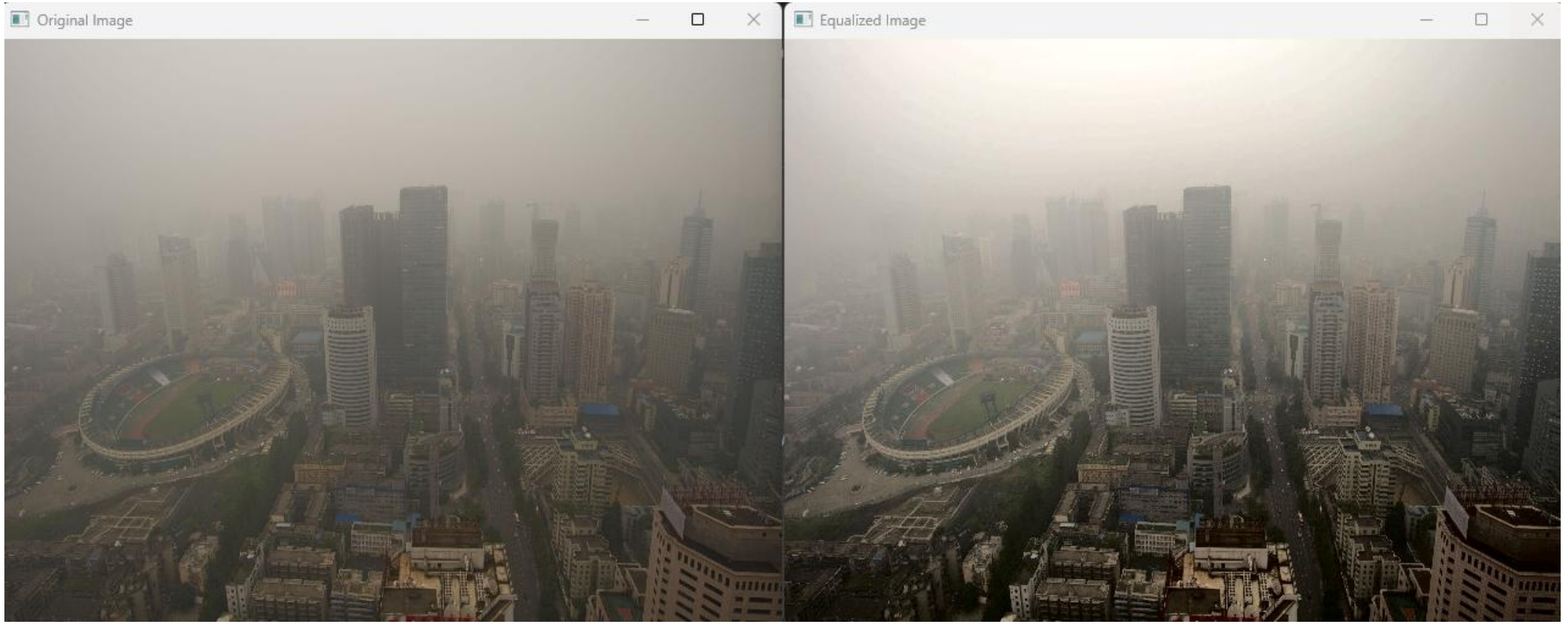
3. 영상 필터링

- 히스토그램 정보를 활용하여, 사용되고 있지 않는 픽셀 정보들을 강화



3. 영상 필터링

- 히스토그램 평활화(Histogram Equalization)
  - 이미지의 히스토그램(밝기 값의 분포)을 평탄화하여 픽셀 값들이 더 균등하게 분포되도록 조정
  - 이미지 컬러 공간을 RGB에서 LAB 등으로 변환하여, 밝기 채널에만 적용 (색상은 유지)

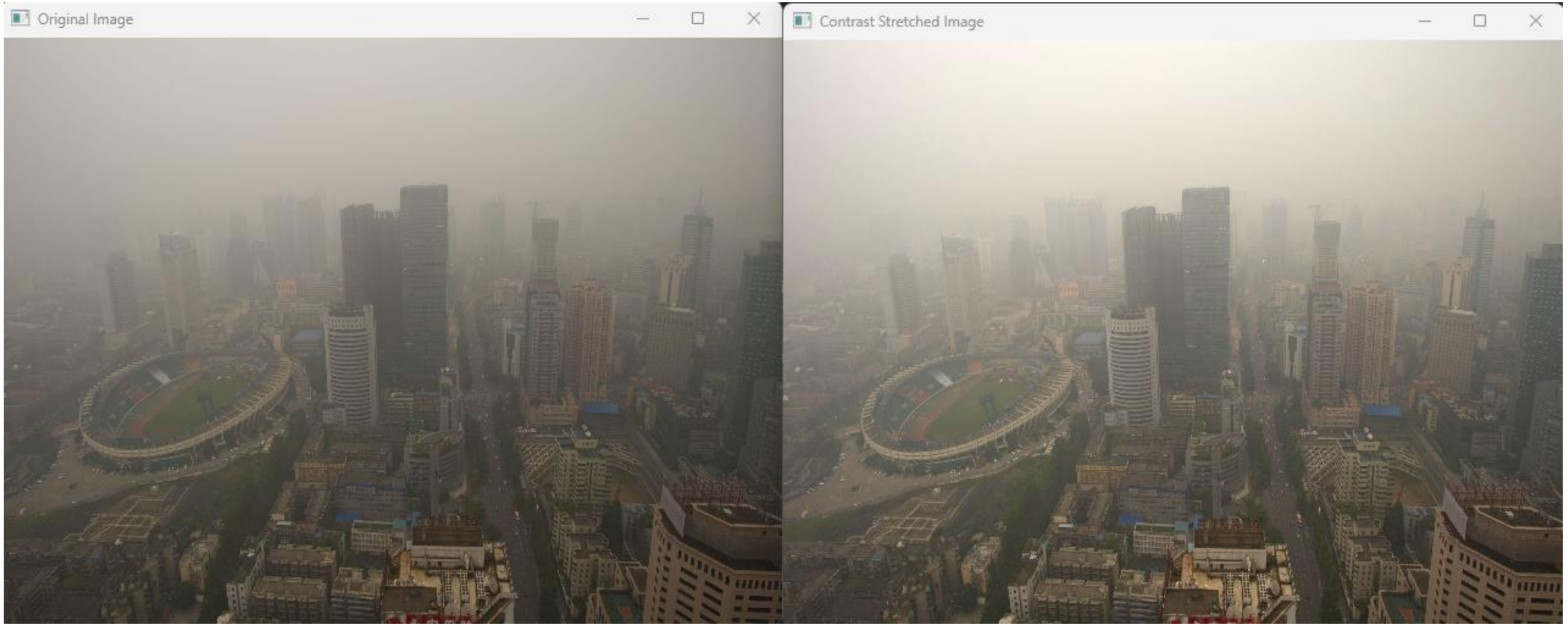


3. 영상 필터링

```python
import cv2

# 컬러 이미지 읽기
image = cv2.imread('hazy.jpg')

# BGR에서 LAB 색 공간으로 변환
lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

# L, A, B 채널 분리
L, A, B = cv2.split(lab_image)

# L 채널에 히스토그램 평활화 적용
L_eq = cv2.equalizeHist(L)

# 평활화된 L 채널과 원본 A, B 채널을 다시 합침
lab_eq_image = cv2.merge([L_eq, A, B])

# LAB에서 다시 BGR로 변환
equalized_image = cv2.cvtColor(lab_eq_image, cv2.COLOR_LAB2BGR)

cv2.imshow('Original Image', image)
cv2.imshow('Equalized Image', equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. 영상 필터링

- 콘트라스트 스트레칭
  - 이미지의 최소값과 최대값을 찾아 이 값들을 이용해 픽셀 값을 선형적으로 변환하는 방법
  - New pixel = (pixel_value - min_val) / (max_val - min_val) * 255

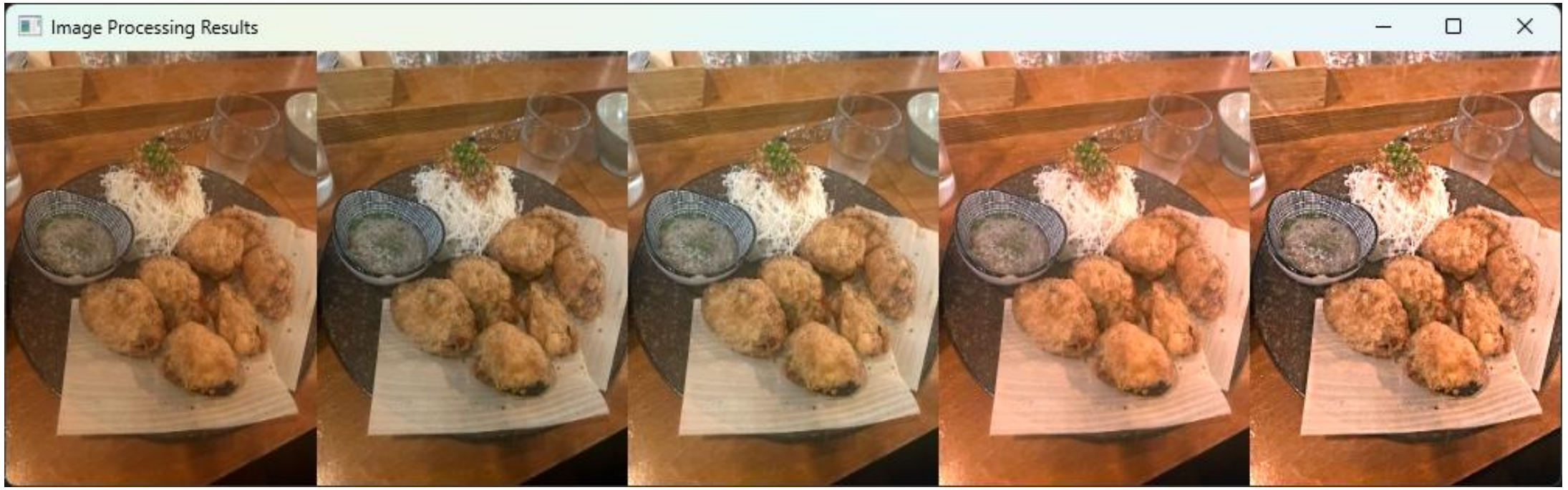


3. 영상 필터링

```
import cv2
import numpy as np

# 이미지 읽기
image = cv2.imread('hazy.jpg')

# 이미지의 최소값과 최대값 찾기
min_val = np.min(image)
max_val = np.max(image)

# 콘트라스트 스트레칭 적용 (새로운 범위: 0~255)
stretched_image = ((image - min_val) / (max_val - min_val) * 255).astype(np.uint8)

cv2.imshow('Original Image', image)
cv2.imshow('Contrast Stretched Image', stretched_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

실습 5. 음식사진 필터 만들기

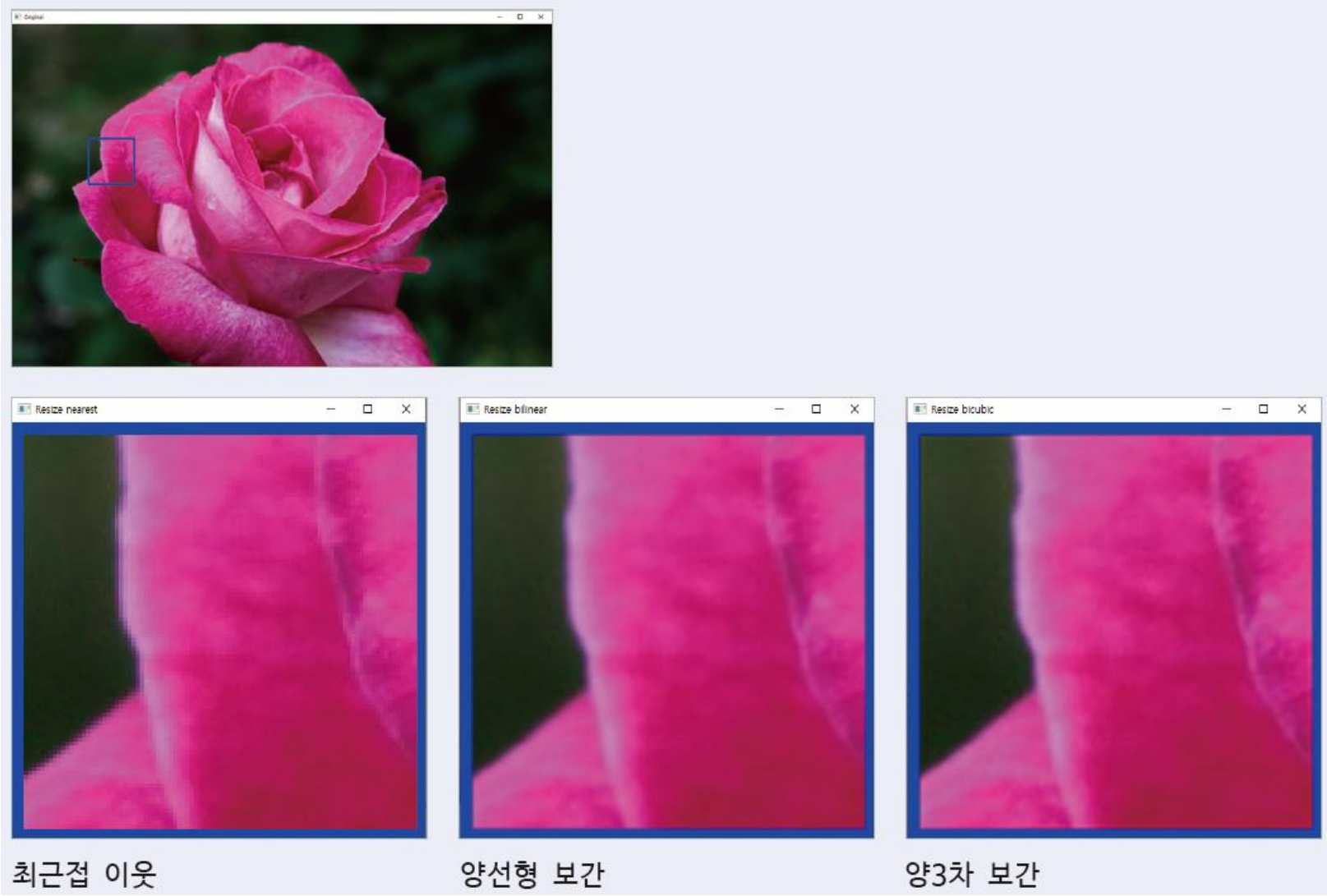
- food.jpg 이미지 또는, 직접 촬영한 음식 사진을 더 맛있어 보이는 사진으로 변환하기
- 실습함에 원본 → 변환과정 → 최종이미지 순서로 이어붙인(cv2.hconcat([img1, img2,]) 이미지 파일 제출하기 (cv2.imwrite로 저장) - **변환중 이미지 수는 자유롭게 조절**



원본 변환중 변환중 변환중 최종

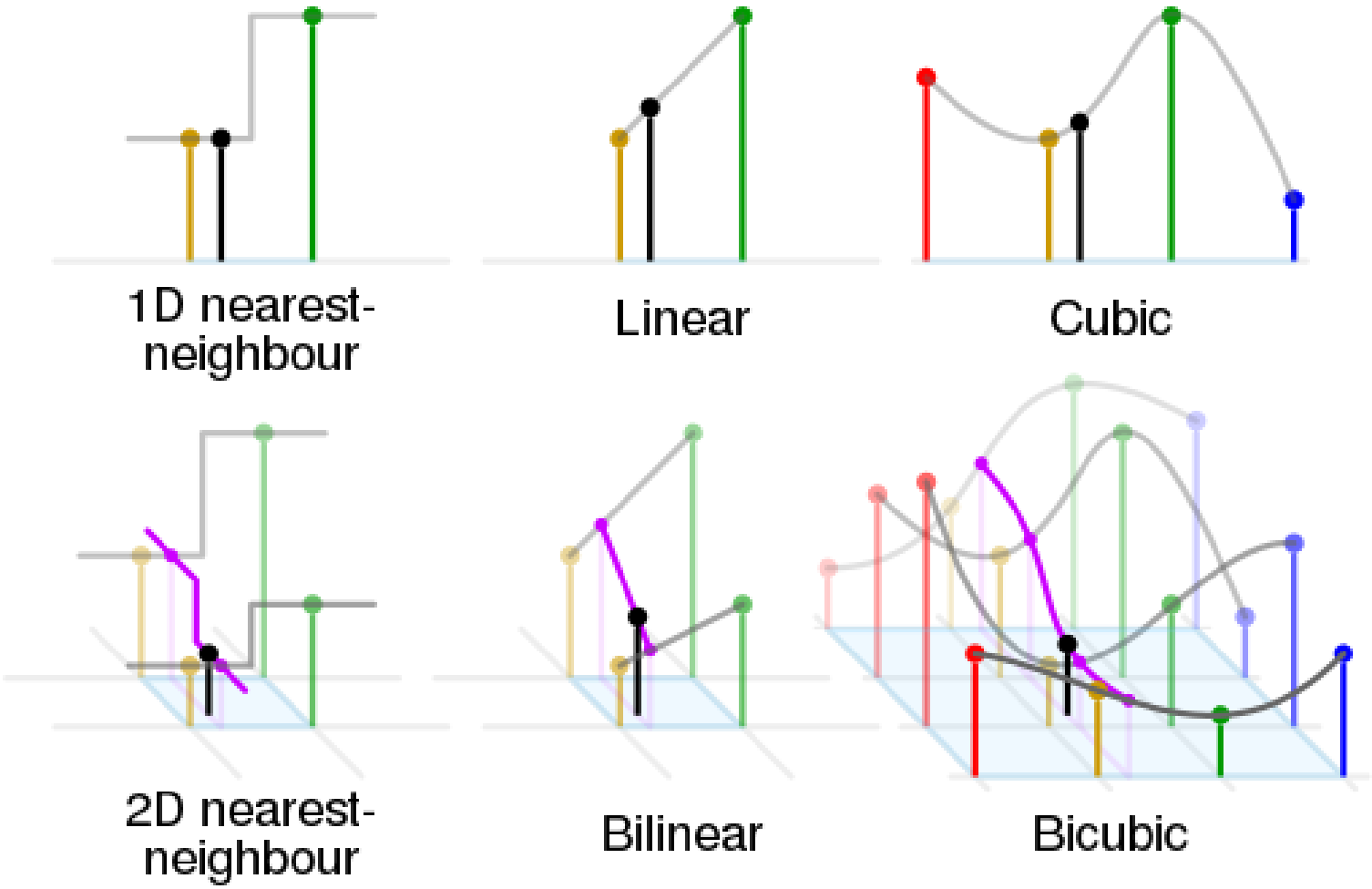
4. 영상 변환

- 크기 조절


Original
Resize nearest
Resize bilinear
Resize bicubic
최근접 이웃
양선형 보간
양3차 보간

4. 영상 변환

- 크기 조절



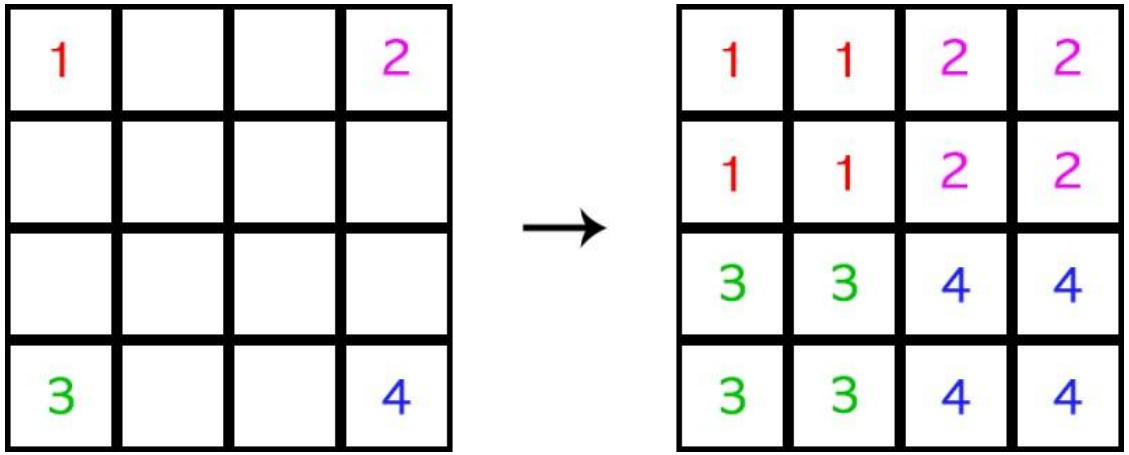
인근 픽셀로 채움

주변 4픽셀 정보를 활용

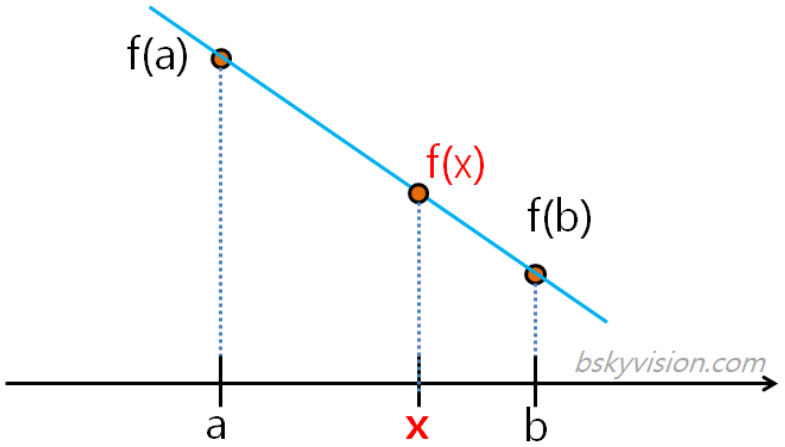
주변 16픽셀 정보를 활용 (3차원 함수로 확장)

4. 영상 변환

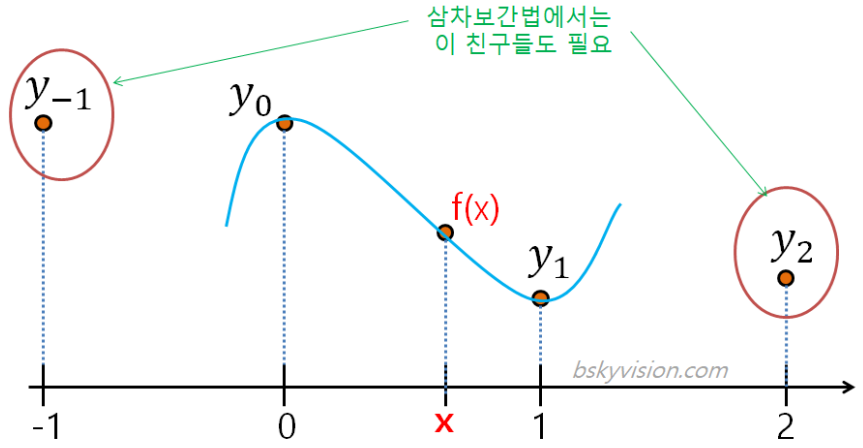
- 크기 조절



NN(Nearest Neighbor): 인근 픽셀로 채움



Bilinear: 주변 4픽셀 정보를 활용



Bicubic: 주변 16픽셀 정보를 활용 (3차원 함수로 확장)

4. 영상 변환

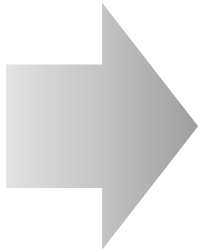
```python
import cv2
import numpy as np

image = cv2.imread('soccer.jpg')
crop_image = image[150:200, 150:200]
resize_dim = (200, 200)

# Nearest Neighbor 방식으로 리사이즈
resize_nn = cv2.resize(crop_image, resize_dim, interpolation=cv2.INTER_NEAREST)
# Bilinear 방식으로 리사이즈
resize_bilinear = cv2.resize(crop_image, resize_dim, interpolation=cv2.INTER_LINEAR)
# Bicubic 방식으로 리사이즈
resize_bicubic = cv2.resize(crop_image, resize_dim, interpolation=cv2.INTER_CUBIC)

result = cv2.hconcat([resize_nn, resize_bilinear, resize_bicubic])
cv2.imshow('Resized Comparison', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

실습 6. 이미지 보간법 적용

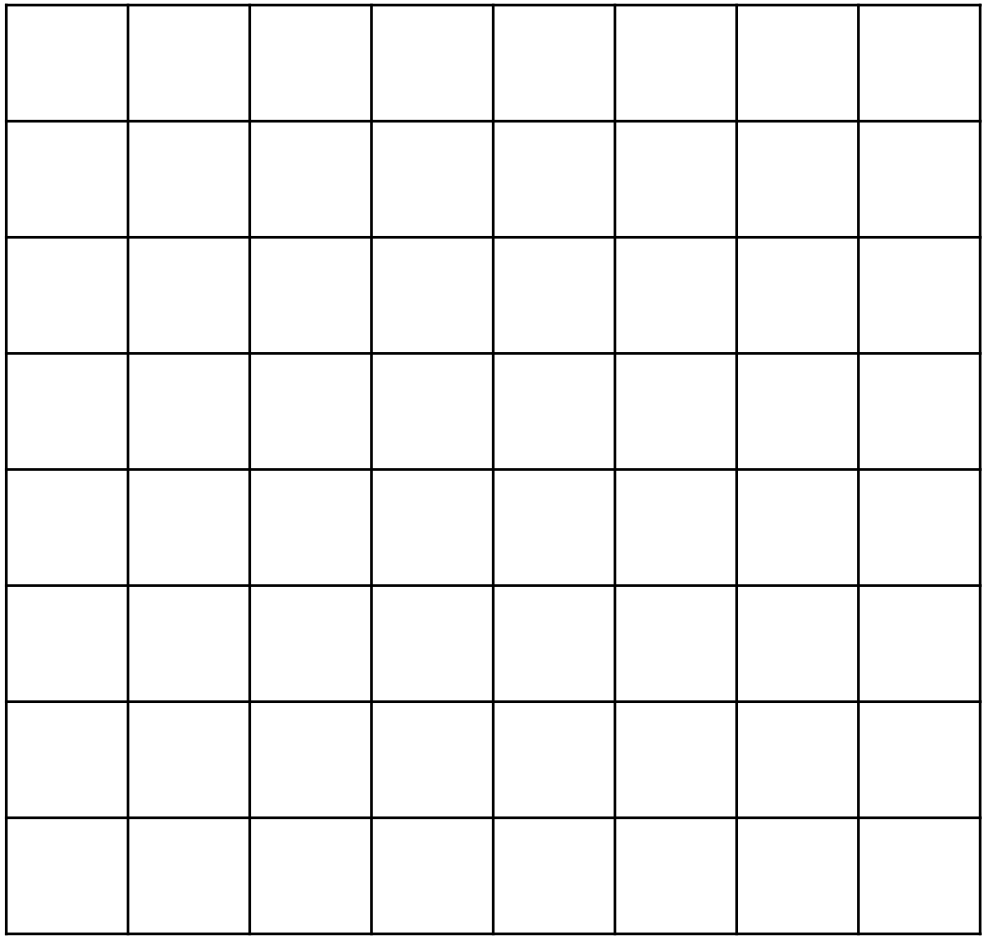
- 4x4의 numpy array를 만들어서, 세 가지 이미지 보간법에 따른 결과를 출력해보기
  - `arr = np.array([[10, 20, 30, 40],[50, 60, 70, 80],[90, 70, 80, 100],[110, 120, 130, 140]], dtype=np.uint8)`
  - 코드 작성 전, 결과를 계산하여 예측해보기
  - 8x8 크기로 변환하기
  - 보간법: NN, Bilinear, Bicubic
  - 실습함에 제출 불필요

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

Numpy array



보간법 적용



4. 영상 변환

- 이미지 기하 변환

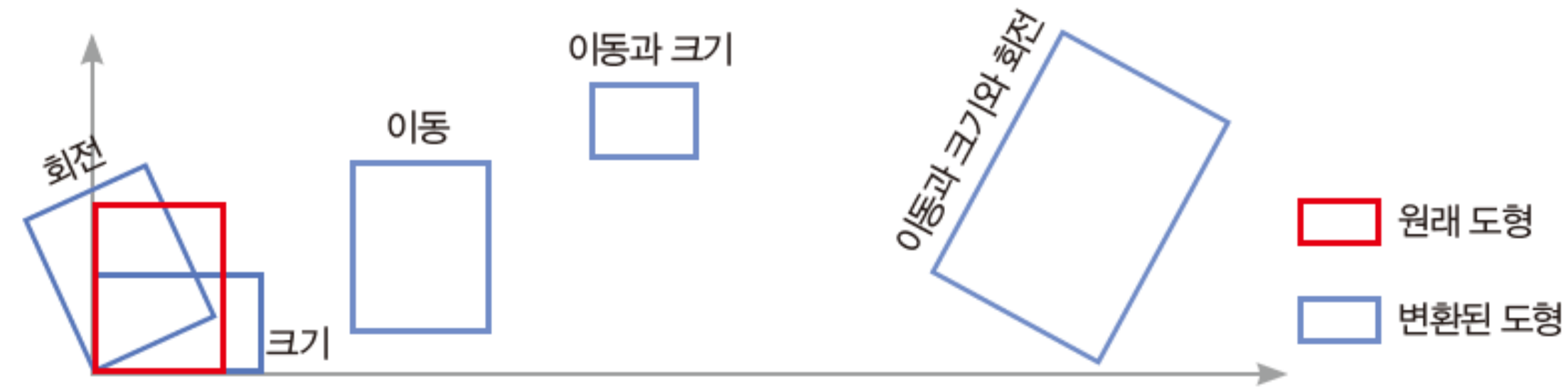


그림 3-19 여러 가지 기하 변환

4. 영상 변환

- 이동, 회전, 크기 등을 동차행렬 연산을 통해 변환

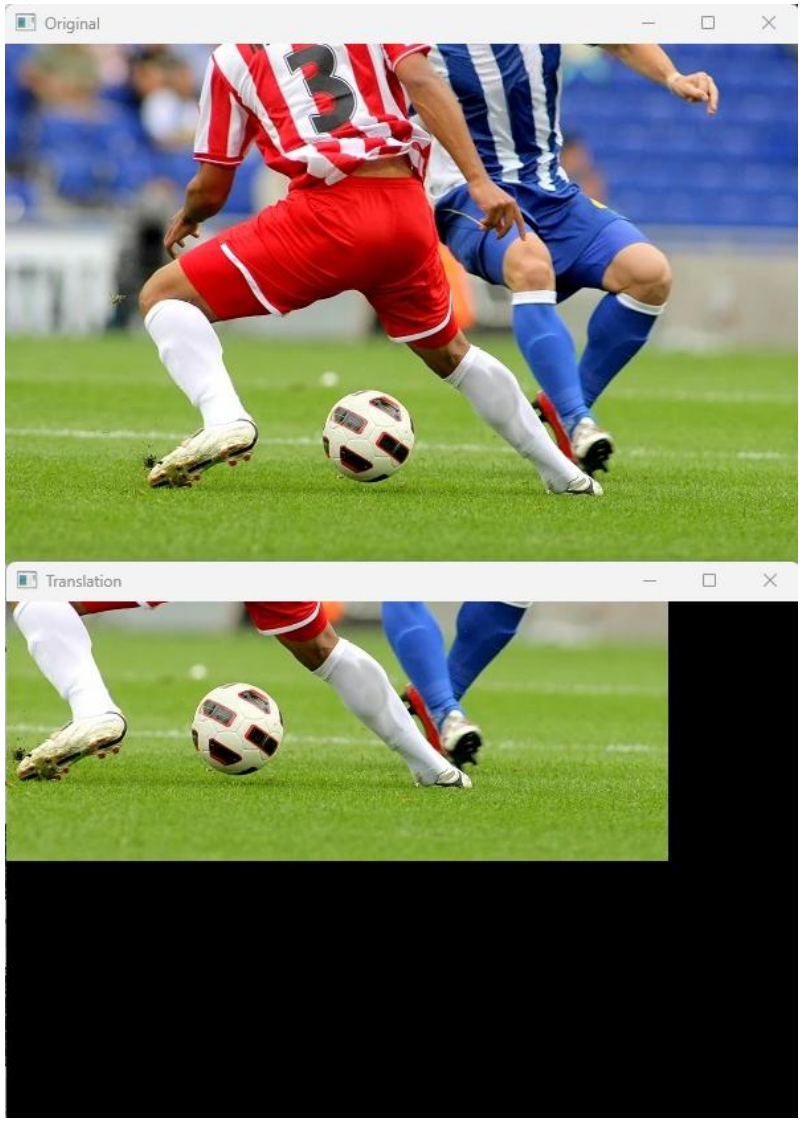
표 3-1 3가지 기하 변환

기하 변환	동차 행렬	설명
이동	$\mathrm{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$	x 방향으로 t_x, y 방향으로 t_y만큼 이동
회전	$\mathrm{R}(\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$	원점을 중심으로 반시계 방향으로 θ만큼 회전
크기	$\mathrm{S}(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	x 방향으로 s_x, y 방향으로 s_x만큼 크기 조정(1보다 크면 확대, 1보다 작으면 축소)

4. 영상 변환

- 이미지 이동
  - X축으로 -100 픽셀, Y축으로 -200 픽셀

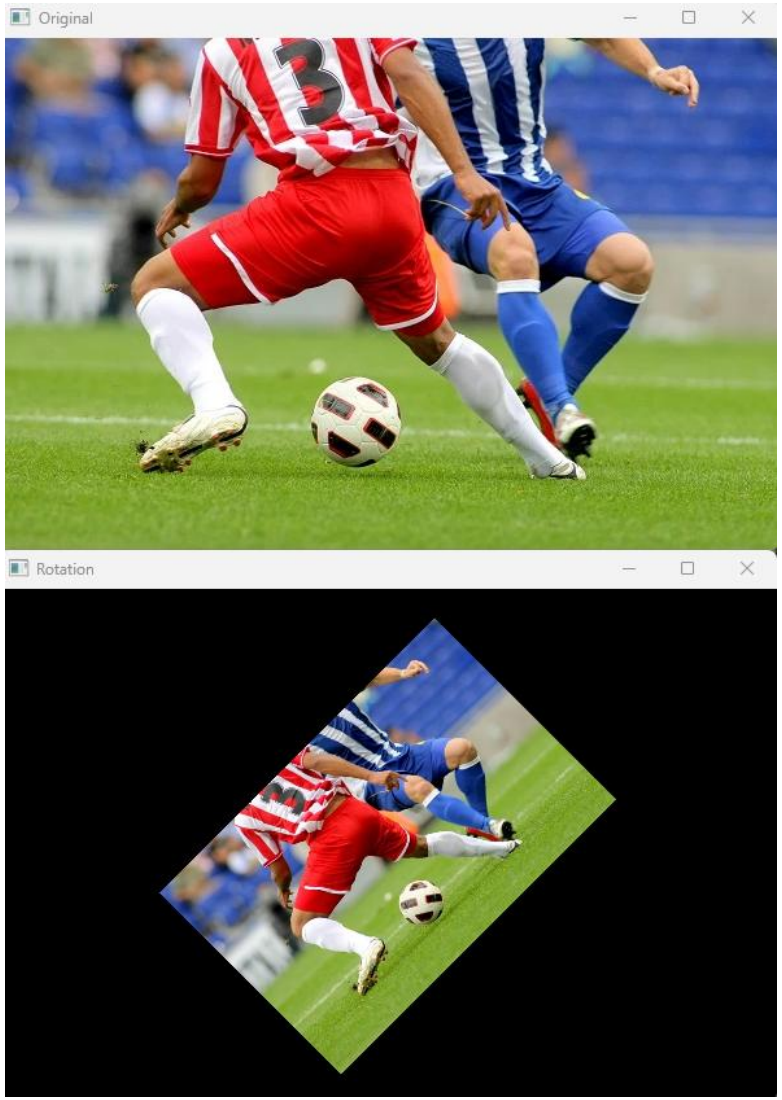
```
import cv2
import numpy as np
img = cv2.imread('soccer.jpg')
h,w,c = img.shape

# 변환 행렬, X축으로 -100, Y축으로 -200 이동
M = np.float32([[1,0,-100],[0,1,-200]])
dst = cv2.warpAffine(img, M,(w, h))
cv2.imshow('Original', img)
cv2.imshow('Translation', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



4. 영상 변환

- 이미지 회전 및 크기변환
  - 45도 회전, 0.5배로 조절

```
import cv2

img = cv2.imread('soccer.jpg')
h,w,c = img.shape

# 이미지의 중심점을 기준으로 45도 회전 하면서 0.5배 Scale
M= cv2.getRotationMatrix2D((w/2, h/2), 45, 0.5)
dst = cv2.warpAffine(img, M, (w, h))
cv2.imshow('Original', img)
cv2.imshow('Rotation', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



5. 영상 품질 측정

- PSNR(최대 신호 대 잡음 비, Peak Signal to Noise Ratio)
  - 신호가 가질 수 있는 최대 전력에 대한 잡음의 전력을 나타내는 값으로 영상 화질 손실을 측정하기 위해 사용되는 품질 측정 지표

$$PSNR = 20 \cdot \log_{10}\left(\frac{PIXEL\_MAX}{\sqrt{MSE}}\right)$$

  - PSNR은 보통 데시벨(dB) 단위로 표현
  - MSE (Mean Squared Error)는 두 이미지 간의 차이를 계산한 평균 제곱 오차를 의미
  - PIXEL_MAX는 이미지의 최대 픽셀 값으로, 8비트 이미지에서는 255

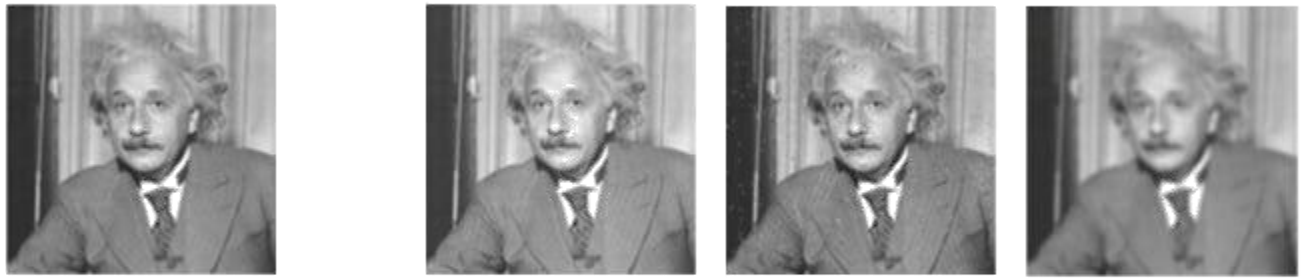
```
def calculate_psnr(img1, img2):
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    psnr = 20 * np.log10(PIXEL_MAX / np.sqrt(mse))
    return psnr
```

5. 영상 품질 측정

- SSIM(구조적 유사도 측정지표, Structural Similarity Index Measure)
  - SSIM은 두 이미지 간의 구조적 유사도를 측정하는 지표로, PSNR보다 인간의 시각 특성을 더 잘 반영하는 평가 방법

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

  - 1에 가까울수록 두 이미지가 구조적으로 유사
  - 0에 가까울수록 이미지 간의 차이가 큼
  - PSNR과 반드시 비례하지는 않음



Original
SSIM=1

PSNR=26.547
SSIM=0.988

PSNR=26.547
SSIM=0.840

PSNR=26.547
SSIM=0.694

과제 1

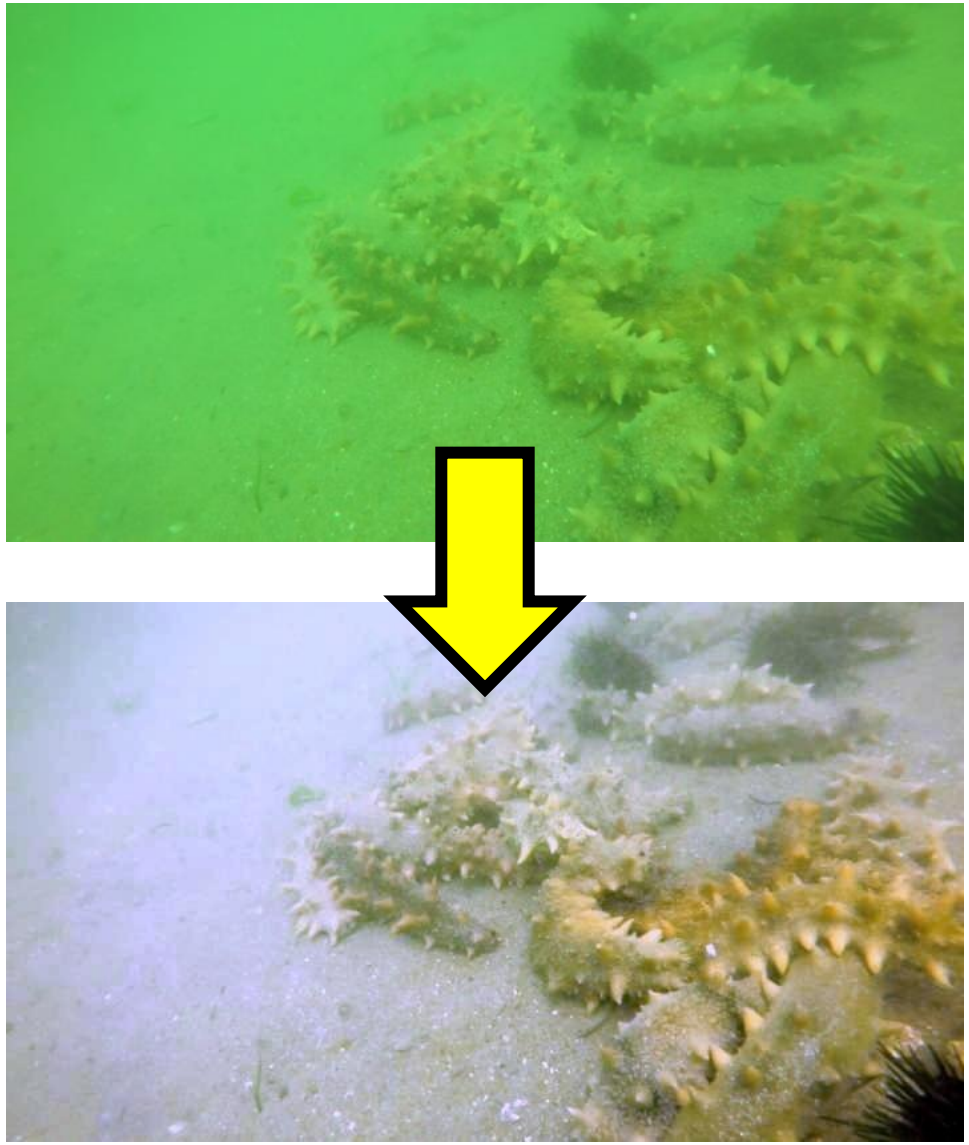
- 물속(Underwater) 이미지를 복원하는 컴퓨터비전 알고리즘 만들기
  - Python코드로 작성할 것 (enhancement.py의 enhancement 함수를 작성)
  - 딥러닝 모델은 사용하지 말 것 (Pytorch, Tensorflow 등 금지)
  - opencv, numpy만 활용하여 개발할 것
  - CPU에서도 작동하는 영상처리 기술을 활용할 것
  - GT 이미지를 활용하는 경우 0점 처리, 이미지 처리 기술만을 활용할 것
  - 모든 이미지(20장)에 동일한 코드를 적용하여, 결과를 예측해야 함

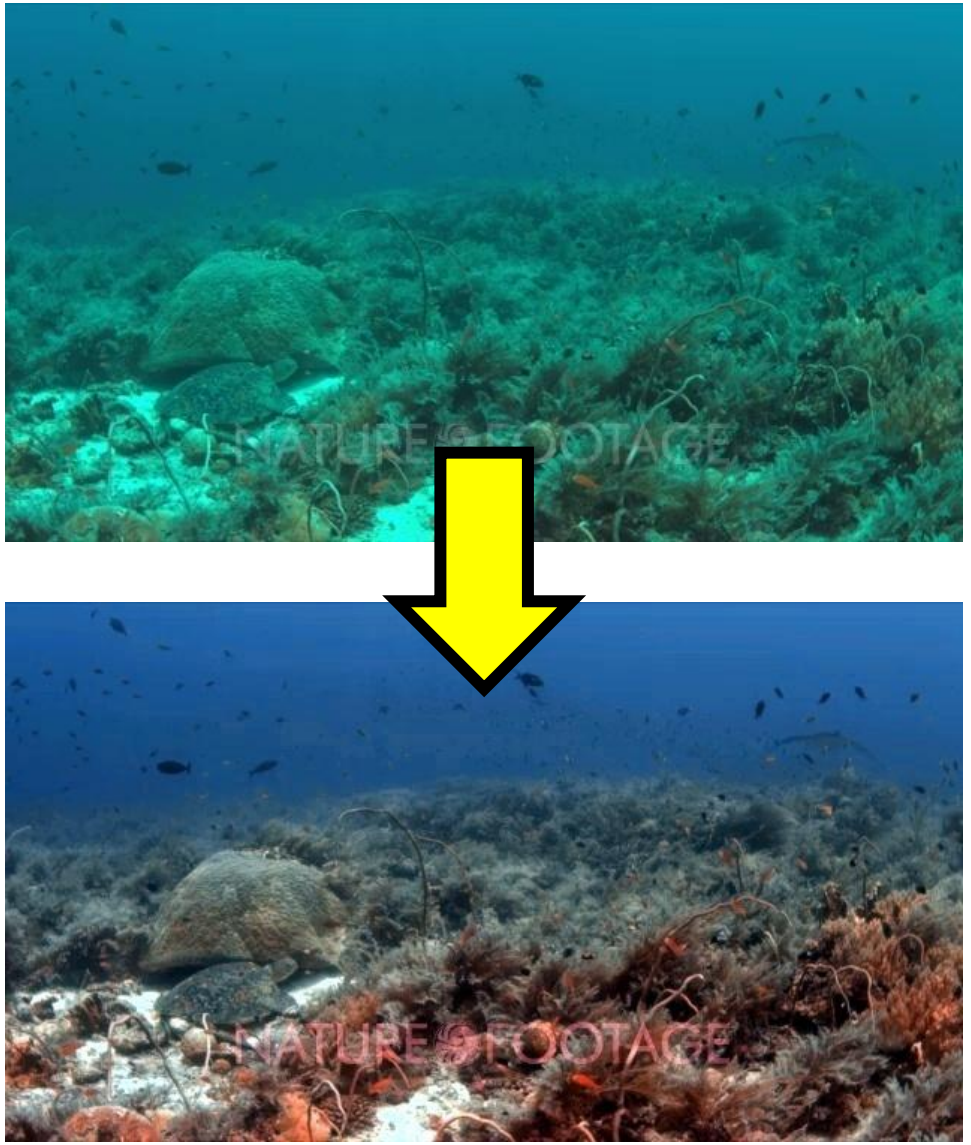
- 제출물: enhancement.py 파일만 제출 (결과물, test코드 제출 불필요)

- 평가기준 (SSIM 점수로 순위 평가, SSIM이 1에 가까울수록 좋음)
  - 1~15등: 5점, 16~25등: 4점, 26등~: 3점
  - 코드 미작동, 미완성, 미제출: 0점
  - Delay는 0점 (반드시 기한 내 제출)

*GT와 Noisy 데이터 간 PSNR/SSIM 측정결과
Average PSNR: 28.4850 dB
Average SSIM: 1.9189
(Lower Bound, 최저점 기준)

과제 1





실습 6. 이미지 효과 적용하기

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