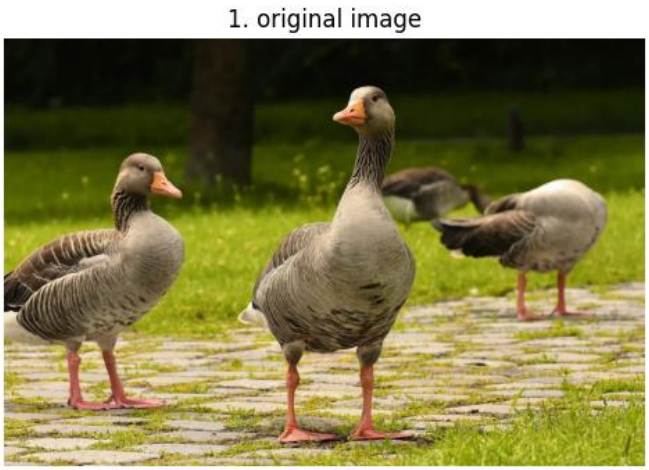
차량비전시스템

Assignment 2

학번 5533082
전공 컴퓨터공학전공
이름 최승환

문제 1) RGB 이미지를 세 개의 채널(R, G, B)로 분리한 뒤 다시 합치는 프로그램을 작성하시오.

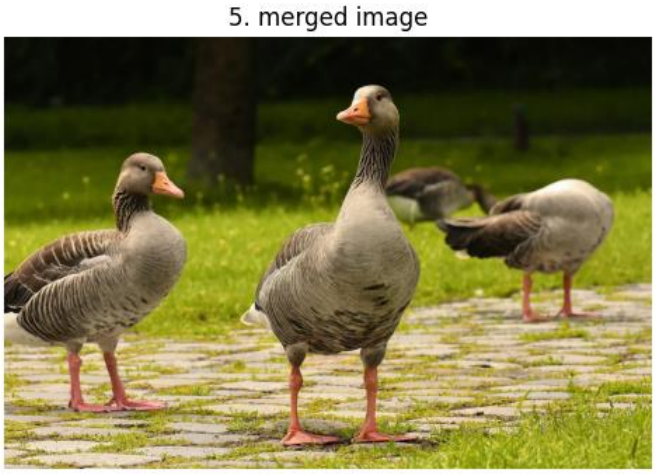


위 원본 이미지를 R/G/B 세 개 채널로 분리하는 방법은 아래 코드와 같다.

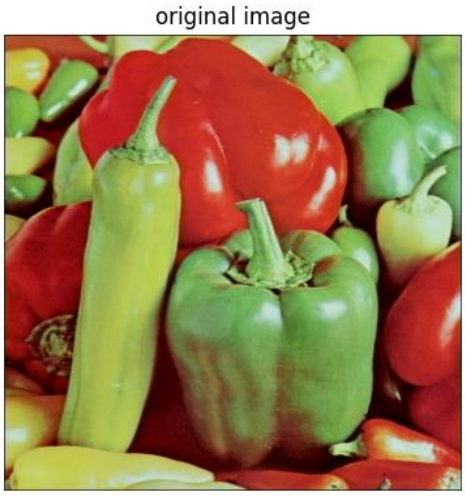
```
origin_img = cv2.imread('./img/duck.jpg')

# RGB 이미지로 변환
RGB_img = cv2.cvtColor(origin_img, cv2.COLOR_BGR2RGB)

# 채널 분리
Red_img, Green_img, Blue_img = cv2.split(RGB_img)

# R-channel
RGB_img[:,:,0] = Red_img
RGB_img[:,:,1] = 0
RGB_img[:,:,2] = 0
plt.subplot(1, 3, 1)
plt.title("2. Red image")
plt.axis("off")
plt.imshow(RGB_img)
```

분리된 이미지들을 다시 합치는 방법은 opencv에서 제공하는 메소드인 cv2.merge 함수를 사용하면 된다.

```
# 채널 병합
merged_img = cv2.merge((Red_img, Green_img, Blue_img))

# cv2.merge
plt.figure( figsize=(8, 4) )
plt.title("5. merged image")
plt.axis("off")
plt.imshow(merged_img)
plt.show()
```



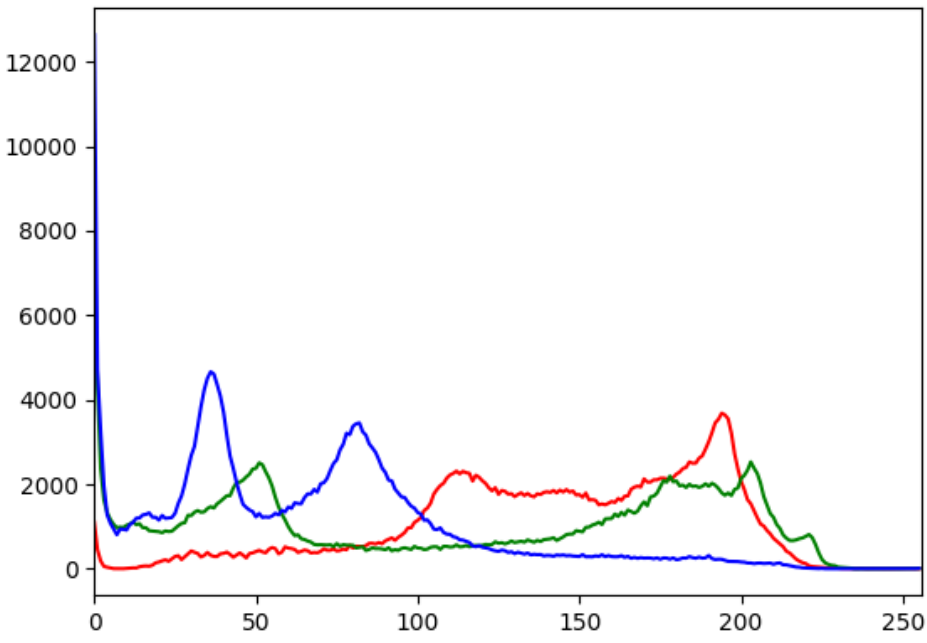
문제 2) peppers.jpg 이미지를 사용하여



2-1) R, G, B 각 채널별로 전역 임계값을 구하시오.

위 원본 이미지의 최적의 전역 임계값을 구하기 위해서 히스토그램을 확인해봐야 한다.

```
# RGB 채널 나누기
Red_img, Green_img, Blue_img = cv2.split(RGB_img)

# 히스토그램 출력, 채널은 0 으로 표시
plt.subplot(222)
hist = cv2.calcHist([Red_img], [0], None, [256], [0, 256])
plt.xlim([0, 256])
plt.plot(hist, color = 'r') # R 색상 히스토그램 표현

hist = cv2.calcHist([Green_img], [0], None, [256], [0, 256])
plt.xlim([0, 256])
plt.plot(hist, color = 'g') # G 색상 히스토그램 표현

hist = cv2.calcHist([Blue_img], [0], None, [256], [0, 256])
plt.xlim([0, 256])
plt.plot(hist, color = 'b') # B 색상 히스토그램 표현
plt.show()
```

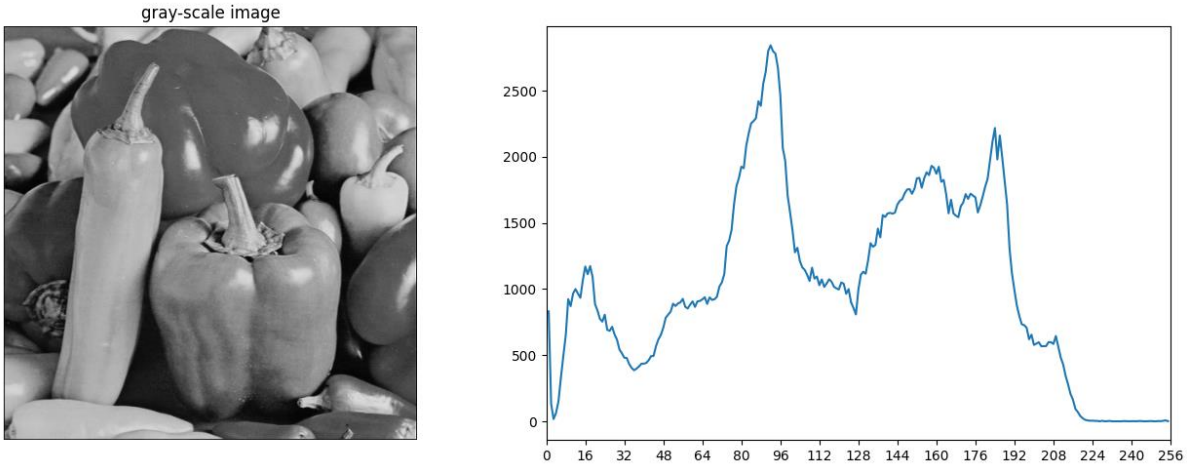


임계값은 전경 영역과 배경 영역의 밝기 차이나 색상 차이를 이용하여 영상의 배경으로부터 전경 영역을 분리할 수 있는 값이기 때문에 위 히스토그램을 보고 최대한의 차이를 나타낼 수 있는 경계를 찾아야 한다.

이 히스토그램의 경우 R-channel : 90, G-channel: 100, B-channel: 52 이 가장 적합하다고 판단된다.

2-2) gray scale로 변환한 후 전역 임계값을 구하시오.

```
# 그레이스케일 이미지로 변환
Gray_img = cv2.cvtColor(RGB_img, cv2.COLOR_RGB2GRAY)

# gray-scale
plt.figure(figsize=(20, 5))
plt.subplot(131)
plt.title("gray-scale image")
plt.axis("off")
plt.imshow(Gray_img, 'gray')

# gray-scale histogram
plt.subplot(132)
hist = cv2.calcHist([Gray_img], [0], None, [256], [0, 256])
plt.xlim([0, 256])
plt.xticks([i*16 for i in range(17)])
plt.plot(hist)
```



위 히스토그램을 보면 임계값을 127로 설정했을 때 이진화 결과가 가장 좋았다.

2-3) 2)에서 얻은 전역 임계값을 기준으로 영상을 이진화 하시오.

```
# 그레이스케일 이미지에 이진화
# ret: 설정 임계값, Binary_img: 결과 이미지
ret, Binary_img = cv2.threshold(Gray_img, 127, 255, cv2.THRESH_BINARY)

plt.subplot(133)
plt.axis("off")
plt.title("threshold = 127")
plt.imshow(Binary_img, 'gray')

plt.tight_layout()
plt.show()
```

문제 3) peppers.jpg 이미지를 사용하여

3-1) 이미지의 블록 개수를 25개로 나눈 후 적응적 임계값을 구하시오.

```
img = cv2.imread('./img/peppers.jpg')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(RGB_img, cv2.COLOR_RGB2GRAY)

# 가로세로 블록의 개수
N=25
```

```
# 블록 당 가로와 세로 크기 계산
dimh = np.int32(gray_img.shape[0] / N)
dimw = np.int32(gray_img.shape[1] / N)

# 연산에서 제외될 영상 가장자리 크기 계산
dh_rest = np.int32(gray_img.shape[0] % N)
dw_rest = np.int32(gray_img.shape[1] % N)

# 임계값이 저장될 기억 장소 생성
mean_img = np.zeros((N, N)) # 블록의 평균값 저장 배열
median_img = np.zeros((N, N)) # 블록의 중앙값 저장 배열
maxmin_mean_img = np.zeros((N, N)) # 블록의 최대 최소 평균값 저장 배열

# 임계값 적용 후 이진 영상을 담을 기억 장소 생성
binary_mean_img = np.zeros((gray_img.shape[0], gray_img.shape[1]))
binary_median_img = np.zeros((gray_img.shape[0], gray_img.shape[1]))
binary_maxmin_mean_img = np.zeros((gray_img.shape[0], gray_img.shape[1]))
```

블록 내의 평균, 중앙값, (최대+최소)/2 값을 구하는 함수를 정의하고,

이 함수들로부터 반환된 임계값들은 이진화 작업에 쓰일 임계값이기 때문에 따로 저장한다.

```
# 각 블록의 평균값 계산을 위한 함수
def mean_function(img, dimh, dimw, h, w):
   slice = img[h:h+dimh, w:w+dimw]
   mean = np.mean(slice)
   return mean

# 각 블록의 중앙값 계산을 위한 함수
def medain_function(img, dimh, dimw, h, w):
   slice = img[h:h+dimh, w:w+dimw]
   median = np.median(slice)
   return median

# 각 블록의 최대최소 평균값 계산을 위한 함수
def maxmin_mean_function(img, dimh, dimw, h, w):
   slice = img[h:h+dimh, w:w+dimw]
   max = int(np.max(slice))
   min = int(np.min(slice))
   return (max+min)/2

# 각 블록의 평균값, 중앙값, 최대최소평균값 계산
for h in range(0, img.shape[0] - dh_rest, dimh):
    for w in range(0, img.shape[1] - dw_rest, dimw):
        if(h + dimh < img.shape[0] and w + dimw < img.shape[1]):
            mean_img[np.int32(h/dimh), np.int32(w/dimw)]
```

```
                                    = mean_function(gray_img, dimh, dimw, h, w)
            median_img[np.int32(h/dimh), np.int32(w/dimw)]
                                    = medain_function(gray_img, dimh, dimw, h, w)
            maxmin_mean_img[np.int32(h/dimh), np.int32(w/dimw)]
                                    = maxmin_mean_function(gray_img, dimh, dimw, h, w)
```

3-2) 적응적 임계값을 구한 결과를 기준으로 각 블록을 이진화 한 결과를 나타내시오.

이진화 작업을 위해 따로 저장한 임계값들을 활용한다.

```
# 평균을 이진화 임계값으로
for h in range(0, gray_img.shape[0] - dh_rest):
    for w in range(0, gray_img.shape[1] - dw_rest):
        if(gray_img[h, w] >= mean_img[np.int32(h/dimh), np.int32(w/dimw)]):
            binary_mean_img[h, w] = 255
        else:
            binary_mean_img[h, w] = 0

# 중앙값을 이진화 임계값으로
for h in range(0, gray_img.shape[0] - dh_rest):
    for w in range(0, gray_img.shape[1] - dw_rest):
        if(gray_img[h, w] >= median_img[np.int32(h/dimh), np.int32(w/dimw)]):
            binary_median_img[h, w] = 255
        else:
            binary_median_img[h, w] = 0

# 최대최소 평균을 이진화 임계값으로
for h in range(0, gray_img.shape[0] - dh_rest):
    for w in range(0, gray_img.shape[1] - dw_rest):
        if(gray_img[h, w] >= maxmin_mean_img[np.int32(h/dimh), np.int32(w/dimw)]):
            binary_maxmin_mean_img[h, w] = 255
        else:
            binary_maxmin_mean_img[h, w] = 0
```

gray-scale image

threshold type = mean

threshold type = median

threshold type = (max+min)/2

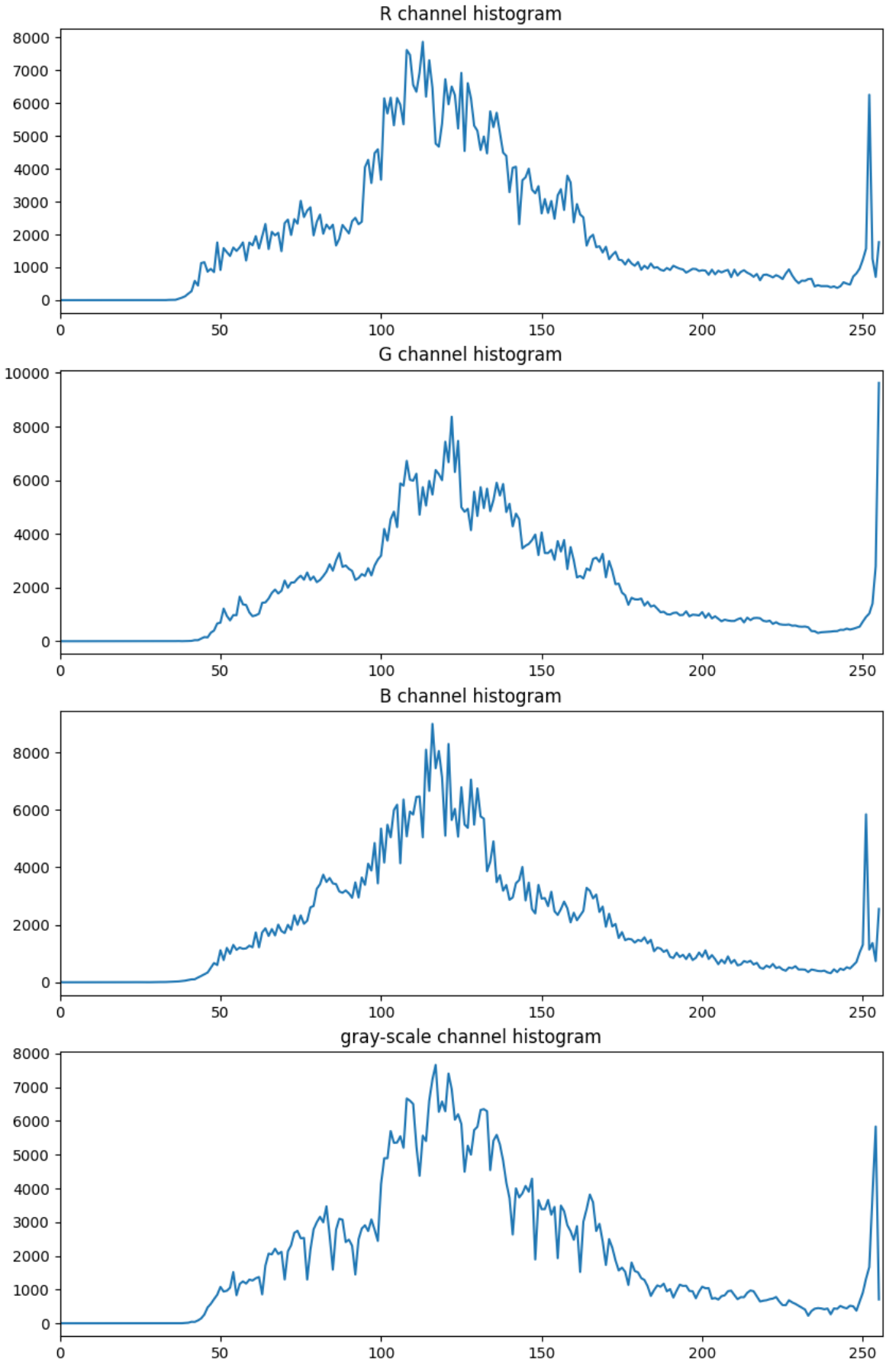
문제 4) MOT16-1.png 이미지 활용

original image

4-1) R/G/B 각 채널별 히스토그램 분포를 나타내시오.

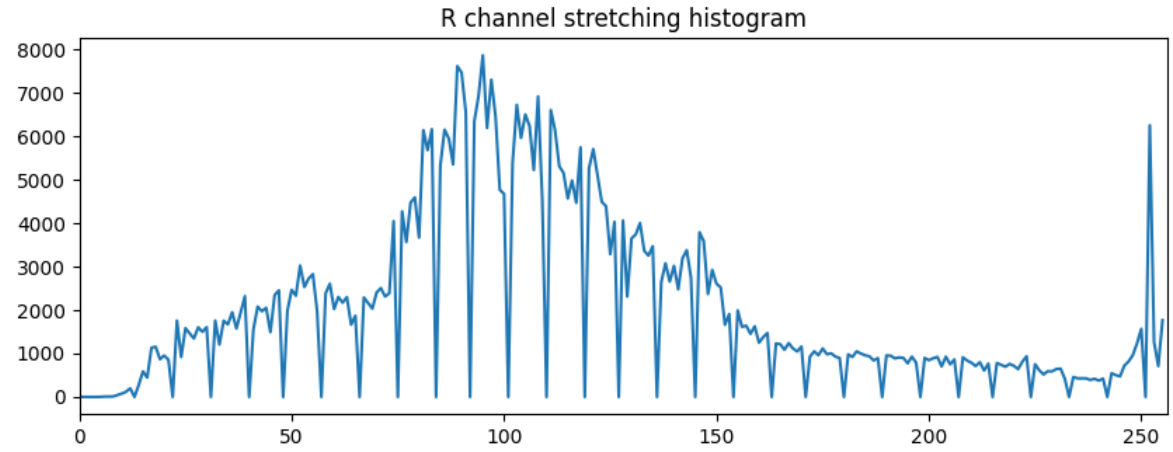
4-2) gray scale로 변환한 후 히스토그램 분포를 나타내시오

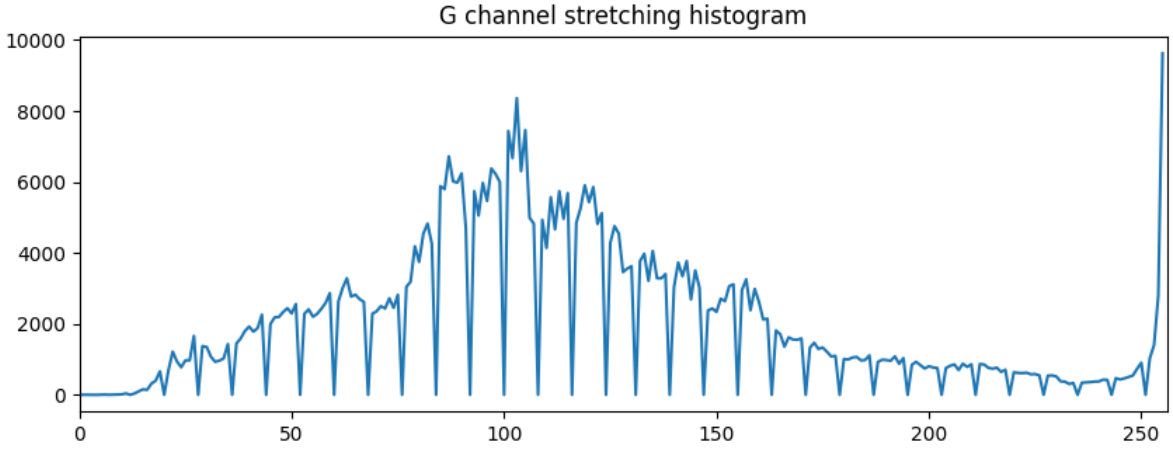
```
img = cv2.imread('./img/MOT16-1.png')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# RGB 채널 나누기
Red_img, Green_img, Blue_img = cv2.split(RGB_img)
# 그레이스케일
gray_img = cv2.cvtColor(RGB_img, cv2.COLOR_RGB2GRAY)

plt.figure(figsize=(10,16))

output_img = [Red_img, Green_img, Blue_img, gray_img]
title = ["R", "G", "B", "gray-scale"]

for idx, img in enumerate(output_img):
    hist = cv2.calcHist([img],[0],None,[256],[0,256])
    plt.subplot(4,1,idx+1)
    plt.title(title[idx] + " channel histogram")
    plt.plot(hist)
    plt.xlim([0,256])
plt.show()
```

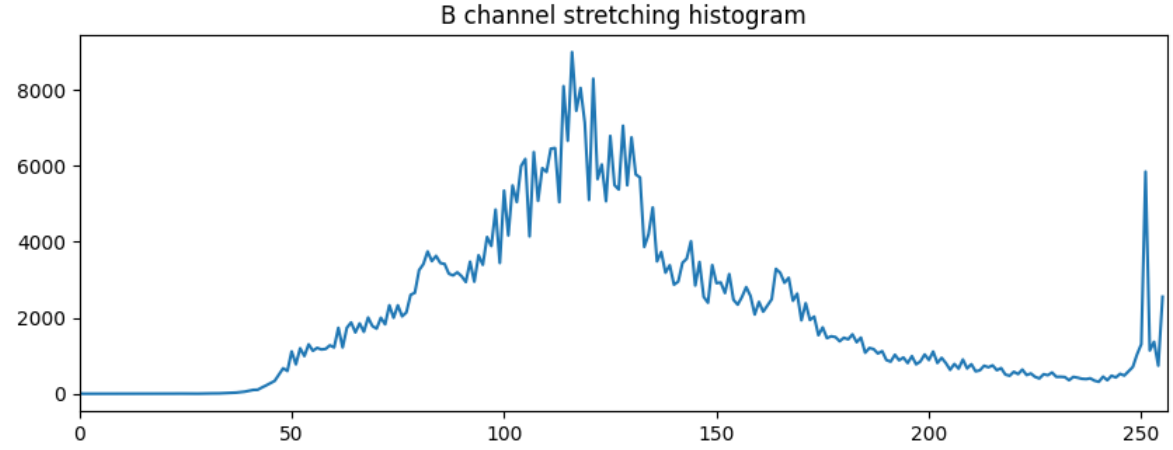

R channel histogram
G channel histogram
B channel histogram
gray-scale channel histogram

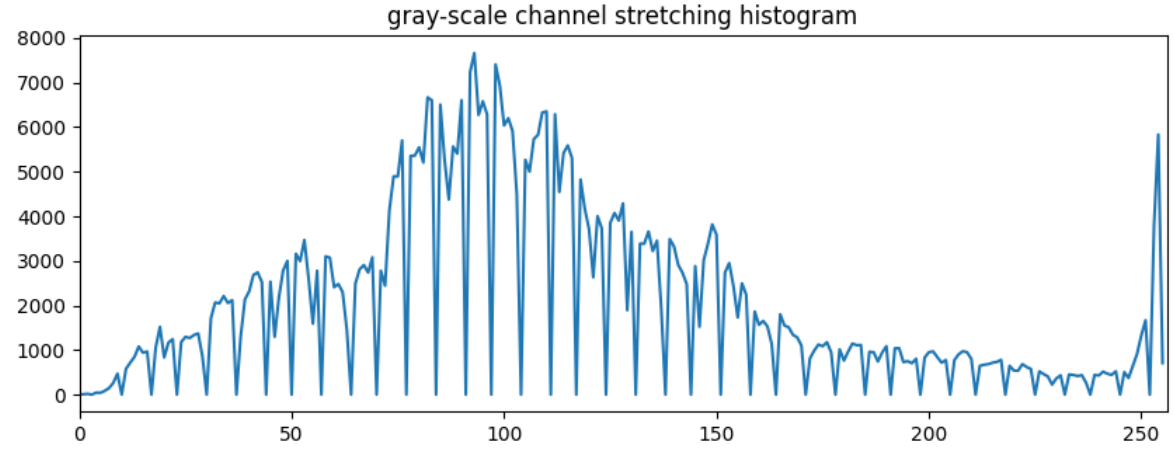
4-3) 히스토그램 스트레칭을 적용한 결과의 이미지와 스트레칭 적용 후의 히스토그램 분포 그래프를 나타내시오.

```
#히스토그램 스트레칭
Red_img_stretch = cv2.normalize(Red_img, None, 0, 255, cv2.NORM_MINMAX)
Green_img_stretch = cv2.normalize(Green_img, None, 0, 255, cv2.NORM_MINMAX)
Blue_img_stretch = cv2.normalize(Blue_img, None, 0, 255, cv2.NORM_MINMAX)
Gray_img_stretch = cv2.normalize(gray_img, None, 0, 255, cv2.NORM_MINMAX)
output_img = [Red_img_stretch, Green_img_stretch, Blue_img_stretch, Gray_img_stretch]

plt.figure(figsize=(16,8))
plt.subplot(121), plt.axis("off"), plt.title("original gray-scale image")
plt.imshow(gray_img, 'gray')
plt.subplot(122), plt.axis("off"), plt.title("stretched gray-scale image")
plt.imshow(Gray_img_stretch, 'gray')
plt.show()

plt.figure(figsize=(10, 16))
for idx, img in enumerate(output_img):
    hist = cv2.calcHist([img],[0],None,[256],[0,256])
    plt.subplot(4,1,idx+1)
    plt.title(title[idx] + " channel stretching histogram")
    plt.plot(hist)
    plt.xlim([0,256])
plt.show()
```

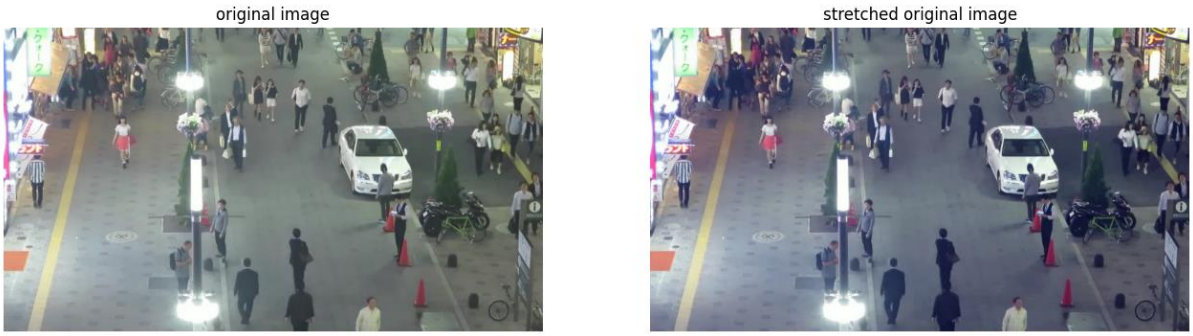

R channel stretching histogram
8000
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250


G channel stretching histogram
10000
8000
6000
4000
2000
0
0
50
100
150
200
250


B channel stretching histogram
8000
6000
4000
2000
0
0
50
100
150
200
250

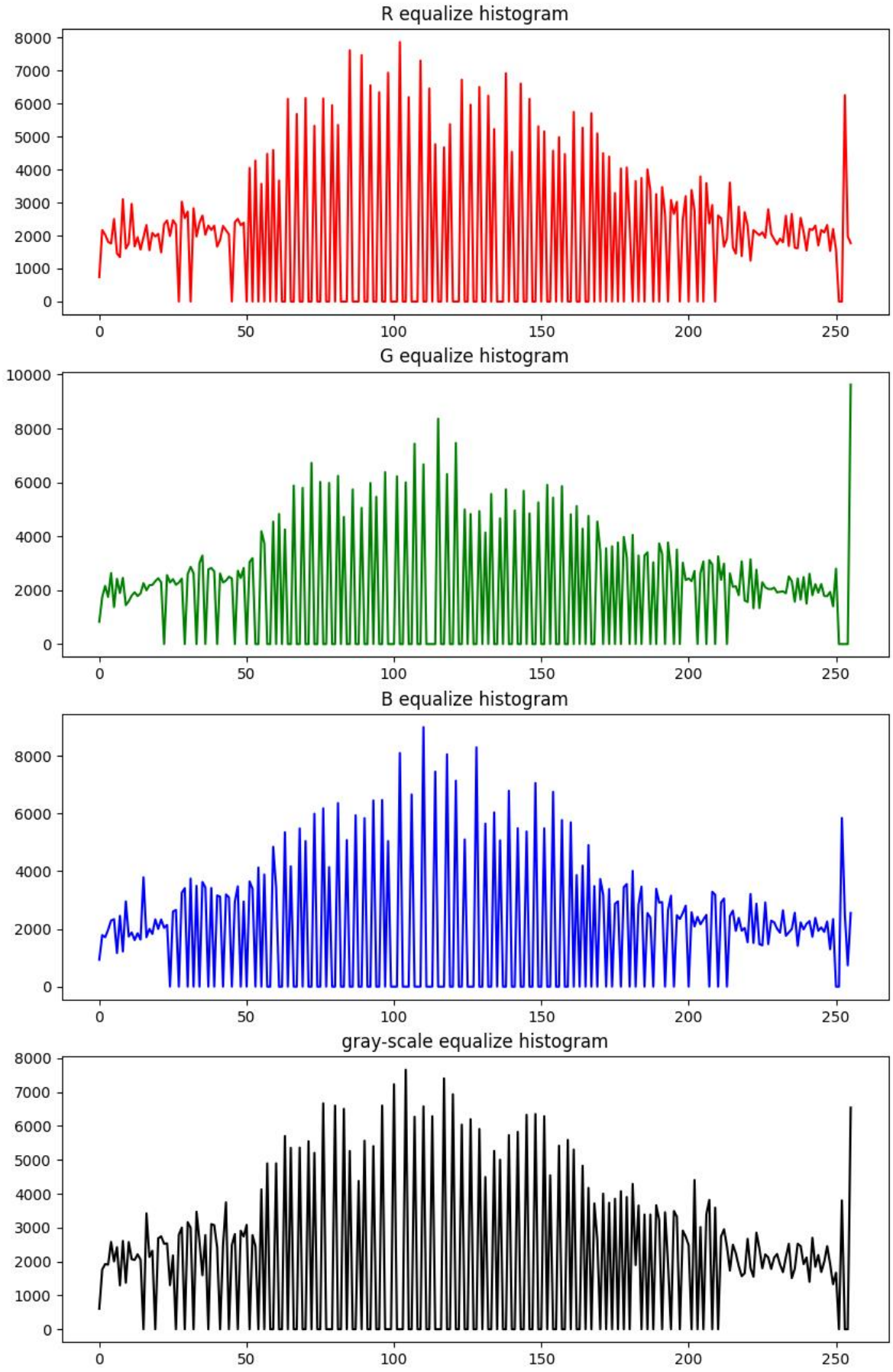

gray-scale channel stretching histogram
8000
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250

```
# 스트레칭 적용한 이미지 병합
RGB_img_stretch = cv2.merge([Red_img_stretch, Green_img_stretch, Blue_img_stretch])

plt.figure(figsize=(16,8))
plt.subplot(121), plt.axis("off"), plt.title("original image")
plt.imshow(RGB_img)
plt.subplot(122), plt.axis("off"), plt.title("stretched original image")
plt.imshow(RGB_img_stretch)
plt.show()
```

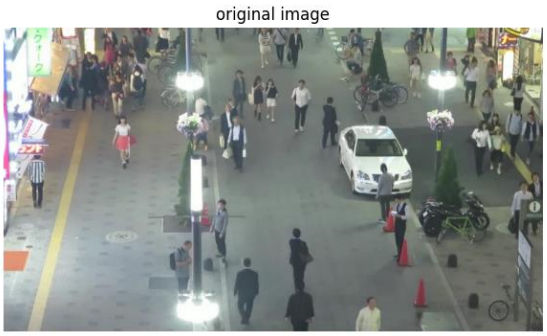


4-4) 히스토그램 평활화를 적용한 결과의 이미지와 평활화 후의 히스토그램 분포 그래프를 나타내시오.

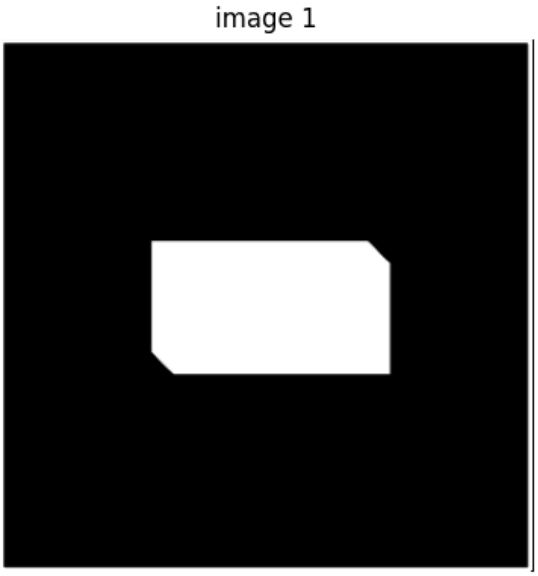
```
#히스토그램 평활화 (output_image 를 따로 정의하지 말고 동적으로 생성하자)
Red_img_equalize = cv2.equalizeHist(Red_img)
Green_img_equalize = cv2.equalizeHist(Green_img)
Blue_img_equalize = cv2.equalizeHist(Blue_img)
Gray_img_equalize = cv2.equalizeHist(gray_img)
output_img = [Red_img_equalize, Green_img_equalize, Blue_img_equalize, Gray_img_equalize]

plt.figure(figsize=(10, 10))
plt.axis('off'), plt.imshow(Gray_img_equalize, 'gray'), plt.title('equalized image')
plt.show()

plt.figure(figsize=(16,8))
plt.subplot(121), plt.axis("off"), plt.title("original gray-scale image")
plt.imshow(gray_img, 'gray')
plt.subplot(122), plt.axis("off"), plt.title("equalized gray-scale image")
plt.imshow(Gray_img_equalize, 'gray')
plt.show()

plt.figure(figsize=(10, 16))
for idx, img in enumerate(output_img):
    hist = cv2.calcHist([img],[0],None,[256],[0,256])
    plt.subplot(4,1,idx+1)
    plt.title(title[idx] + " channel equailize")
    plt.plot(hist)
    plt.xlim([0,256])
plt.show()
```

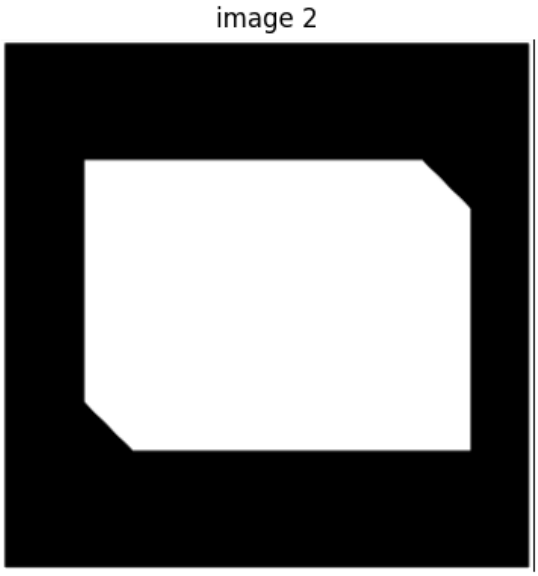

R equalize histogram
G equalize histogram
B equalize histogram
gray-scale equalize histogram

```
# 평활화 적용한 이미지 병합
RGB_img_equalize = cv2.merge([Red_img_equalize, Green_img_equalize, Blue_img_equalize])

plt.figure(figsize=(8,4))
plt.imshow(RGB_img_equalize)
plt.axis("off")
plt.title("equalized original image")
plt.show()
```



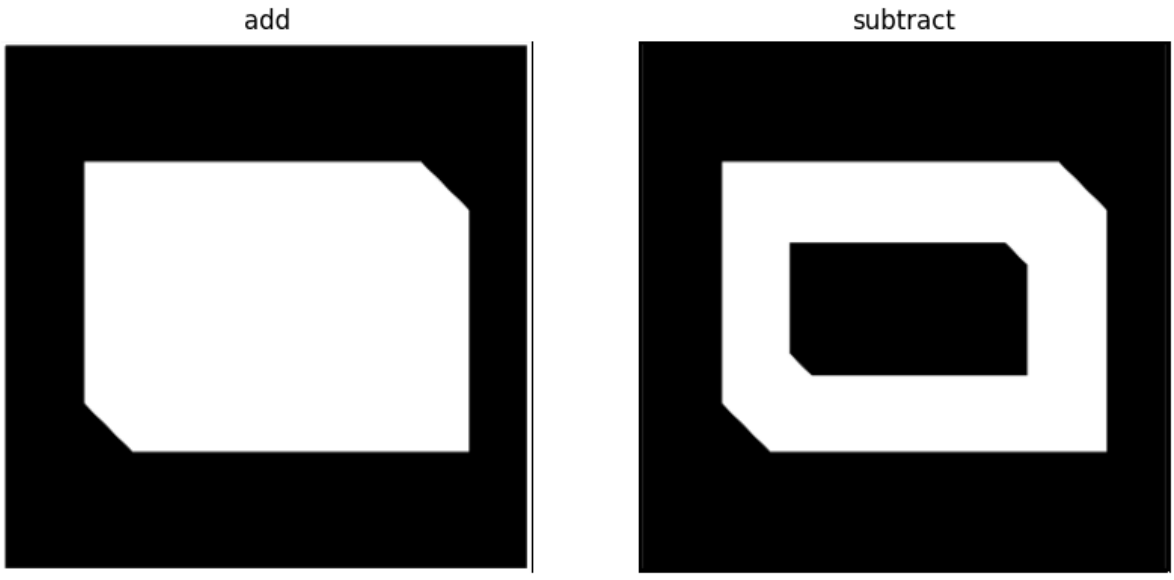
문제 5) Copy of Picture1.png 와 Copy of Picture2.png를 사용하여 연산을 수행하시오.





5-1, 5-2) 이미지 덧셈, 뺄셈

```
# 덧셈
add_img = cv2.add(img1, img2)

# 뺄셈
subtract_img = np.zeros_like(RGB_img1)
R_img1,G_img1,B_img1=cv2.split(RGB_img1)
R_img2,G_img2,B_img2=cv2.split(RGB_img2)

# 출력 array 생성하고 0으로 초기화, unsigned byte (0~255)로 설정
R_plus=np.zeros((RGB_img1.shape[0],RGB_img1.shape[1]),dtype=np.ubyte)
G_plus=np.zeros((RGB_img1.shape[0],RGB_img1.shape[1]),dtype=np.ubyte)
B_plus=np.zeros((RGB_img1.shape[0],RGB_img1.shape[1]),dtype=np.ubyte)

#for문을 돌며 픽셀 빼기 연산 하기
for h in range(RGB_img1.shape[0]):
  for w in range(RGB_img1.shape[1]):
    R_plus[h,w] = np.abs(np.int32(R_img1[h,w]) - np.int32(R_img2[h,w]))
    G_plus[h,w] = np.abs(np.int32(G_img1[h,w]) - np.int32(G_img2[h,w]))
    B_plus[h,w] = np.abs(np.int32(B_img1[h,w]) - np.int32(B_img2[h,w]))

subtract_img[:,:,0]=R_plus
subtract_img[:,:,1]=G_plus
subtract_img[:,:,2]=B_plus

plt.figure(figsize=(10,10))
plt.subplot(121), plt.imshow(add_img), plt.axis("off"), plt.title("add")
plt.subplot(122), plt.imshow(subtract_img), plt.axis("off"), plt.title("subtract")
plt.show()
```

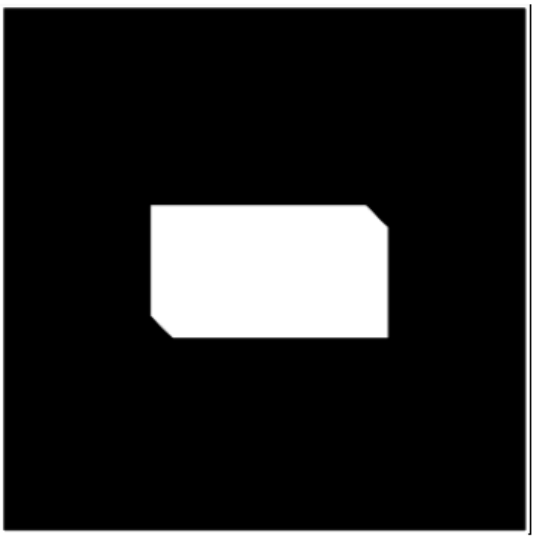


5-3, 5-4, 5-5) AND, OR, XOR 연산

```
def saturation(value): #saturation 함수로 정의하기
    if(value>255):
       value = 255;
    return value

# AND 연산
and_img = np.zeros_like(RGB_img1)
for h in range(RGB_img1.shape[0]):
  for w in range(RGB_img1.shape[1]):
    R_plus[h,w] = saturation(np.int32(R_img1[h,w]) & np.int32(R_img2[h,w]))
    G_plus[h,w] = saturation(np.int32(G_img1[h,w]) & np.int32(G_img2[h,w]))
    B_plus[h,w] = saturation(np.int32(B_img1[h,w]) & np.int32(B_img2[h,w]))

and_img[:,:,0]=R_plus
and_img[:,:,1]=G_plus
and_img[:,:,2]=B_plus

# OR 연산
or_img = np.zeros_like(RGB_img1)
for h in range(RGB_img1.shape[0]):
  for w in range(RGB_img1.shape[1]):
    R_plus[h,w] = saturation(np.int32(R_img1[h,w]) | np.int32(R_img2[h,w]))
    G_plus[h,w] = saturation(np.int32(G_img1[h,w]) | np.int32(G_img2[h,w]))
    B_plus[h,w] = saturation(np.int32(B_img1[h,w]) | np.int32(B_img2[h,w]))

or_img[:,:,0]=R_plus
or_img[:,:,1]=G_plus
or_img[:,:,2]=B_plus

# XOR 연산
xor_img = np.zeros_like(RGB_img1)
for h in range(RGB_img1.shape[0]):
  for w in range(RGB_img1.shape[1]):
    R_plus[h,w] = saturation(np.int32(R_img1[h,w]) ^ np.int32(R_img2[h,w]))
    G_plus[h,w] = saturation(np.int32(G_img1[h,w]) ^ np.int32(G_img2[h,w]))
    B_plus[h,w] = saturation(np.int32(B_img1[h,w]) ^ np.int32(B_img2[h,w]))

or_img[:,:,0]=R_plus
or_img[:,:,1]=G_plus
or_img[:,:,2]=B_plus

plt.figure(figsize=(10,10))
plt.subplot(121), plt.imshow(and_img), plt.axis("off"), plt.title("AND")
plt.subplot(122), plt.imshow(or_img), plt.axis("off"), plt.title("OR")
plt.show()
```

```
plt.figure(figsize=(8,4))
plt.imshow(and_img), plt.axis("off"), plt.title("XOR")
plt.show()
```

AND



OR

XOR