4장. 이미지 프로세싱 기초

세부목차

ROI(Region of Interest)

❖ ROI

- Region Of Interest
  - 관심영역
  - 이미지 작업 대상 : Region(영역), No All Pixel
  - 연산 속도 향상
- Numpy slicing으로 특정 영역 대상화
  - img[y:y+h, x:x+w]
    - 영역 픽셀 접근은 slicing 이 효과적
  - img.item(h, w, c), img.itemset((h, w, c), val)
    - 개별 픽셀 접근은 item() 함수가 효과적
  - 원본 참조 반환
    - Python list와 다르다.
    - ROI를 수정하면 원본도 바뀐다.
    - 원본을 수정하면 ROI도 바뀐다.
    - 수정을 원치 않으면 복사해서 사용 (img.copy())

ROI(Region of Interest)

❖ ROI(Region Of Image)

- ROI 지정

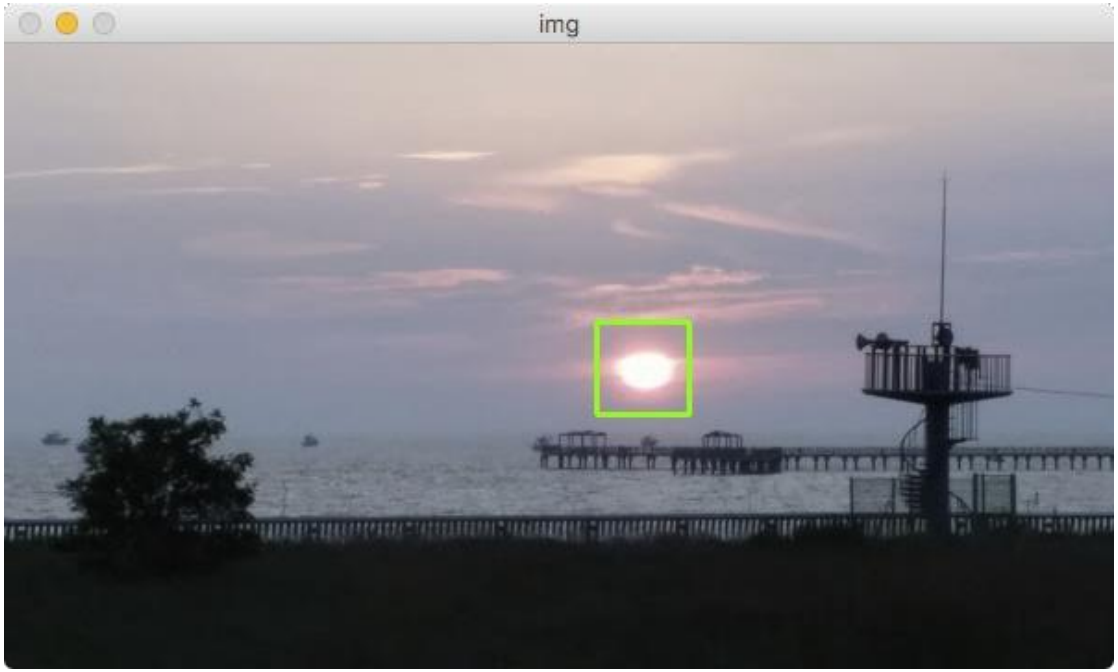
```
import cv2
import numpy as np
img = cv2.imread('./img/sunset.jpg')
x=320; y=150; w=50; h=50 # roi 좌표
roi = img[y:y+h, x:x+w] # roi 지정 ---①
print(roi.shape) # roi shape, (50,50,3)
cv2.rectangle(roi, (0,0), (h-1, w-1), (0,255,0)) # roi 전체에 사각형 그리기 ---②
cv2.imshow("img", img)
key = cv2.waitKey(0)
print(key)
cv2.destroyAllWindows()
```

[예제 4-1] 관심영역 지정(roi.py)

ROI(Region of Interest)

❖ ROI(Region Of Image)

- ROI 지정



[그림 4-1] 관심영역 표시

ROI(Region of Interest)

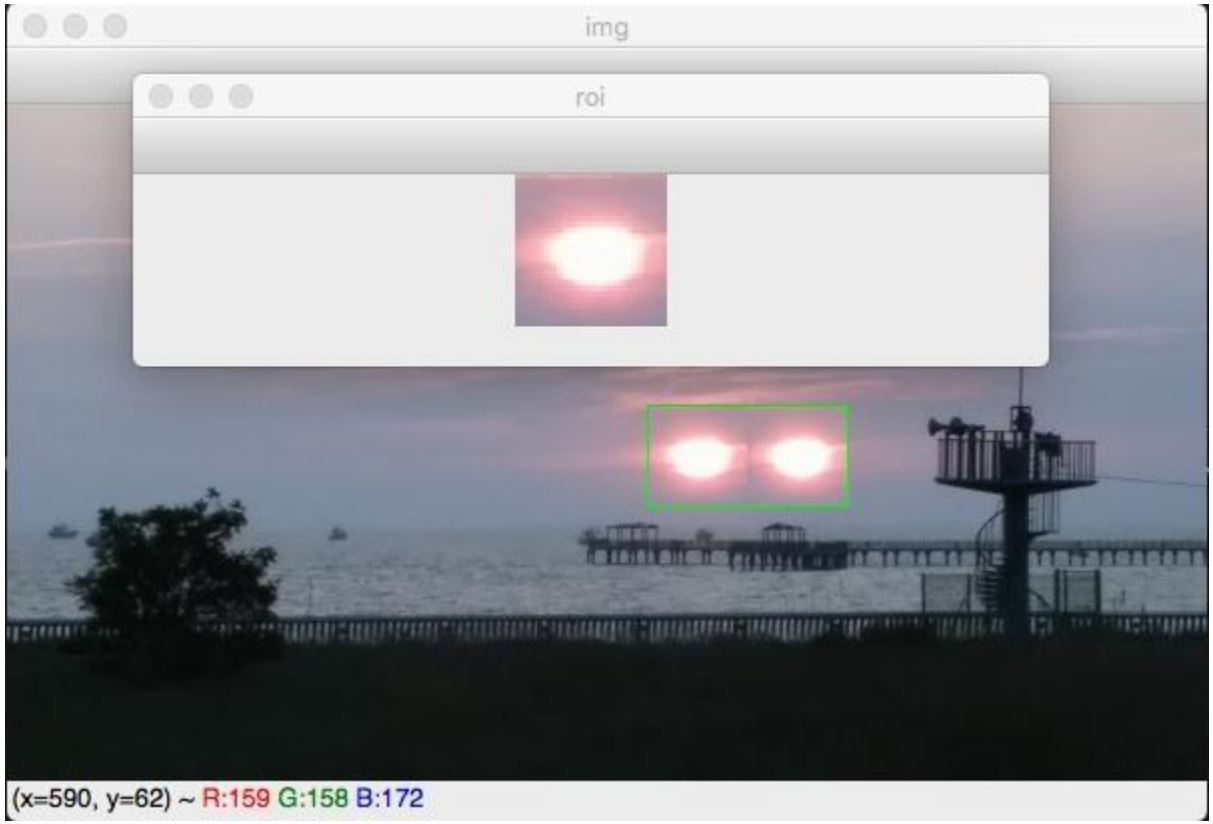
❖ ROI 복제 및 새창 띄우기

```
import cv2
import numpy as np
img = cv2.imread('../img/sunset.jpg')
x=320; y=150; w=50; h=50
roi = img[y:y+h, x:x+w] # roi 지정
img2 = roi.copy() # roi 배열 복제 ---①
img[y:y+h, x+w:x+w+w] = roi # 새로운 좌표에 roi 추가, 태양 2개 만들기
cv2.rectangle(img, (x,y), (x+w+w, y+h), (0,255,0)) # 2개의 태양 영역에 사각형 표시
cv2.imshow("img", img) # 원본 이미지 출력
cv2.imshow("roi", img2) # roi 만 따로 출력
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 4-2] 관심영역 복제 및 새 창 띄우기(roi_copy .py)

ROI(Region of Interest)

❖ ROI 복제 및 새창 띄우기



[그림 4-2] [예제 4-2] 의 실행 결과

ROI(Region of Interest)

❖ 마우스로 ROI 지정(1/3)

[예제 4-3] 마우스로 관심영역 지정(roi_copy_mouse.py)

```
import cv2
import numpy as np
isDragging = False # 마우스 드래그 상태 저장
x0, y0, w, h = -1,-1,-1,-1 # 영역 선택 좌표 저장
blue, red = (255,0,0),(0,0,255) # 색상 값
def onMouse(event,x,y,flags,param): # 마우스 이벤트 핸들 함수 ---①
    global isDragging, x0, y0, img # 전역변수 참조
    if event == cv2.EVENT_LBUTTONDOWN: # 왼쪽 마우스 버튼 다운, 드래그 시작 ---②
        isDragging = True
        x0 = x
        y0 = y
```

ROI(Region of Interest)

❖ 마우스로 ROI 지정(2/3)

```
elif event == cv2.EVENT_MOUSEMOVE: # 마우스 움직임 ---③
    if isDragging: # 드래그 진행 중
        img_draw = img.copy() # 사각형 그림 표현을 위한 이미지 복제
        cv2.rectangle(img_draw, (x0, y0), (x, y), blue, 2) # 드래그 진행 영역 표시
        cv2.imshow('img', img_draw) # 사각형 표시된 그림 화면 출력
elif event == cv2.EVENT_LBUTTONUP: # 왼쪽 마우스 버튼 업 ---④
    if isDragging: # 드래그 중지
        isDragging = False
        w = x - x0 # 드래그 영역 폭 계산
        h = y - y0 # 드래그 영역 높이 계산
        print("x:%d, y:%d, w:%d, h:%d" % (x0, y0, w, h))
        if w > 0 and h > 0: # 폭과 높이가 음수이면 드래그 방향이 옳음
            img_draw = img.copy() # 선택 영역에 사각형 그림을 표시할 이미지 복제
            # 선택 영역에 빨간 사각형 표시
            cv2.rectangle(img_draw, (x0, y0), (x, y), red, 2)
            cv2.imshow('img', img_draw) # 빨간 사각형 그려진 이미지 화면 출력
```

ROI(Region of Interest)

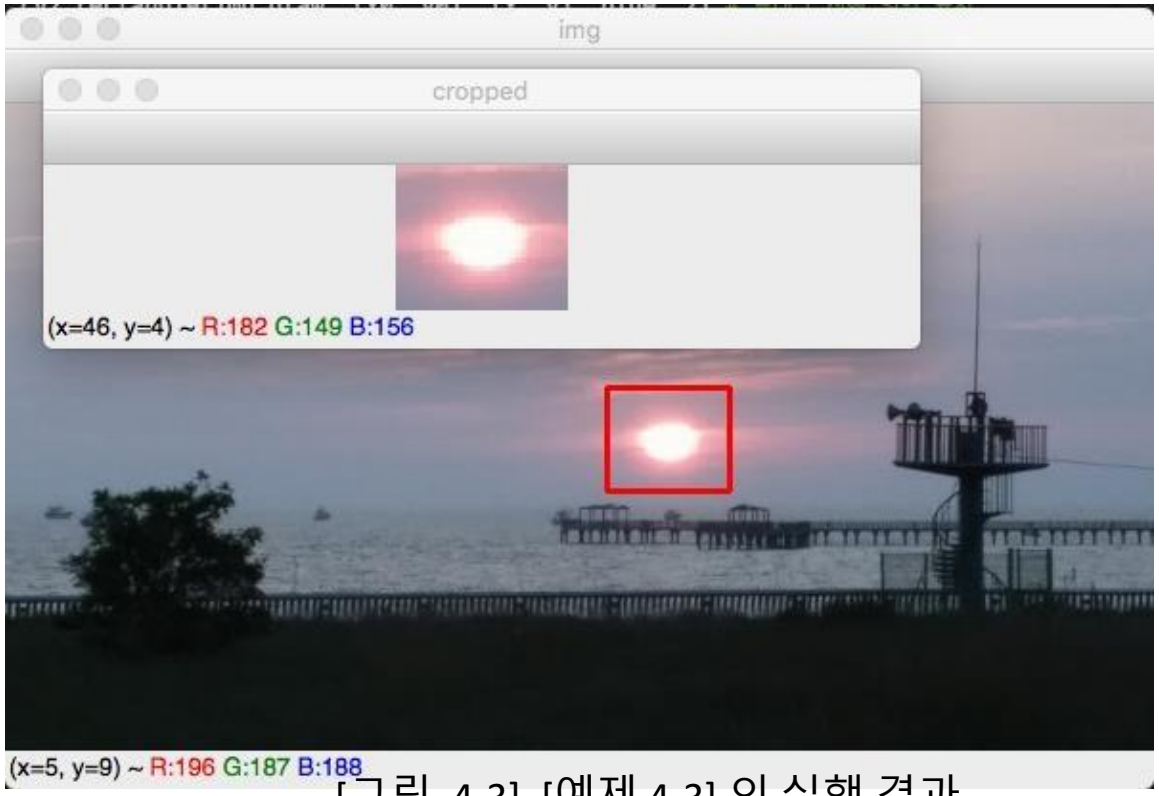
❖ 마우스로 ROI 지정(3/3)

```
                roi = img[y0:y0+h, x0:x0+w] # 원본 이미지에서 선택 영영만 ROI로 지정
                cv2.imshow('cropped', roi) # ROI 지정 영역을 새창으로 표시
                cv2.moveWindow('cropped', 0, 0) # 새창을 화면 좌측 상단에 이동
                cv2.imwrite('./cropped.jpg', roi) # ROI 영역만 파일로 저장
                print("croped.")
            else:
 # 드래그 방향이 잘못된 경우 사각형 그림이 없는 원본 이미지 출력
                cv2.imshow('img', img)
                print("좌측 상단에서 우측 하단으로 영역을 드래그 하세요.")
img = cv2.imread('../img/sunset.jpg')
cv2.imshow('img', img)
cv2.setMouseCallback('img', onMouse) # 마우스 이벤트 등록
cv2.waitKey()
cv2.destroyAllWindows()
```

ROI(Region of Interest)

❖ **마우스로 ROI 지**

```
x:193, y:174, w:-56, h:-77
좌측 상단에서 우측 하단으로 영역을 드래그 하세요.
x:314, y:148, w:64, h:54
croped.
```



[그림 4-3] [예제 4-3] 의 실행 결과

ROI(Region of Interest)

❖ selectROI

- 손쉬운 마우스 ROI 선택
- ret=cv2.selectROI([win_name,] img[, showCrossHair, fromCenter])
  - win_name : ROI 선택을 진행할 창의 이름, str
  - img : ROI 선택을 진행할 이미지, NumPy ndarray
  - showCrossHair=True : 선택 영역 중심에 십자 모양 표시 여부
  - fromCenter=False : 마우스 시작 지점을 영역의 중심으로 지정
  - ret : 선택한 영역 좌표와 크기 (x, y, w, h), 선택을 취소한 경우 모두 0
- Space or Enter : 선택 완료
- key board 'C' : 선택 취소

ROI(Region of Interest)

❖ selectROI

```
import cv2, numpy as np

img = cv2.imread('../img/sunset.jpg')

x,y,w,h =  cv2.selectROI('img', img, False)
if w and h:
    roi = img[y:y+h, x:x+w]
    cv2.imshow('cropped', roi) # ROI 지정 영역을 새창으로 표시
    cv2.moveWindow('cropped', 0, 0) # 새창을 화면 좌측 상단에 이동
    cv2.imwrite('./cropped2.jpg', roi) # ROI 영역만 파일로 저장

cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

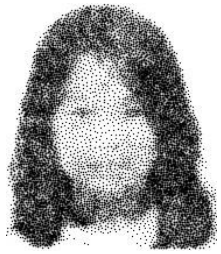
[예제 4-4] select ROI로 관심영역 지정(roi_select _img.py)

세부목차

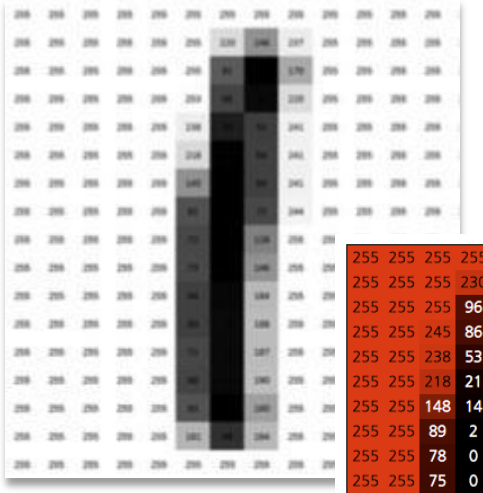
Color Space

❖ Digital Image

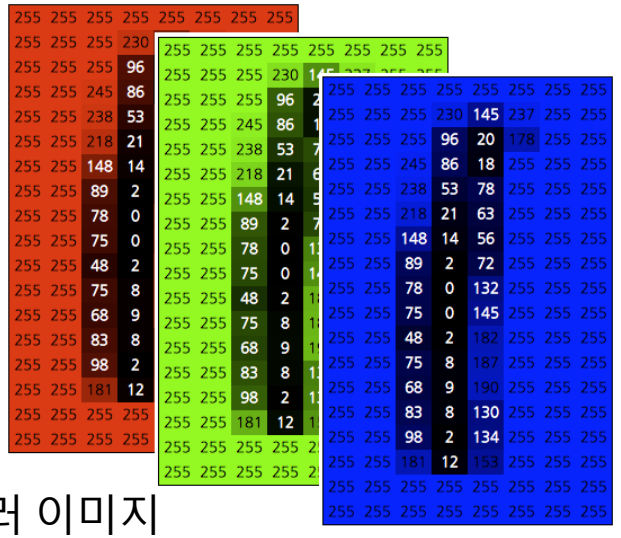
- Binary Image
  - 1px per 1bit (white or black),
  - 점의 밀도로 명암 표현
- Grayscale Image
  - 1px per 1byte=8bit
  - 255가지 명암 표현
- Color Image
  - 1px per 3byte=24bit
  - 16,777,216 색상 표현
  - 다양한 표현 방법 존재



[그림 4-4] 바이너리 이미지



[그림 4-5] 그레이 스케일 이미지

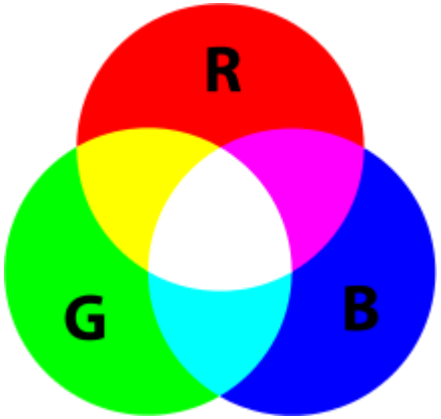


[그림 4-6] 컬러 이미지

Color Space

❖ RGB, BGR, RGBA

- RGB
  - 가장 널리 사용하는 Color Space
  - row x column x channel
  - 각 색상은 0 ~ 255
- BGR
  - OpenCV는 RGB의 순서를 거꾸로 사용
- RGBA
  - RGB + Alpha
  - Alpha : 투명도 표현
  - 파일이 RGBA 이고, cv2.IMREAD_UNCHANGED flag 사용 => BGRA



[그림 4-7] RGB 컬러 스페이스

Color Space

❖ BGR, BGRA, Alpha

[예제 4-5] BGR, BGRA, Ahlpha 채널 (rgba .py)

```
import cv2
import numpy as np
img = cv2.imread('../img/opencv_logo.png') # 기본 값 옵션
bgr = cv2.imread('../img/opencv_logo.png', cv2.IMREAD_COLOR) # IMREAD_COLOR 옵션
# IMREAD_UNCHANGED 옵션
bgra = cv2.imread('../img/opencv_logo.png', cv2.IMREAD_UNCHANGED)
# 각 옵션에 따른 이미지 shape
print("default", img.shape, "color", bgr.shape, "unchanged", bgra.shape)
cv2.imshow('bgr', bgr)
cv2.imshow('bgra', bgra)
cv2.imshow('alpha', bgra[:,:,3]) # 알파 채널만 표시
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[그림 4-9] [예제 4-5]의 실행 결과

Color Space

❖ Color 변환 함수

- out = cv2.cvtColor(img, flag)img : Numpy array, 변환할 이미지
  - flag : 변환할 컬러 스페이스, cv2.COLOR_ 로 시작하는 이름, 274개
    - cv2.COLOR_BGR2GRAY : BGR 컬러 이미지를 그레이 스케일로 변환
    - cv2.COLOR_GRAY2BGR : 그레이 스케일 이미지를 BGR 컬러 이미지로 변환
    - cv2.COLOR_BGR2RGB : BGR 컬러 이미지를 RGB 이미지로 변환
    - cv2.COLOR_BGR2HSV : BGR 컬러 이미지를 HSV 이미지로 변환
    - cv2.COLOR_HSV2BGR : HSV 컬러 이미지를 BGR로 변환
    - cv2.COLOR_BGR2YUV : BGR 컬러 이미지를 YUV로 변환
    - cv2.COLOR_YUV2BGR : YUV 컬러 이미지를 BGR로 변환
    - [i for i in dir(cv2) if i.startswith('COLOR_')]
  - out : 변환한 결과 이미지(NumPy 배열)

Color Space

❖ **BGR to Gray**

- cv2.COLOR_BGR2GRAY

[그림 4-10] [예제 4-6]의 실행 결과


```
import cv2
import numpy as np
img = cv2.imread('../img/girl.jpg')
img2 = img.astype(np.uint16) # dtype 변경 ---①
b,g,r = cv2.split(img2) # 채널 별로 분리 ---②
#b,g,r = img2[:,:,0], img2[:,:,1], img2[:,:,2]
gray1 = ((b + g + r)/3).astype(np.uint8) # 평균 값 연산후 dtype 변경 ---③
gray2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # BGR을 그레이 스케일로 변경 ---④
cv2.imshow('original', img)
cv2.imshow('gray1', gray1)
cv2.imshow('gray2', gray2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

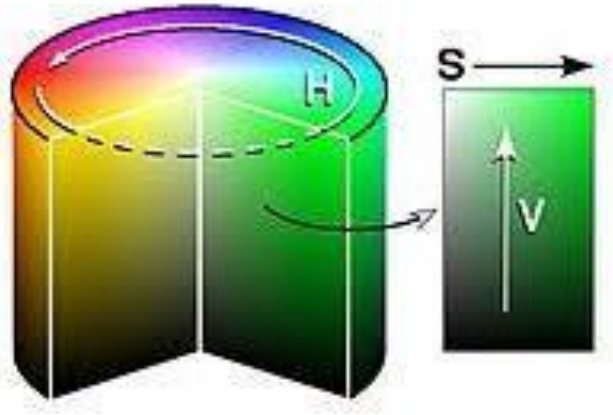
[예제 4-6] BGR을 그레이 스케일로 변환(bgr2gray.py)

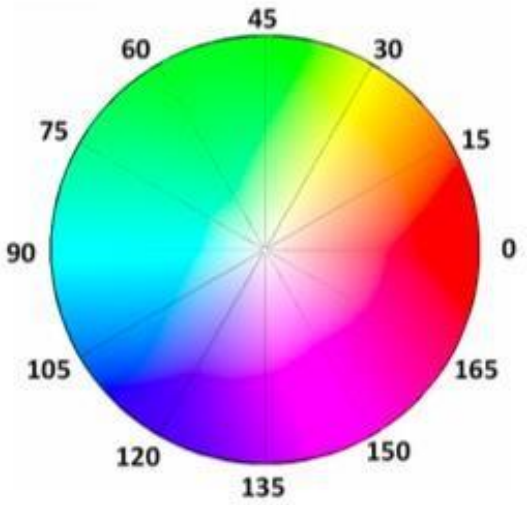
Color Space

❖ HSV, HSI, HSL

- 컬러의 3가지 특성을 분리
- 원통형 시스템 채널로 표현
- Hue(색상) : 0~179
- Saturation(채도) :0~255
- Value(명도) : 0~255
- Color Range
  - RED : 165~15, 0 ~ 15
  - Green : 45~75
  - Blue : 90~120
- cv2.COLOR_BGR2HSV

[그림 4-11] HSV 컬러 스페이스





[그림 4-12] 위에서 바라본 HsV 컬러 원통

Color Space

❖ BGR to HSV

```
import cv2
import numpy as np
#---① BGR 컬러 스페이스로 원색 픽셀 생성
red_bgr = np.array([[[0,0,255]]], dtype=np.uint8) # 빨강 값만 갖는 픽셀
green_bgr = np.array([[[0,255,0]]], dtype=np.uint8) # 초록 값만 갖는 픽셀
blue_bgr = np.array([[[255,0,0]]], dtype=np.uint8) # 파랑 값만 갖는 픽셀
yellow_bgr = np.array([[[0,255,255]]], dtype=np.uint8) # 노랑 값만 갖는 픽셀
#---② BGR 컬러 스페이스를 HSV 컬러 스페이스로 변환
red_hsv = cv2.cvtColor(red_bgr, cv2.COLOR_BGR2HSV);
green_hsv = cv2.cvtColor(green_bgr, cv2.COLOR_BGR2HSV);
blue_hsv = cv2.cvtColor(blue_bgr, cv2.COLOR_BGR2HSV);
yellow_hsv = cv2.cvtColor(yellow_bgr, cv2.COLOR_BGR2HSV);
#---③ HSV로 변환한 픽셀 출력
print("red:",red_hsv)
print("green:", green_hsv)
print("blue", blue_hsv)
print("yellow", yellow_hsv)
```

[예제 4-7] BGR에서 HSV로 변환(bgr2hsv.py)

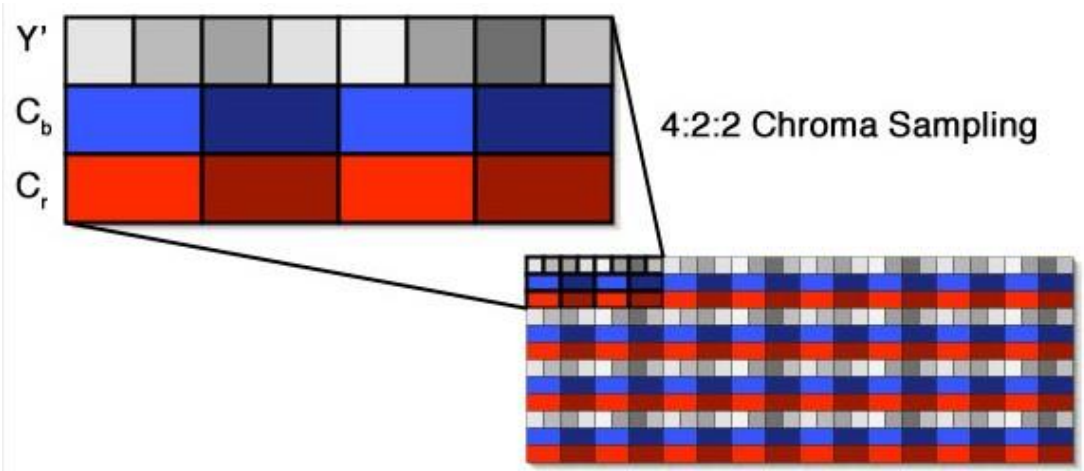
```
red: [[[   0 255 255]]]
green: [[[ 60 255 255]]]
blue [[[120 255 255]]]
yellow [[[ 30 255 255]]]
```

출력 결과

Color Space

❖ YUV, YCbCr

- 사람은 밝기에 더 민감, 상대적으로 색상은 둔감
- Y : 밝기
- U(chroma Blue, Cb) : 밝기와 파랑색과의 색상 차
- V(Chroma Red, Cr) : 밝기와 빨강색과의 색상 차
- 데이타 압축 효과
  - Y에 많은 비트 할당, UV에 적은 비트 할당,
- cv2.COLOR_BGR2YUV
- cv2.COLOR_BGR2YCrCb



[그림 4-13] YUV 개념도

Color Space

❖ BGR to YUV

```
import cv2
import numpy as np
#---① BGR 컬러 스페이스로 3가지 밝기의 픽셀 생성
dark = np.array([[[0,0,0]]], dtype=np.uint8) # 3 채널 모두 0인 가장 어두운 픽셀
middle = np.array([[[127,127,127]]], dtype=np.uint8) # 3 채널 모두 127인 중간 밝기 픽셀
bright = np.array([[[255,255,255]]], dtype=np.uint8) # 3 채널 모두 255인 가장 밝은 픽셀
#---② BGR 컬러 스페이스를 YUV 컬러 스페이스로 변환
dark_yuv = cv2.cvtColor(dark, cv2.COLOR_BGR2YUV)
middle_yuv = cv2.cvtColor(middle, cv2.COLOR_BGR2YUV)
bright_yuv = cv2.cvtColor(bright, cv2.COLOR_BGR2YUV)
#---③ YUV로 변환한 픽셀 출력
print("dark:",dark_yuv)
print("middle:", middle_yuv)
print("bright", bright_yuv)
```

출력 결과

```
dark: [[[   0 128 128]]
middle: [[[127 128 128]]]
bright [[[255 128 128]]]
```

[예제 4-8] BGR에서 YUV로 변환(bgr2yuv.py)

세부목차

1. ROI(Region Of Interest)
2. Color Space
3. **Threshold**
4. Image Arithmetic
5. Histogram
6. Workshop

Thresholding

- **Binarization**
- **이진화**
  - Two Class Classification
  - 글씨와 종이
  - 배경과 전경
- **Global fixed thresholding**
  - 전역 고정 이진화 : 이미지 전체에 하나의 임계 값 적용
  - 최적의 임계값(Optimal Threshold) 결정 방법
  - Simple Thresholding : Trial and Error Method
  - Outsu's Method
- **Locally Adaptive Thresholding**
  - 지역 가변 이진화 : 각 픽셀 마다 서로 다른 임계 값 적용
  - 영상 영역마다 밝기가 서로 다른 경우

Thresholding

❖ Simple Global fixed threshold

- 임의의 임계(경계) 값 선정
- numpy masking
  - dst[src > thresh] =255
- ret, dst = cv.threshold(src, thresh, maxval, flag)
  - src : 입력 이미지
  - thresh : 임계(경계) 값
  - value : 임계 값 기준에 만족한 픽셀에 적용할 값
  - flag : cv2.THRESH_*
    - BINARY : px > thresh ? value : 0
    - BINARY_INV : px > thresh ? 0 : value
    - TRUNC : px > thresh ? value : px
    - TOZERO : px > thresh ? px : 0
    - TOZERO_INV : px > thresh ? 0 : px

Thresholding

❖ Simple Global fixed threshold

- example < 1/2 >

```
import cv2
import numpy as np
import matplotlib.pylab as plt

img = cv2.imread('../img/gray_gradient.jpg', cv2.IMREAD_GRAYSCALE)

_, t_bin = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
_, t_bininv = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
_, t_truc = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)
_, t_2zr = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)
_, t_2zrinv = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)
```

[예제 4-9] 바이너리 이미지 만들기(threshold.py)

Thresholding

❖ Simple Global fixed threshold

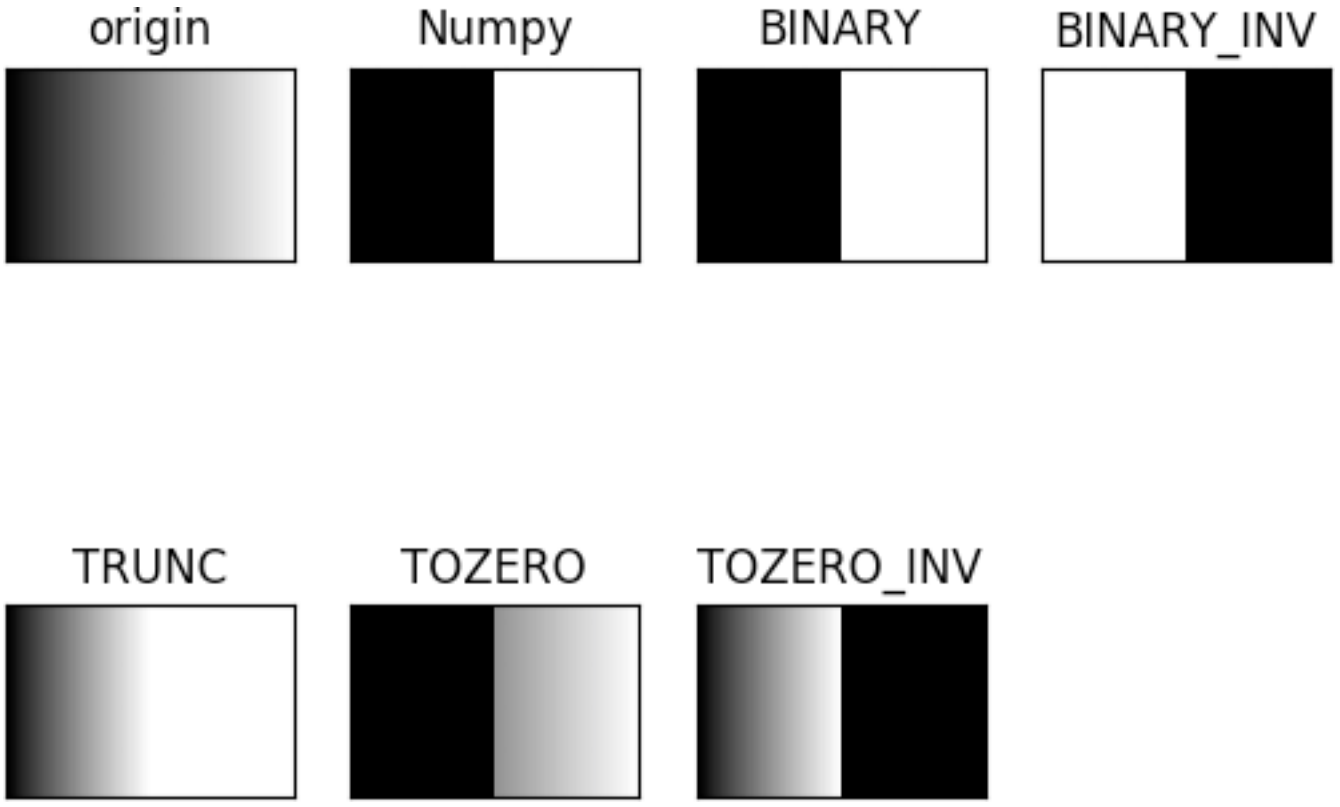
- example < 2/2 >

```
imgs = {'origin':img, 'BINARY':t_bin, 'BINARY_INV':t_bininv, \
'TRUNC':t_truc, 'TOZERO':t_2zr, 'TOZERO_INV':t_2zrinv}
for i, (key, value) in enumerate(imgs.items()):
    plt.subplot(2,3, i+1)
    plt.title(key)
    plt.imshow(value, cmap='gray')
    plt.xticks([]); plt.yticks([])

plt.show()
```

Thresholding

❖ Simple Global fixed threshold

- example < 결과 화면>



[그림 4-15] [예제 4-10]의 실행 결과

Thresholding

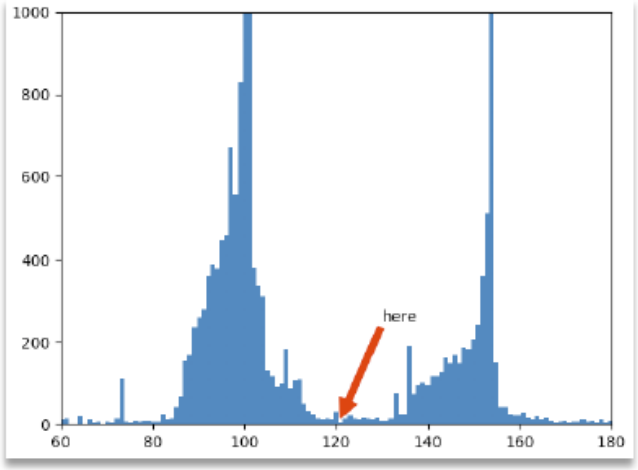
❖ Otsu's Binarization Method

- 최적의 Threshold 값 찾기

[그림 4-16] 여러 가지 경계 값으로 스레시 홀딩한 결과

Thresholding

❖ Otsu's Binarization Method



- Nobuyuki Otsu(오츠 노부유키) 제안, 1979
- 모든 픽셀을 밝기 기준으로 두 클래로 분류
- 중간 지점을 임계점으로 적용
- 방법
  - 임계값 t 로 영상 픽셀들을 두 클래스로 분류
    - t : 0~255
    - 각 클래스의 비율 : ω_1, ω_2
    - 각 클래스의 밝기 평균 : μ_1, μ_2
    - 각 클래스의 분산 : σ_1^2 , σ_2^2
    - Intra-Class Variance : $\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$
      - 최소가 되는 t 값
    - Inter-class Variance : $\sigma_b^2(t) = \omega_1(t)\omega_2(t)\,[\mu_1(t) - \mu_2(t)]^2$
      - 최대가 되는 t 값
  - cv2.threshold
    - flag : cv2.THRESH_OTSU, 반환 : T 값
    - 가우시안 필터 적용 후 사용 효과적

Thresholding

❖ Otsu's Binarization Method

```
import cv2
import numpy as np
import matplotlib.pylab as plt
# 이미지를 그레이 스케일로 읽기
img = cv2.imread('../img/scaned_paper.jpg', cv2.IMREAD_GRAYSCALE)
# 경계 값을 130으로 지정 ---①
_, t_130 = cv2.threshold(img, 130, 255, cv2.THRESH_BINARY)
# 경계 값을 지정하지 않고 OTSU 알고리즘 선택 ---②
t, t_otsu = cv2.threshold(img, -1, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
print('otsu threshold:', t) # Otsu 알고리즘으로 선택된 경계 값 출력
imgs = {'Original': img, 't:130':t_130, 'otsu:%d'%t: t_otsu}
for i , (key, value) in enumerate(imgs.items()):
    plt.subplot(1, 3, i+1)
    plt.title(key)
    plt.imshow(value, cmap='gray')
    plt.xticks([]); plt.yticks([])
plt.show()
```

[예제 4-11] 오츠의 알고리즘을 적용한 스레시홀드(threshold_otsu .py)

Thresholding

❖ Otsu's Binarization Method

Original

t:130

otsu:131

[그림 4-17] [예제 4-11]의 실행결과

Thresholding

❖ Adaptive Threshold

- 주변 화소 값에 따라 임계값 결정
- 주변 상황에 반응/적응
- cv2.adaptiveThreshold(src, maxVal, method, type, blockSize, C)
  - src : gray scaled image
  - maxVal : 임계 값 만족한 픽셀에 적용할 값
  - method : thredshold 결정 방법
    - cv2.ADPTIVE_THRESH_*
      - MEAN_C : 이웃 픽셀의 평균으로 결정
      - GAUSSIAN_C : 가우시안 분포에 따른 가중치의 합으로 결정
  - type : threshold type
  - blockSize : 이웃 영역 크기
  - C : 계산한 결과에 차감하여 임계값 결정

$$T(x,y)=\frac{1}{n^2}\sum_{x_i}\sum_{y_i} I(x+x_i,y+y_j) \; - \; C$$

Thresholding

❖ Adaptive Threshold

- sudoku scan example < 1/2 >

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
blk_size = 9 # 블럭 사이즈
C = 5 # 차감 상수
img = cv2.imread('../img/sudoku.png', cv2.IMREAD_GRAYSCALE) # 그레이 스케일로 읽기
# ---① 오츠의 알고리즘으로 단일 경계 값을 전체 이미지에 적용
ret, th1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
# ---② 어뎁티드 쓰레시홀드를 평균과 가우시안 분포로 각각 적용
th2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,\
                                        cv2.THRESH_BINARY, blk_size, C)
th3 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                        cv2.THRESH_BINARY, blk_size, C)
```

[예제 4-12]적용형 스레시홀드 적용(thresh_adaped .py)

Thresholding

❖ Adaptive Threshold

- sudoku scan example < 2/2 >

```
# ---③ 결과를 Matplot으로 출력
imgs = {'Original': img, 'Global-Otsu:%d'%ret:th1, \
            'Adapted-Mean':th2, 'Adapted-Gaussian': th3}
for i, (k, v) in enumerate(imgs.items()):
    plt.subplot(2,2,i+1)
    plt.title(k)
    plt.imshow(v,'gray')
    plt.xticks([]),plt.yticks([])
plt.show()
```

Thresholding

❖ Adaptive Threshold

- sudoku scan example < 결과 >

Original

Global

Mean

Gaussian

[그림 4-18] [예제 4-12]의 실행 결과

세부목차

Image Arithmetic

❖ 이미지 연산

- NumPy Broadcasting (+ , - , *, /)
  - 0 > 연산결과 > 255
- OpenCV 함수
  - cv2.add(src1, src2[, dest, mask, dtype])
    - src1 : 입력 영상1 또는 수
    - src2 : 입력 영상2 또는 수
    - out : 출력 영상
    - mask : 0이 아닌 픽셀만 연산
    - dtype : 출력 dtype
  - cv2.substract(src1, src2[, dest, mask, dtype])
    - cv2.add() 와 동일
  - cv2.multiply(src1, src2[, dest, scale, dtype])
    - scale : 연산 결과에 추가 연산할 값
  - cv2.divide(src1, src2[, dest, scale, dtype])
    - cv2.multiply()와 동일

Image Arithmetic

❖ Image Addition

```
import cv2
import numpy as np
# ---① 연산에 사용할 배열 생성
a = np.uint8([[200, 50]])
b = np.uint8([[100, 100]])
#---② NumPy 배열 직접 연산
add1 = a + b
sub1 = a - b
mult1 = a * 2
div1 = a / 3
# ---③ OpenCV API를 이용한 연산
add2 = cv2.add(a, b)
sub2 = cv2.subtract(a, b)
mult2 = cv2.multiply(a , 2)
div2 = cv2.divide(a, 3)
```

[예제 4-13]영상의 사칙 연산(arithmetic.py)

Image Arithmetic

❖ Image Addition

```
#---④ 각 연산 결과 출력
print(add1, add2)
print(sub1, sub2)
print(mult1, mult2)
print(div1, div2)
```

출력 결과

```
[ 44 150]] [[255 150]]
[[100 206]] [[100   0]]
[[144 100]] [[255 100]]
[[66.66666667 16.66666667]] [[67 17]]
```

Image Arithmetic

❖ Image Addition with Mask

```
import cv2
import numpy as np
#---① 연산에 사용할 배열 생성

a = np.array([[1, 2]], dtype=np.uint8)
b = np.array([[10, 20]], dtype=np.uint8)
#---② 2번째 요소가 0인 마스크 배열 생성
mask = np.array([[1, 0]], dtype=np.uint8)

#---③ 누적 할당과의 비교 연산
c1 = cv2.add( a, b , None, mask)
print(c1)
c2 = cv2.add( a, b , b.copy(), mask)
print(c2, b)
```

출력 결과

```
[[11  0]]
[[11 20]]
```

[예제 4-14] mask와 누적 할당 연산(arithmetic_mask.py)

Image Arithmetic

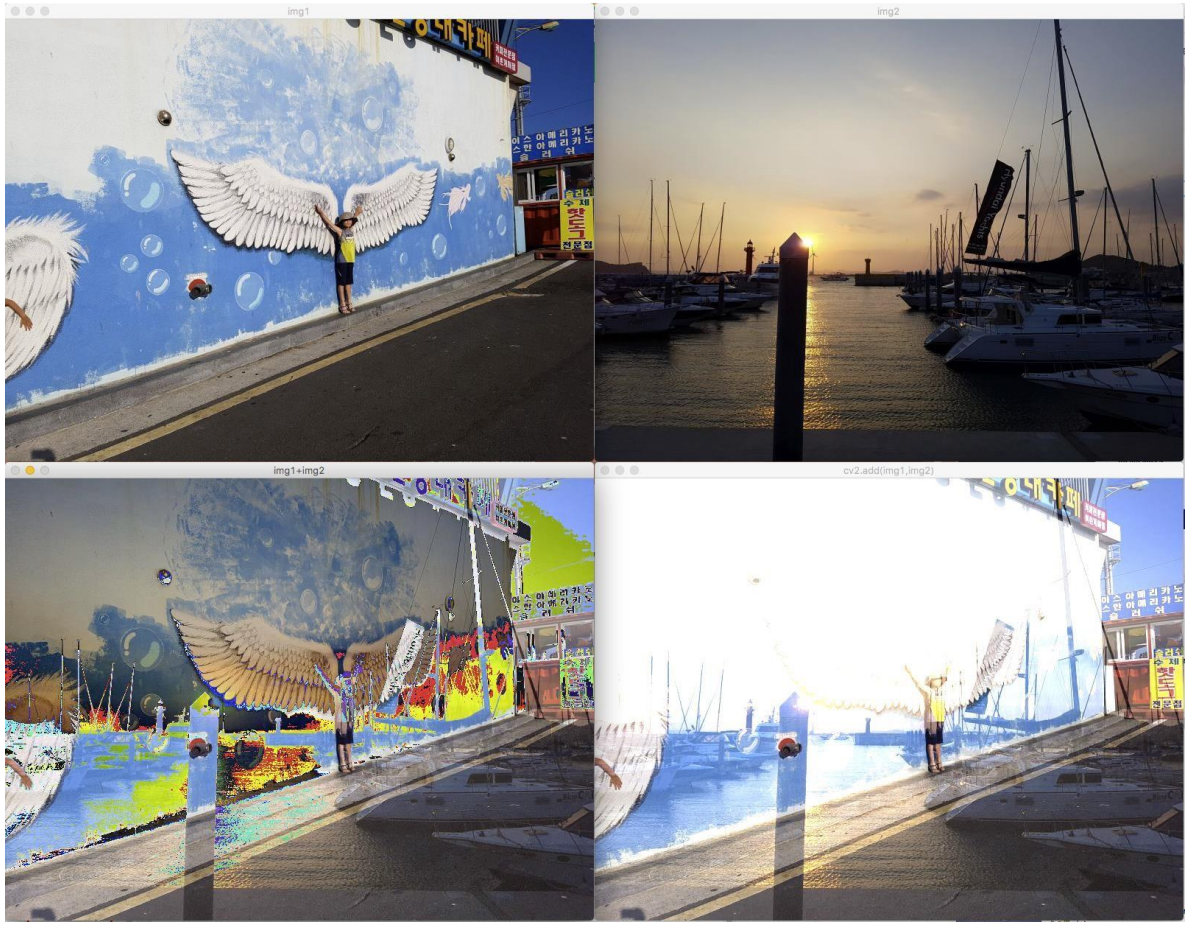
❖ Image Blendin

```
import cv2
import numpy as np
import matplotlib.pylab as plt
# ---① 연산에 사용할 이미지 읽기
img1 = cv2.imread('../img/wing_wall.jpg')
img2 = cv2.imread('../img/yate.jpg')
# ---② 이미지 덧셈
img3 = img1 + img2 # 더하기 연산
img4 = cv2.add(img1, img2) # OpenCV 함수
imgs = {'img1':img1, 'img2':img2, 'img1+img2': img3, 'cv.add(img1, img2)': img4}
# ---③ 이미지 출력
for i, (k, v) in enumerate(imgs.items()):
    plt.subplot(2,2, i + 1)
    plt.imshow(v[:,:,::-1])
    plt.title(k)
    plt.xticks([]); plt.yticks([])
plt.show()
```

[예제 4-15] 이미지 단순 합성(blending_simple.py)

Image Arithmetic

❖ Image Blending



[그림 4-19] [예제 4-15]의 실행 결과

Image Arithmetic

❖ Alpha Blending

- 2개의 이미지 혼합
- 가중치 지정, alpha : beta , alpha + beta = 1
- a*alpha + b*beta <= 255
- numpy 로 직접 구현

$$g(x) = (1 - \alpha) f_0(x) + \alpha f_1(x)$$

- cv2.addWeighted(img1, alpha, img2, beta, gamma)
  - img1, img2 : 합성할 2 영상
  - alpha : img1에 지정할 가중치(알파 값)
  - beta : img2에 지정할 가중치, 흔히 (1- alpha) 적용
  - gamma : 연산 결과에 추가할 상수, 흔히 0(zero) 적용

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

Image Arithmetic

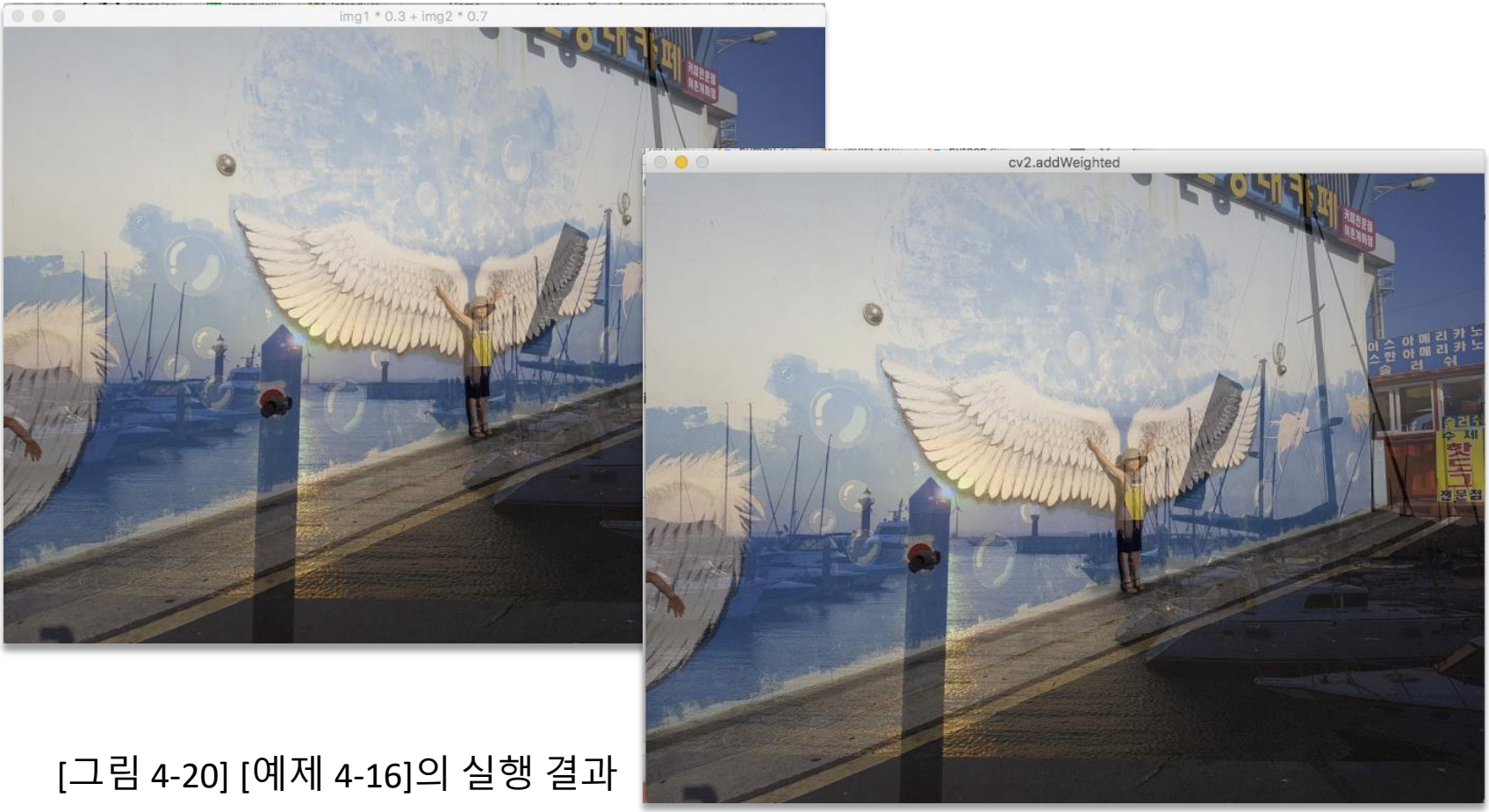
❖ Alpha Blending

```
import cv2
import numpy as np
alpha = 0.5 # 합성에 사용할 알파 값
#---① 합성에 사용할 영상 읽기
img1 = cv2.imread('../img/wing_wall.jpg')
img2 = cv2.imread('../img/yate.jpg')
# ---② NumPy 배열에 수식을 직접 연산해서 알파 블렌딩 적용
blended = img1 * alpha + img2 * (1-alpha)
blended = blended.astype(np.uint8) # 소수점 발생을 제거하기 위함
cv2.imshow('img1 * alpha + img2 * (1-alpha)', blended)
# ---③ addWeighted() 함수로 알파 블렌딩 적용
dst = cv2.addWeighted(img1, alpha, img2, (1-alpha), 0)
cv2.imshow('cv2.addWeighted', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 4-16] 50%의 알파 블렌딩(blending_alpha.py)

Image Arithmetic

❖ Image Blending



[그림 4-20] [예제 4-16]의 실행 결과

Image Arithmetic

❖ Image Blending –Track bar

```
import cv2
import numpy as np
win_name = 'Alpha blending' # 창 이름
trackbar_name = 'fade' # 트렉바 이름
# ---① 트렉바 이벤트 핸들러 함수
def onChange(x):
    alpha = x/100
    dst = cv2.addWeighted(img1, 1-alpha, img2, alpha, 0)
    cv2.imshow(win_name, dst)
# ---② 합성 영상 읽기
img1 = cv2.imread('../img/man_face.jpg')
img2 = cv2.imread('../img/lion_face.jpg')
# ---③ 이미지 표시 및 트렉바 붙이기
cv2.imshow(win_name, img1)
cv2.createTrackbar(trackbar_name, win_name, 0, 100, onChange)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-17] 트랙바로 알파 블렌딩(blending_alpha_trackbar.py)

Image Arithmetic

❖ **Image Blending –Track bar**

[그림 4-21] [예제 4-17]의 실행 결과

Image Arithmetic

❖ Bitwise Operation

- 2이미지의 각픽셀 대해 bitwise 연산
- 배경 제거, 전경과 배경 분리, 합성 등에 활용
  - cv2.bitwise_and(img1, img2, mask=None)
  - cv2.bitwise_or(img1, img2, mask=None)
  - cv2.bitwise_xor(img1, img2, mask=None)
  - cv2.bitwise_not(img1, mask=None)
    - mask (optional)
      - mask의 값이 0이 아닌 픽셀만 연산, 0인 부분은 mask 값 사용
      - binary image(흑백 이미지)
    - img1, img2, mask 의 rows, cols가 동일해야 한다.

Image Arithmetic

❖ Image Bitwise

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

#--① 연산에 사용할 이미지 생성
img1 = np.zeros( ( 200,400), dtype=np.uint8)
img2 = np.zeros( ( 200,400), dtype=np.uint8)
img1[:, :200] = 255 # 왼쪽은 검정색(0), 오른쪽은 흰색(255)
img2[100:200, :] = 255 # 위쪽은 검정색(0), 아래쪽은 흰색(255)

#--② 비트와이즈 연산
bitAnd = cv2.bitwise_and(img1, img2)
bitOr = cv2.bitwise_or(img1, img2)
bitXor = cv2.bitwise_xor(img1, img2)
bitNot = cv2.bitwise_not(img1)
```

[예제 4-18] 비트와이즈 연산(bitwise.py)

Image Arithmetic

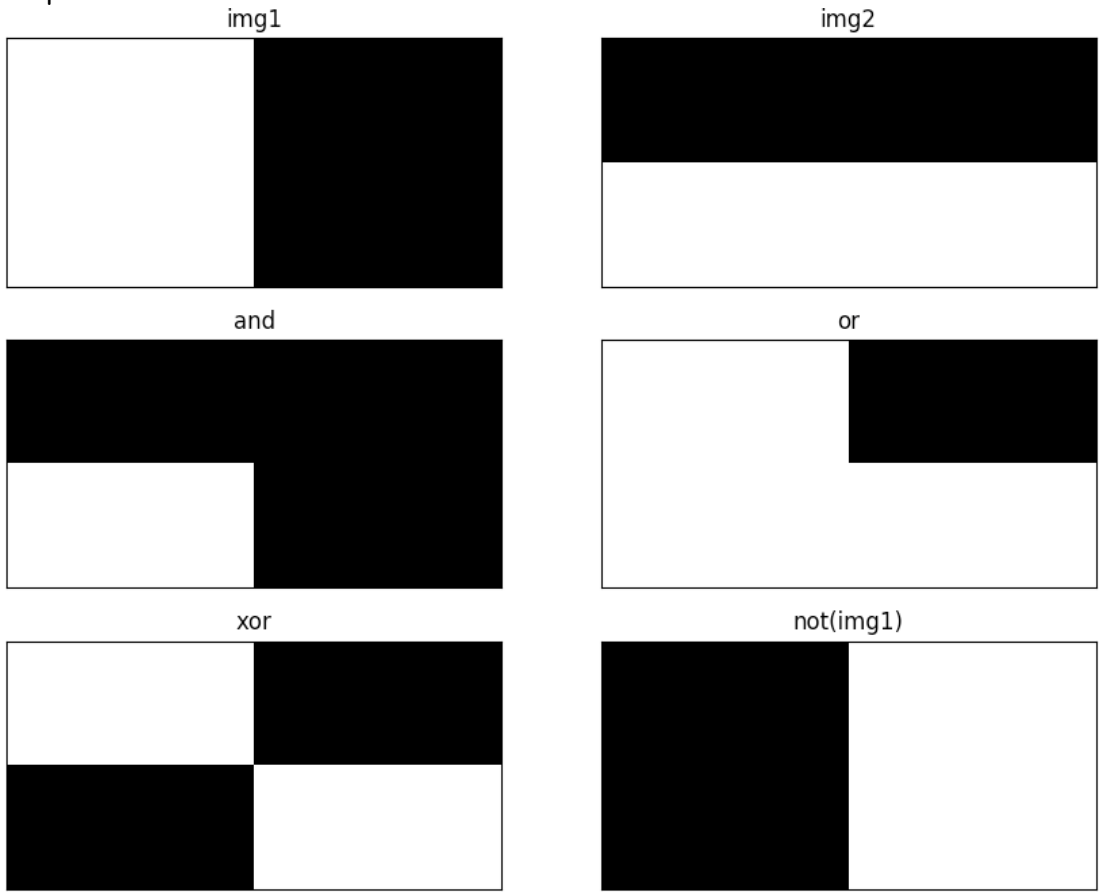
❖ Image Bitwise

```
#--③ Plot으로 결과 출력
imgs = {'img1':img1, 'img2':img2, 'and':bitAnd,
                'or':bitOr, 'xor':bitXor, 'not(img1)':bitNot}
for i, (title, img) in enumerate(imgs.items()):
    plt.subplot(3,2,i+1)
    plt.title(title)
    plt.imshow(img, 'gray')
    plt.xticks([]); plt.yticks([])

plt.show()
```

Image Arithmetic

❖ Image Bitwise

- bitwise example



[그림 4-22] [예제 4-18]의 실행 결과

Image Arithmetic

❖ Masking with Bitwise

```
import numpy as np, cv2
import matplotlib.pylab as plt
#--① 이미지 읽기
img = cv2.imread('../img/girl.jpg')
#--② 마스크 만들기
mask = np.zeros_like(img)
cv2.circle(mask, (150,140), 100, (255,255,255), -1)
#cv2.circle(대상이미지, (원점x, 원점y), 반지름, (색상), 채우기)
#--③ 마스킹
masked = cv2.bitwise_and(img, mask)
#--④ 결과 출력
cv2.imshow('original', img)
cv2.imshow('mask', mask)
cv2.imshow('masked', masked)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-19] bitwise_and 연산으로 마스킹하기(bitwise_masking.py)

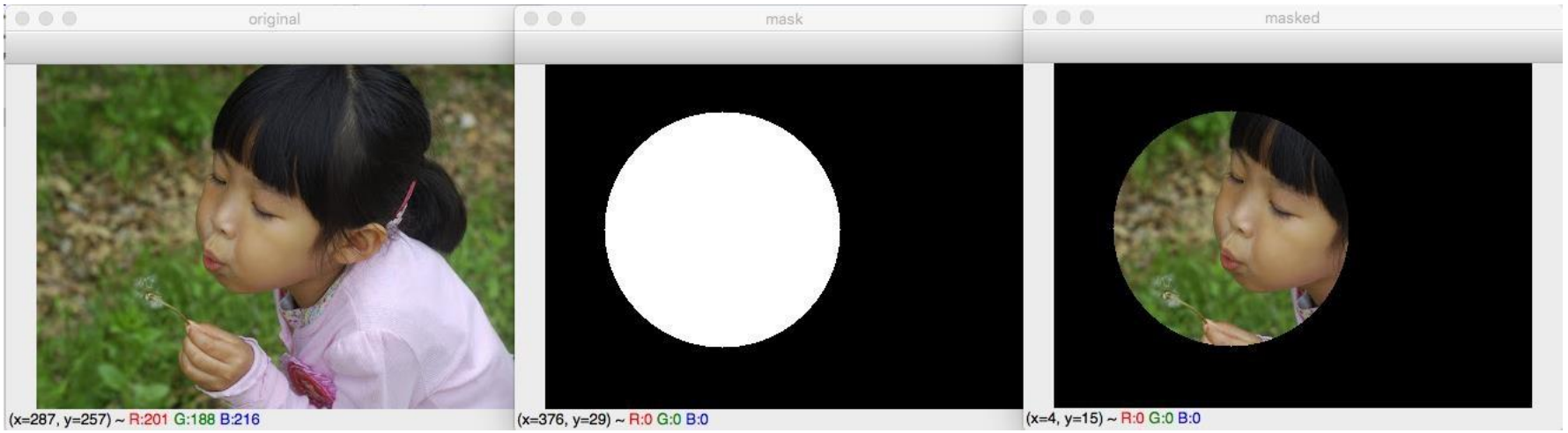
Image Arithmetic

❖ Masking with Bitwise

```
import numpy as np, cv2
import matplotlib.pylab as plt
#--① 이미지 읽기
img = cv2.imread('../img/girl.jpg')
#--② 마스크 만들기
mask = np.zeros(img.shape[:2], dtype=np.uint8)
cv2.circle(mask, (150,140), 100, (255), -1)
#cv2.circle(대상이미지, (원점x, 원점y), 반지름, (색상), 채우기)
#--③ 마스킹
masked = cv2.bitwise_and(img, img, mask=mask)
#--④ 결과 출력
cv2.imshow('original', img)
cv2.imshow('mask', mask)
cv2.imshow('masked', masked)
cv2.waitKey()
cv2.destroyAllWindows()
```

Image Arithmetic

❖ Masking with Bitwise



[그림 4-23] [예제 4-19]의 실행 결과

Image Arithmetic

❖ Diff Image

- 차영상, 영상에서 영상 빼기
- 틀린 그림 찾기
- diff = cv2.absdiff(img1, img2)
  - img1, img2 : 입력 영상
  - diff : 두 영상의 차의 절대 값 반환

Image Arithmetic

❖ Diff Image (도면 차이 찾기)

```
import numpy as np, cv2

#--① 연산에 필요한 영상을 읽고 그레이스케일로 변환
img1 = cv2.imread('../img/robot_arm1.jpg')
img2 = cv2.imread('../img/robot_arm2.jpg')
img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

#--② 두 영상의 절대값 차 연산
diff = cv2.absdiff(img1_gray, img2_gray)
#--③ 차 영상을 극대화 하기 위해 쓰레시홀드 처리 및 컬러로 변환

_, diff = cv2.threshold(diff, 1, 255, cv2.THRESH_BINARY)
diff_red = cv2.cvtColor(diff, cv2.COLOR_GRAY2BGR)
diff_red[:,:,2] = 0
```

[예제 4-20] 차영상으로 도면의 차이 찾아내기(diff_absolute.py)

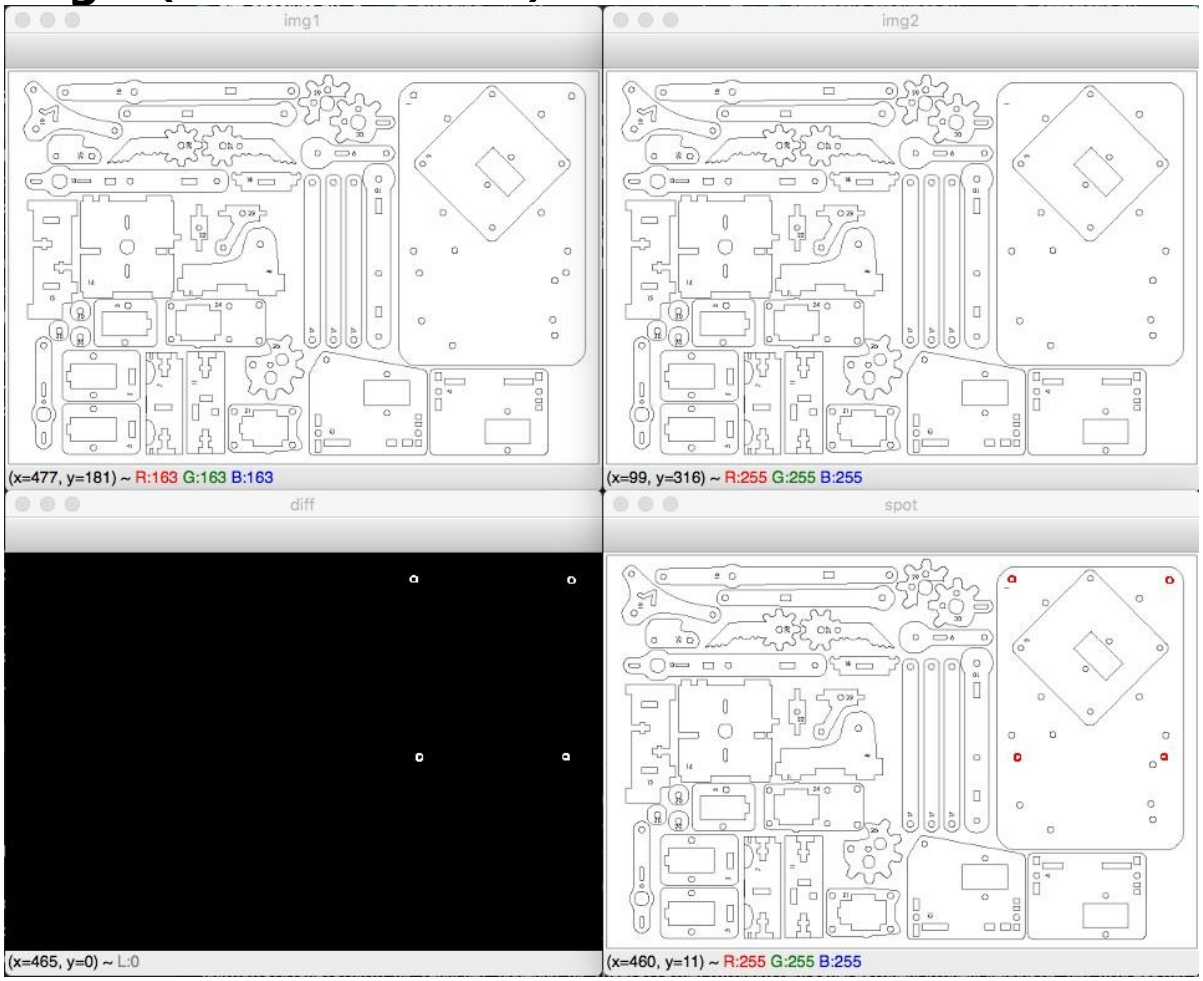
Image Arithmetic

❖ Diff Image (도면 차이 찾기)

```
#--④ 두 번째 이미지에 변화 부분 표시
spot = cv2.bitwise_xor(img2, diff_red)

#--⑤ 결과 영상 출력
cv2.imshow('img1', img1)
cv2.imshow('img2', img2)
cv2.imshow('diff', diff)
cv2.imshow('spot', spot)
cv2.waitKey()
cv2.destroyAllWindows()
```

Image Arithmetic

❖ **Diff Image (도면 차이 찾기)**



[그림 4-24] [예제 4-20]의 실행 결과

Image Arithmetic

❖ Blending with Alpha Channel(RGBA PNG)

- 이미지에 로고 합성하기

[그림 4-25] [예제 4-21]의 실행 결과

Image Arithmetic

❖ HSV Color Masking

- HSV 컬러 스페이스를 이용한 색상 마스크
- dst=cv2.inRange(src, lowerb, upperb)
  - src : 원본 배열
  - low와 upper : 구간
  - dst : 원본의 구간 밖 요소는 zero(0) 로 표시

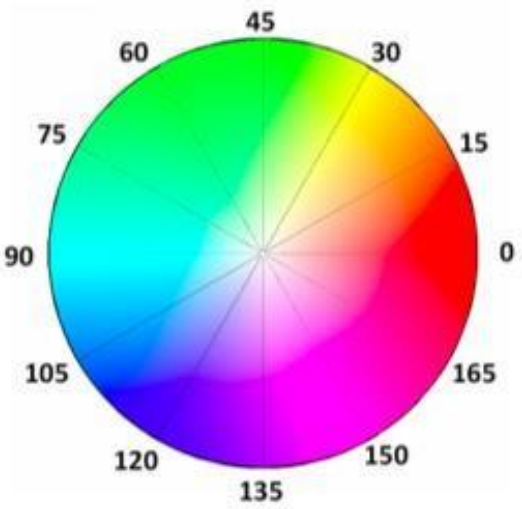


Image Arithmetic

❖ HSV Color Masking

- 색상별로 분류하기 <1/2>

```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 큐브 영상 읽어서 HSV로 변환
img = cv2.imread("../img/cube.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
#--② 색상별 영역 지정
blue1 = np.array([90, 50, 50])
blue2 = np.array([120, 255,255])
green1 = np.array([45, 50,50])
green2 = np.array([75, 255,255])
red1 = np.array([0, 50,50])
red2 = np.array([15, 255,255])
red3 = np.array([165, 50,50])
red4 = np.array([180, 255,255])
yellow1 = np.array([20, 50,50])
yellow2 = np.array([35, 255,255])
```

[예제 4-22] HSV 색상으로 마스킹(hsv_color_mask.py)

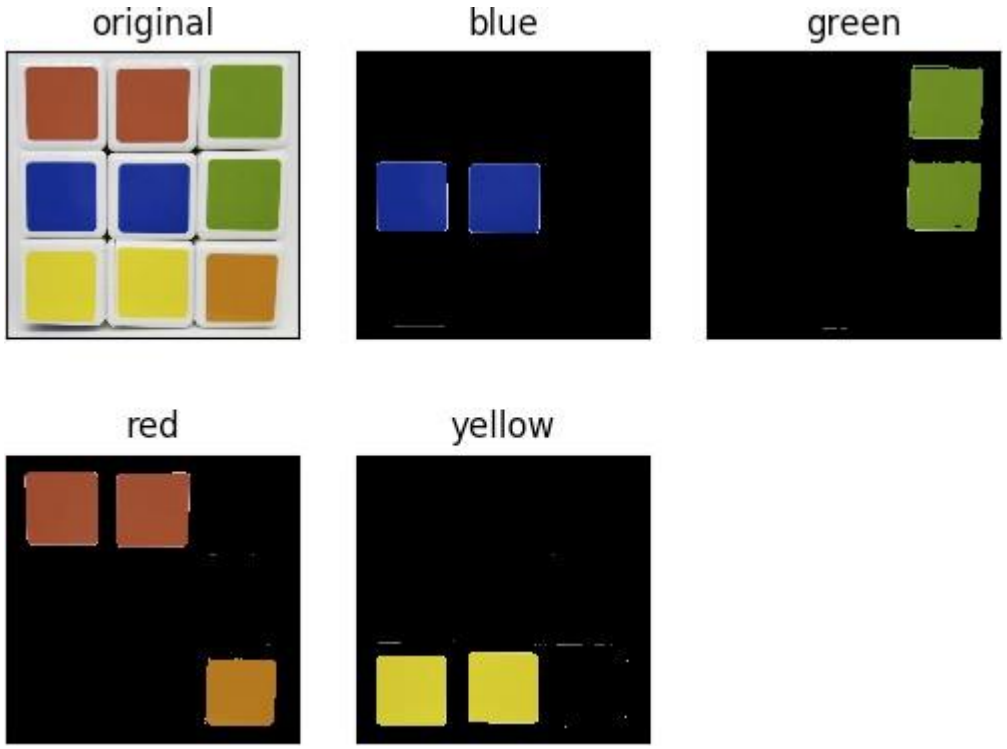
Image Arithmetic

```
# --③ 색상에 따른 마스크 생성
mask_blue = cv2.inRange(hsv, blue1, blue2)
mask_green = cv2.inRange(hsv, green1, green2)
mask_red = cv2.inRange(hsv, red1, red2)
mask_red2 = cv2.inRange(hsv, red3, red4)
mask_yellow = cv2.inRange(hsv, yellow1, yellow2)
#--④ 색상별 마스크로 색상만 추출
res_blue = cv2.bitwise_and(img, img, mask=mask_blue)
res_green = cv2.bitwise_and(img, img, mask=mask_green)
res_red1 = cv2.bitwise_and(img, img, mask=mask_red)
res_red2 = cv2.bitwise_and(img, img, mask=mask_red2)
res_red = cv2.bitwise_or(res_red1, res_red2)
res_yellow = cv2.bitwise_and(img, img, mask=mask_yellow)
#--⑤ 결과 출력
imgs = {'original': img, 'blue':res_blue, 'green':res_green,
                                        'red':res_red, 'yellow':res_yellow}
for i, (k, v) in enumerate(imgs.items()):
    plt.subplot(2,3, i+1)
    plt.title(k)
    plt.imshow(v[:,:,::-1])
    plt.xticks([]); plt.yticks([])
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

Image Arithmetic

❖ HSV Color Masking

- 색상별로 분류하기 <결과>



[그림 4-26] [예제 4-22]의 실행 결과

Image Arithmetic

❖ Chromakey Masking and Blending

- 크로마키 합성 <1/2>

```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 크로마키 배경 영상과 합성할 배경 영상 읽기
img1 = cv2.imread('../img/man_chromakey.jpg')
img2 = cv2.imread('../img/street.jpg')
#--② ROI 선택을 위한 좌표 계산
height1, width1 = img1.shape[:2]
height2, width2 = img2.shape[:2]
x = (width2 - width1)//2
y = height2 - height1
w = x + width1
h = y + height1
#--③ 크로마키 배경 영상에서 크로마키 영역을 10픽셀 정도로 지정
chromakey = img1[:10, :10, :]
offset = 20
```

[예제 4-23] 크로마 키 마스킹과 합성(chromakey.py)

Image Arithmetic

❖ Chromakey Masking and Blending

- 크로마키 합성 <1/2>

```
#--④ 크로마키 영역과 영상 전체를 HSV로 변경
hsv_chroma = cv2.cvtColor(chromakey, cv2.COLOR_BGR2HSV)
hsv_img = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
#--⑤ 크로마키 영역의 H값에서 offset 만큼 여유를 두어서 범위 지정
# offset 값은 여러차례 시도 후 결정
#chroma_h = hsv_chroma[0]
chroma_h = hsv_chroma[:,:,0]
lower = np.array([chroma_h.min()-offset, 100, 100])
upper = np.array([chroma_h.max()+offset, 255, 255])
#--⑥ 마스크 생성 및 마스킹 후 합성
mask = cv2.inRange(hsv_img, lower, upper)
mask_inv = cv2.bitwise_not(mask)
roi = img2[y:h, x:w]
fg = cv2.bitwise_and(img1, img1, mask=mask_inv)
bg = cv2.bitwise_and(roi, roi, mask=mask)
img2[y:h, x:w] = fg + bg
#--⑦ 결과 출력
cv2.imshow('chromakey', img1)
cv2.imshow('added', img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Image Arithmetic

❖ Chromakey Masking and Blending

- 크로마키 합성 <결과>

[그림 4-27] [예제 4-23]의 실행 결과

Image Arithmetic

❖ SeamlessClone

- dst = cv2.seamlessClone(src, dst, mask, coords, flags[, output])
  - src : 입력 영상, 일반적으로 전경
  - dst : 대상 영상, 일반적으로 배경
  - mask : 마스크, src에서 합성하고자 하는 영역은 255 나머지는 0
  - coodrs : src가 놓여 지기 원하는 dst의 좌표(중앙)
  - flags : 합성 방식
    - cv2.NORMAL_CLONE : 입력 원본 유지
    - cv2.MIXED_CLONE : 입력과 대상을 혼합
  - output : 합성 결과
  - dst : 합성 결과

Image Arithmetic

❖ seamless clone blending

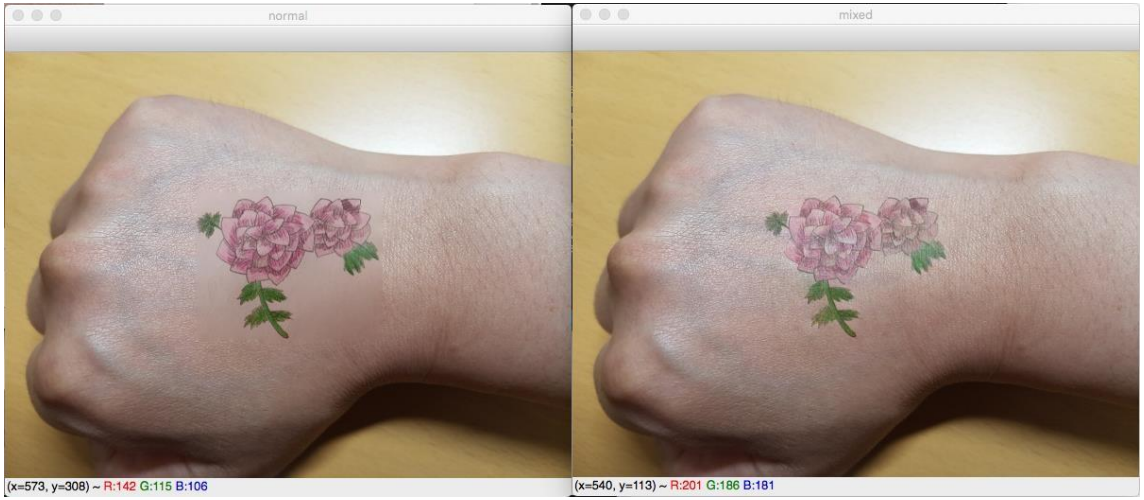
```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 합성 대상 영상 읽기
img1 = cv2.imread("../img/drawing.jpg")
img2= cv2.imread("../img/my_hand.jpg")
#--② 마스크 생성, 합성할 이미지 전체 영역을 255로 셋팅
mask = np.full_like(img1, 255)
#--③ 합성 대상 좌표 계산(img2의 중앙)
height, width = img2.shape[:2]
center = (width//2, height//2)
#--④ seamlessClone 으로 합성
normal = cv2.seamlessClone(img1, img2, mask, center, cv2.NORMAL_CLONE)
mixed = cv2.seamlessClone(img1, img2, mask, center, cv2.MIXED_CLONE)
#--⑤ 결과 출력
cv2.imshow('normal', normal)
cv2.imshow('mixed', mixed)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-24] SeamlessClone을 합성(seamlessclone.py)

Image Arithmetic

❖ **seamless clone blending**

[그림 4-28] 꽃 그림과 필자의 손



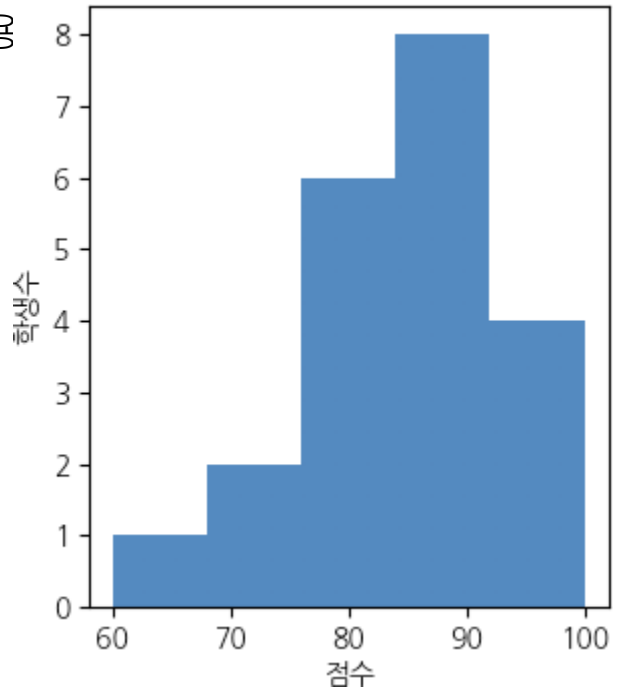
[그림 4-29] [예제 4-24]의 실행 결과

세부목차

Histogram

❖ Histogram

- Pixel들의 색상 값의 빈도를 보기 좋게 표현
  - 1Channel : Gray Color
  - 2Channel : 두 값을 교차
  - 3Channel : 표현하기 어려움
- Contrast Equalization, 배경 제거 등 다양한 활용
- OpenCV, Numpy, MatPlotlib 모두 기능 지원
  - cv2.calcHist()
  - np.histogram()
  - plt.hist()



[그림 4-30] 학생들의 점수를 표현한 히스토그램 예시

Histogram

❖ Histogram

- cv2.calcHist(img, channel, mask, histSize, ranges)
  - img : 입력 영상, [img] 처럼 리스트로 감싸서 표현
  - channel : 처리할 채널, 리스트로 감싸서 표현
    - 1채널: [0], 2채널:[0,1], 3채널:[0,1,2]
  - mask : 마스크에 지정한 픽셀만 히스토그램 계산
  - histSize : 계급(bin)의 갯수, 채널 갯수에 맞게 리스트로 표현
    - 1채널:[256], 2채널:[256, 256], 3채널:[256,256,256]
  - ranges : 각 픽셀이 갖을 수 있는 값의 범위, RGB인 경우 [0, 256]

Histogram

❖ Gray Scale 1Channel Histogram

```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 이미지 그레이 스케일로 읽기 및 출력
img = cv2.imread('../img/mountain.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow('img', img)
#--② 히스토그램 계산 및 그리기
hist = cv2.calcHist([img], [0], None, [256], [0,255])
plt.plot(hist)

print("hist.shape:", hist.shape) #--③ 히스토그램의 shape (256,1)
print("hist.sum():", hist.sum(), "img.shape:",img.shape) #--④ 히스토그램
총 합계와 이미지의 크기
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

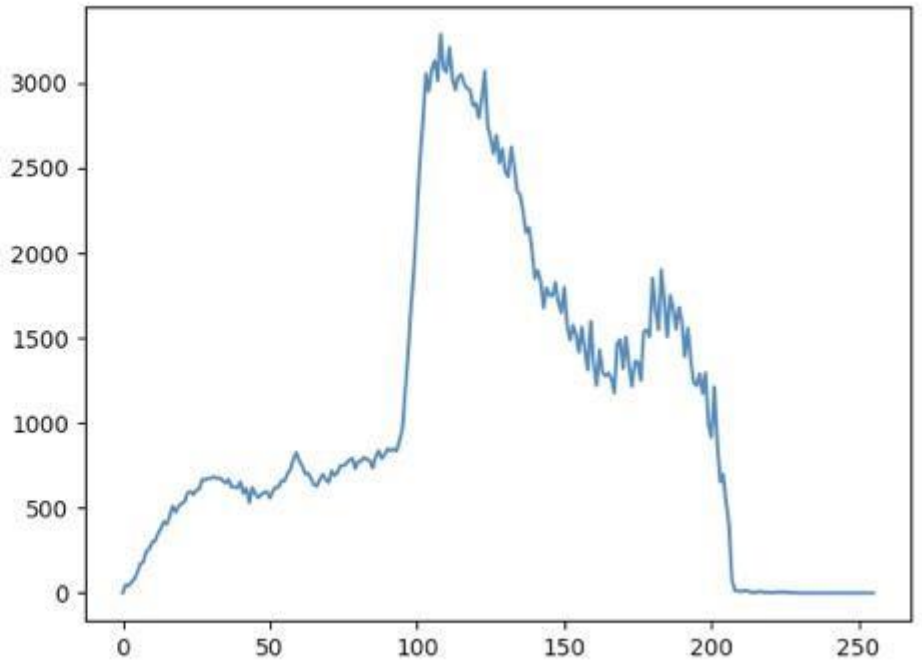
[예제 4-25] 그레이 스케일 1채널 히스토그램(histo.gray.py)

Histogram

❖ Gray Scale 1Channel Histogram

```
hist.shape: (256, 1)
hist.sum(): 270000.0 img.shape: (450, 600)
```

출력 결과



[그림 4-31] [예제 4-25]의 실행 결과

Histogram

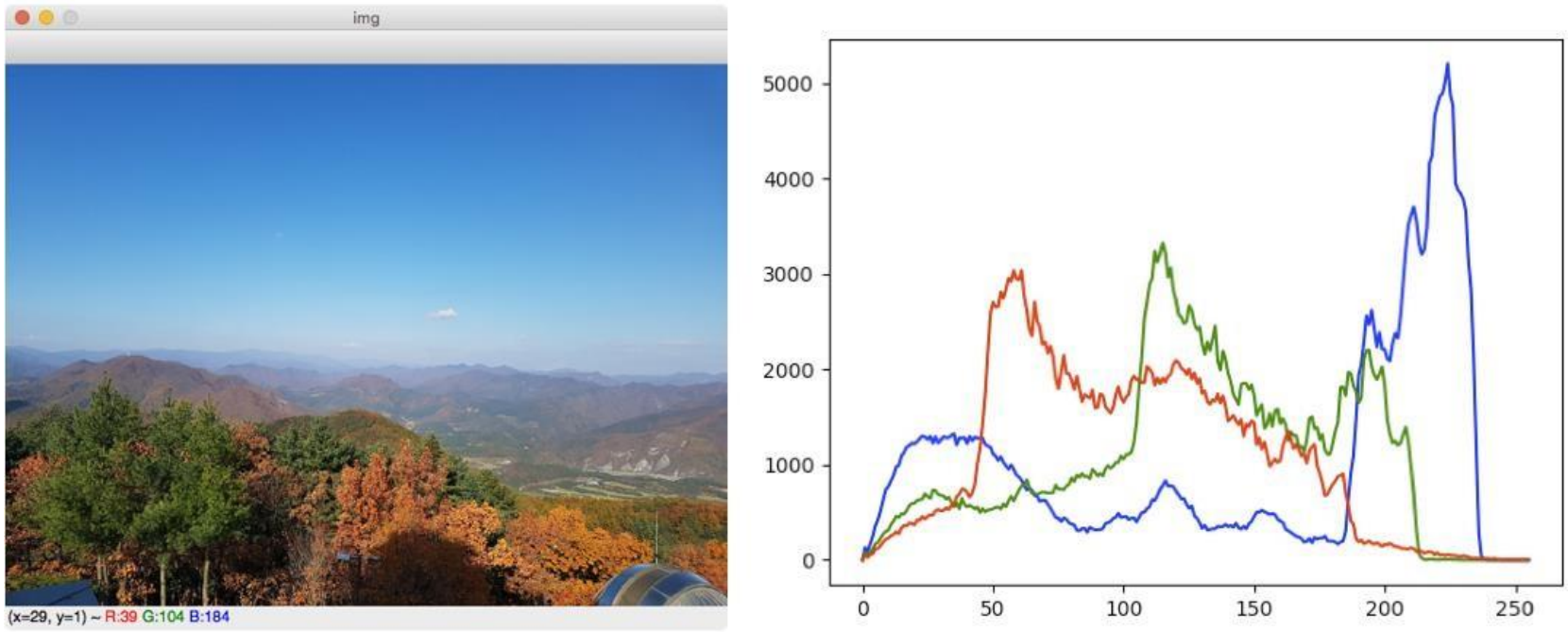
❖ Color Scale Histogram

```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 이미지 읽기 및 출력
img = cv2.imread('../img/mountain.jpg')
cv2.imshow('img', img)
#--② 히스토그램 계산 및 그리기
channels = cv2.split(img)
colors = ('b', 'g', 'r')
for (ch, color) in zip (channels, colors):
    hist = cv2.calcHist([ch], [0], None, [256], [0, 255])
    plt.plot(hist, color = color)
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-26] 컬러 히스토그램(histo_rbg.py)

Histogram

❖ Color Scale Histogram

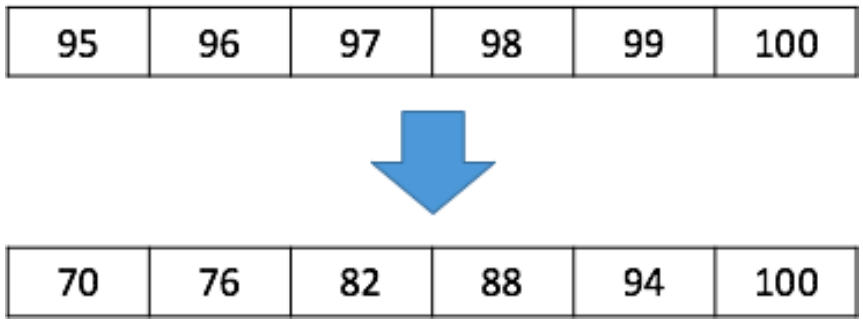


[그림 4-32] [예제 4-26]의 실행 결과

Histogram

❖ Normalize(정규화)

- 분포가 한 곳에 집중된 것을 고르게 한다.
- 픽셀의 분포가 한곳에 모여 있으면 화질이 좋지 못하다.
  - 화질 개선
    - $I_N = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin$
      - I : 정규화 이전 값
      - Min, Max : 정규화전 범위 최소 값, 최대 값
      - $newMin, newMax$: 정규화 후 최소 값, 최대 값
      - I_N: 정규화 이후 값



[그림 4-33] 정규화 예시, 95-100을 70-100로 정규화

Histogram

❖ Normalize(정규화)

- dst = cv2.normalize(src, dst, alpha, beta, type_flag)
  - src : 정규화 이전 데이타
  - dst : 정규화 이후 데이타
  - alpha : 정규화 구간 1
  - beta : 정규화 구간 2, 구간 정규화가 아닌 경우 사용 안함
  - type_flag : 정규화 알고리즘 선택 플래그 상수
    - cv2.NORM_MINMAX : alpha와 beta 구간으로 정규화
    - cv2.NORM_L1 : 전체 합으로 나누기, alpha = 정규화 전체 합
    - cv2.NORM_L2 : 단위 벡터(Unit Vector)로 정규화
    - cv2.NORM_INF : 최대 값으로 나누기

Histogram

❖ Histogram Normalize <1/2>

```
import cv2
import numpy as np
import matplotlib.pylab as plt

#--① 그레이 스케일로 영상 읽기
img = cv2.imread('../img/abnormal.jpg', cv2.IMREAD_GRAYSCALE)

#--② 직접 연산한 정규화
img_f = img.astype(np.float32)
img_norm = ((img_f - img_f.min()) * (255) / (img_f.max() - img_f.min()))
img_norm = img_norm.astype(np.uint8)

#--③ OpenCV API를 이용한 정규화
img_norm2 = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)
```

[예제 4-27] 히스토그램 정규화(hist_normalize.py)

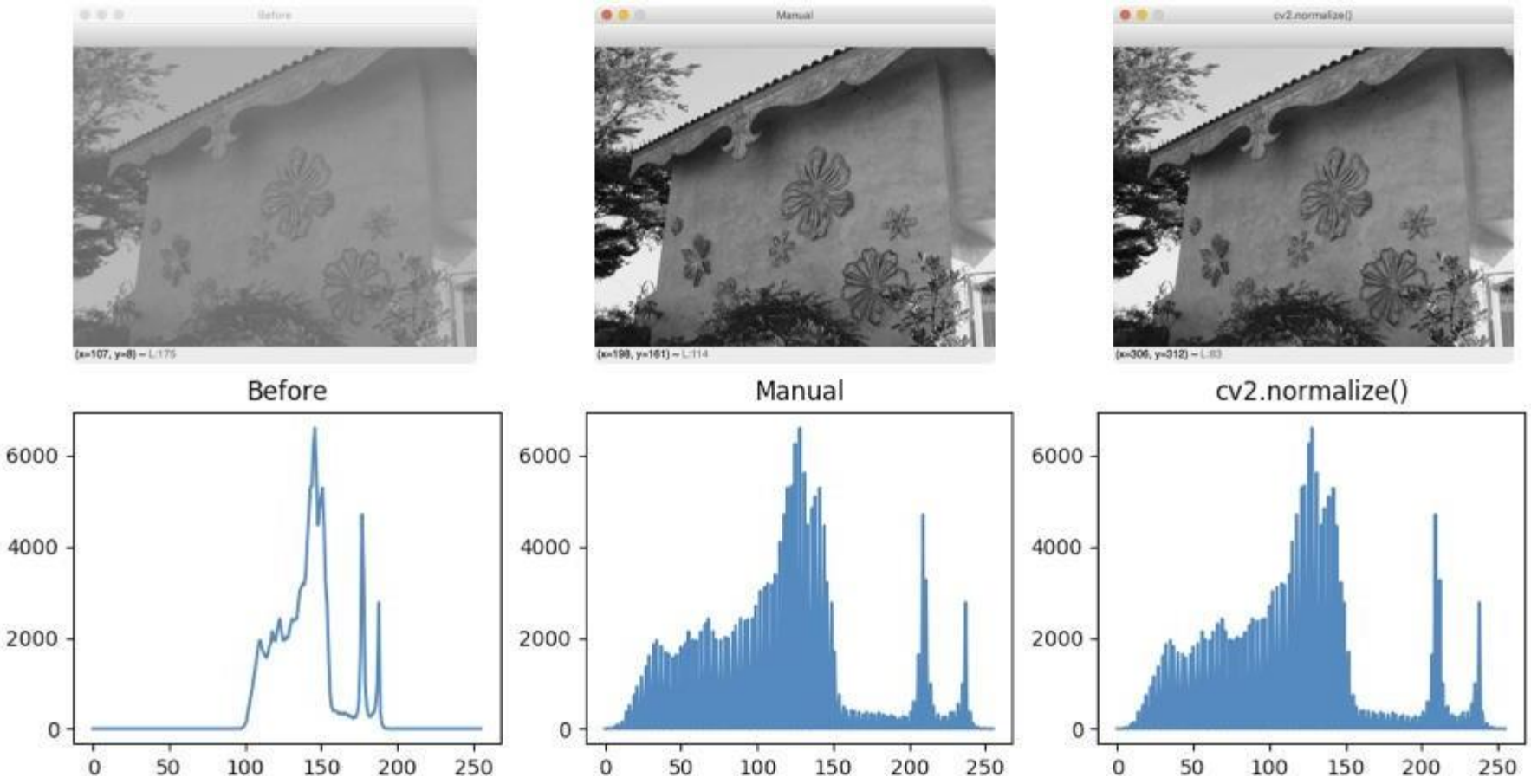
Histogram

❖ Histogram Normalize <1/2>

```
#--④ 히스토그램 계산
hist = cv2.calcHist([img], [0], None, [256], [0, 255])
hist_norm = cv2.calcHist([img_norm], [0], None, [256], [0, 255])
hist_norm2 = cv2.calcHist([img_norm2], [0], None, [256], [0, 255])

cv2.imshow('Before', img)
cv2.imshow('Manual', img_norm)
cv2.imshow('cv2.normalize()', img_norm2)

hists = {'Before' : hist, 'Manual':hist_norm, 'cv2.normalize()':hist_norm2}
for i, (k, v) in enumerate(hists.items()):
    plt.subplot(1,3,i+1)
    plt.title(k)
    plt.plot(v)
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

Histogram

❖ Histogram Normalize



[그림 4-34] [예제 4-27]의 실행 결과

Histogram

❖ Equalize

- 픽셀 분포의 폭이 아닌 높이를 제어
- 밝기 대비에 효과적
- dst = cv.equalizeHist(src[, dst])
  - src : 대상 이미지, 8비트 1채널
  - dst : 결과 이미지

- $H'(v) = round\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$
  - $cdf(v)$: 히스토그램 누적 함수
  - cdf_{min} : 누적 최소 값, 1
  - $M \times N$: 픽셀 수, 폭 x 높이
  - L : 분포 영역, 256
  - $round(v)$: 반올림
  - $H'(v)$: 이퀄라이즈된 히스토그램 값

Histogram

❖ Equalize

- 학생의 점수 이퀄라이즈 점수 계산 예시
  - Original : 70, 96, 98, 98, 100
  - Equalized : 0, 25, 80, 80, 100

점수	70	96	98	100
빈도수	1	1	2	1
누적빈도수	1	$2=1+1$	$4=2+2$	$5=4+1$
정규화빈도수	$0=\frac{1-1}{5-1}$	$0.25=\frac{2-1}{5-1}$	$0.8=\frac{4-1}{5-1}$	$1=\frac{5-1}{5-1}$
이퀄라이즈	$0=0\times100$	$25=0.25\times100$	$80=0.8\times100$	$100=1\times100$

[그림 4-35] 하생 점수 이퀄라이즈 계산 과정 예시

Histogram

❖ Equalize Gray Scale <1/2>

```
import cv2
import numpy as np
import matplotlib.pylab as plt

#--① 대상 영상으로 그레이 스케일로 읽기
img = cv2.imread('../img/yate.jpg', cv2.IMREAD_GRAYSCALE)
rows, cols = img.shape[:2]

#--② 이퀄라이즈 연산을 직접 적용
hist = cv2.calcHist([img], [0], None, [256], [0, 256]) #히스토그램 계산
cdf = hist.cumsum() # 누적 히스토그램
cdf_m = np.ma.masked_equal(cdf, 0) # 0(zero)인 값을 NaN으로 제거
cdf_m = (cdf_m - cdf_m.min()) /(rows * cols) * 255 # 이퀄라이즈 히스토그램 계산
cdf = np.ma.filled(cdf_m,0).astype('uint8') # NaN을 다시 0으로 환원
print(cdf.shape)
img2 = cdf[img] # 히스토그램을 픽셀로 맵핑
```

[예제 4-28] 그레이 스케일 이퀄라이즈 적용(histo_equalize.py)

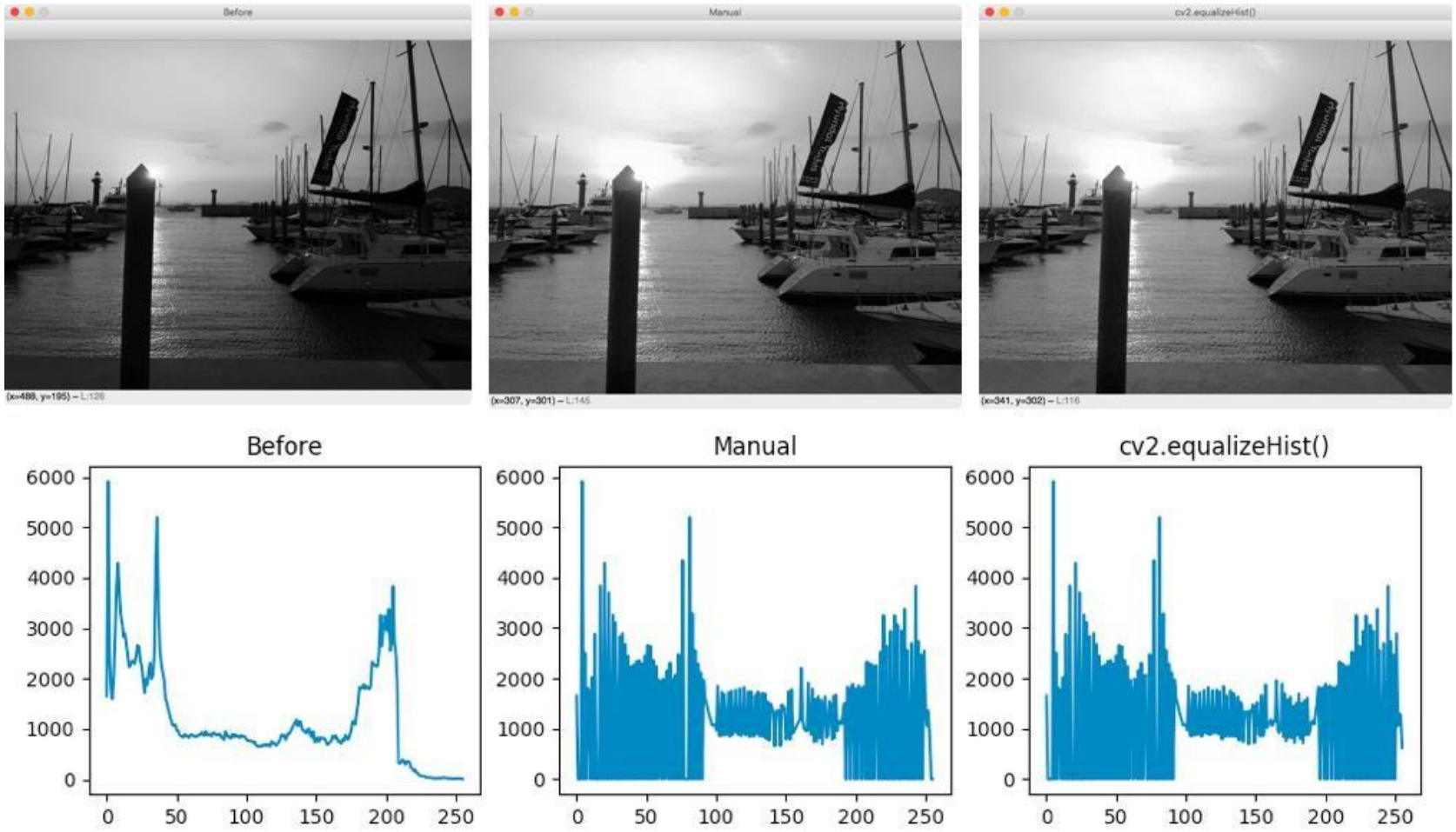
Histogram

❖ Equalize Gray Scale <1/2>

```
#--③ OpenCV API로 이퀄라이즈 히스토그램 적용
img3 = cv2.equalizeHist(img)
#--④ 이퀄라이즈 결과 히스토그램 계산
hist2 = cv2.calcHist([img2], [0], None, [256], [0, 256])
hist3 = cv2.calcHist([img3], [0], None, [256], [0, 256])
#--⑤ 결과 출력
cv2.imshow('Before', img)
cv2.imshow('Manual', img2)
cv2.imshow('cv2.equalizeHist()', img3)
hists = {'Before':hist, 'Manual':hist2, 'cv2.equalizeHist()':hist3}
for i, (k, v) in enumerate(hists.items()):
    plt.subplot(1,3,i+1)
    plt.title(k)
    plt.plot(v)
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

Histogram

❖ Equalize Gray Scale <1/2>



[그림 4-36] [예제 4-28]의 실행 결과

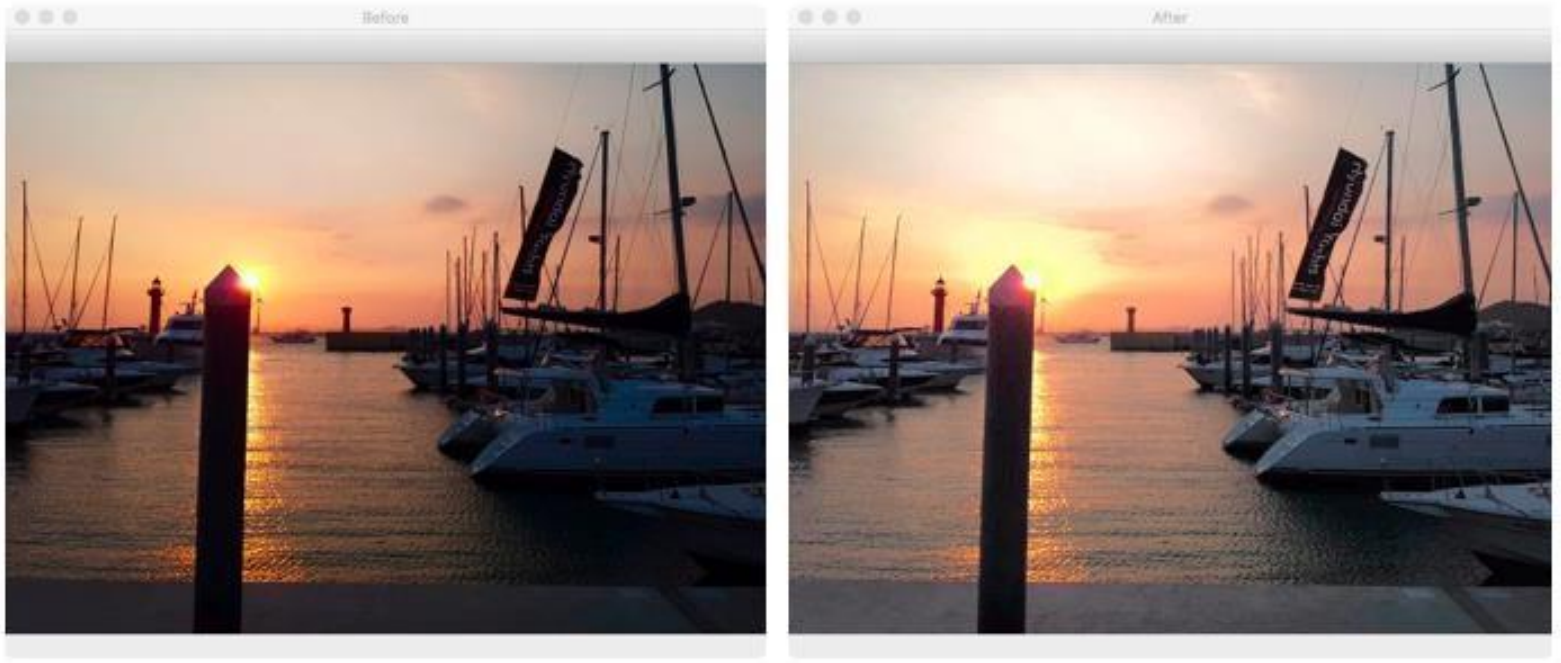
Histogram

❖ Equalize Color(YUV) Scale

```
mport numpy as np, cv2

img = cv2.imread('../img/yate.jpg') #이미지 읽기, BGR 스케일

#--① 컬러 스케일을 BGR에서 YUV로 변경
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

#--② YUV 컬러 스케일의 첫번째 채널에 대해서 이퀄라이즈 적용
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])

#--③ 컬러 스케일을 YUV에서 BGR로 변경
img2 = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
cv2.imshow('Before', img)
cv2.imshow('After', img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-2] 컬러 이미지에 대한 이퀄라이즈 적용(hist_equalize_yuv.py)

❖ Equalize Color(YUV) Scale

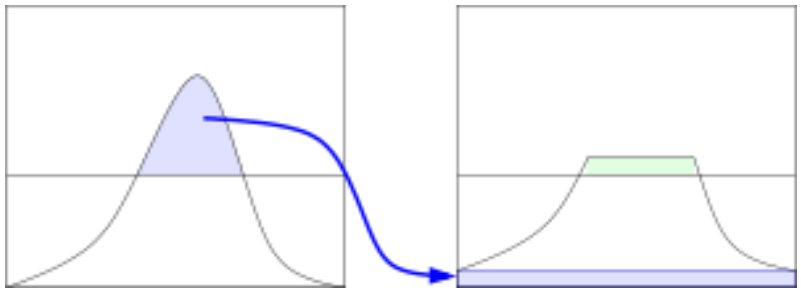


[그림 4-37] [예제 4-29]의 실행 결과

Histogram

❖ CLAHE

- Contrast Limiting Adaptive Histogram Equalization
- 이퀄라이즈 적용시 밝은 부분 최대화 방지
- 히스토그램 계급의 제한 값을 넘으면 다른 계급으로 배분
- cv2.createCLAHE(clipLimit, tileGridSize) :CLAHE 생성
  - clipLimit : Contrast 제한 경계 값, 기본 40.0
  - tileGridSize : 영역 크기, 기본 8x8
- CLAHE : CLAHE 알고리즘 객체
  - apply(src) : CLAHE 적용
    - src : 입력 영상



[그림 4-38] CLAHE 알고리즘

Histogram

❖ CLAHE로 화질 개선

```
import cv2
import numpy as np
import matplotlib.pylab as plt
#--① 이미지 읽어서 YUV 컬러스페이스로 변경
img = cv2.imread('../img/bright.jpg')
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
#--② 밝기 채널에 대해서 이퀄라이즈 적용
img_eq = img_yuv.copy()
img_eq[:,:,0] = cv2.equalizeHist(img_eq[:,:,0])
img_eq = cv2.cvtColor(img_eq, cv2.COLOR_YUV2BGR)
#--③ 밝기 채널에 대해서 CLAHE 적용
img_clahe = img_yuv.copy()
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8)) #CLAHE 생성
img_clahe[:,:,0] = clahe.apply(img_clahe[:,:,0]) #CLAHE 적용
img_clahe = cv2.cvtColor(img_clahe, cv2.COLOR_YUV2BGR)
#--④ 결과 출력
cv2.imshow('Before', img)
cv2.imshow('CLAHE', img_clahe)
cv2.imshow('equalizeHist', img_eq)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 4-30] CLAHE 적용(histo_clathe.py)

Histogram

- **CLAHE로 화질 개선**

원본 CLAHE Equalize

[그림 4-39] [예제 4-30]의 실행 결과

Histogram

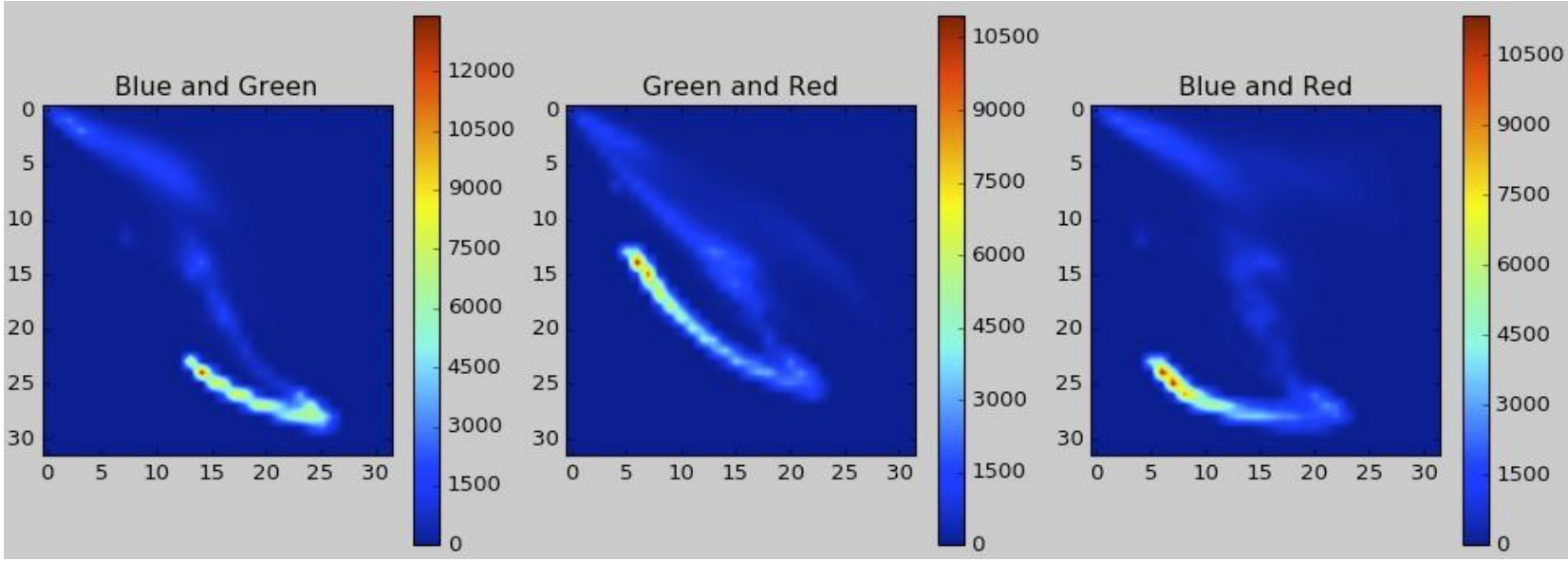
❖ 2D Histogram

```
import cv2
import matplotlib.pylab as plt
plt.style.use('classic') # --①컬러 스타일을 1.x 스타일로 사용
img = cv2.imread('../img/mountain.jpg')
plt.subplot(131)
hist = cv2.calcHist([img], [0,1], None, [32,32], [0,256,0,256]) #--②
p = plt.imshow(hist) #--③
plt.title('Blue and Green') #--④
plt.colorbar(p) #--⑤
plt.subplot(132)
hist = cv2.calcHist([img], [1,2], None, [32,32], [0,256,0,256]) #--⑥
p = plt.imshow(hist)
plt.title('Green and Red')
plt.colorbar(p)
plt.subplot(133)
hist = cv2.calcHist([img], [0,2], None, [32,32], [0,256,0,256]) #--⑦
p = plt.imshow(hist)
plt.title('Blue and Red')
plt.colorbar(p)
plt.show()
```

[예제 4-31] 2D 히스토그램(histo_2d.py)

Histogram

- **2D Histogram**



[그림 4-40] [예제 4-1]의 실행 결과

Histogram

❖ Back Projection

- HSV에서 HS에 대한 히스토그램
- ROI/전체 비율
- 선택한 영역 이외의 비율은 거의 0
- 비율로 원래의 픽셀 매핑
- 선택한 색상을 제외한 픽셀은 0
- 마스크로 사용하기 적합
- cv2.calcBackProject(img, channel, hist, ranges, scale)
  - img : 입력 영상, [img] 처럼 리스트로 감싸서 표현
  - channel : 처리할 채널, 리스트로 감싸서 표현
    - 1채널: [0], 2채널:[0,1], 3채널:[0,1,2]
  - hist : 역투영에 사용할 히스토그램
  - ranges : 각 픽셀이 갖을 수 있는 값의 범위
  - scale : 결과에 적용 배율 계수

❖ Back Projection

- HSV 역투영으로 마스킹 <1/3>

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
win_name = 'back_projection'
img = cv2.imread('../img/pump_horse.jpg')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
draw = img.copy()
#--⑤ 역투영된 결과를 마스킹해서 결과를 출력하는 공통함수
def masking(bp, win_name):
    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    cv2.filter2D(bp,-1,disc,bp)
    _, mask = cv2.threshold(bp, 1, 255, cv2.THRESH_BINARY)
    result = cv2.bitwise_and(img, img, mask=mask)
    cv2.imshow(win_name, result)
#--⑥ 직접 구현한 역투영 함수
def backProject_manual(hist_roi):
```

[예제 4-32] 마우스로 선택한 영역의 물체 배경 제거(histo_backproject.py)

Histogram

❖ Back Projection

- HSV 역투영으로 마스킹 <2/3>

```
        #--⑦ 전체 영상에 대한 H,S 히스토그램 계산
        hist_img = cv2.calcHist([hsv_img], [0,1], None,[180,256], [0,180,0,256])
        #--⑧ 선택영역과 전체 영상에 대한 히스토그램 그램 비율계산
        hist_rate = hist_roi/ (hist_img + 1)
        #--⑨ 비율에 맞는 픽셀 값 매핑
        h,s,v = cv2.split(hsv_img)
        bp = hist_rate[h.ravel(), s.ravel()]
        bp = np.minimum(bp, 1)
        bp = bp.reshape(hsv_img.shape[:2])
        cv2.normalize(bp,bp, 0, 255, cv2.NORM_MINMAX)
        bp = bp.astype(np.uint8)
        #--⑩ 역 투영 결과로 마스킹해서 결과 출력
        masking(bp,'result_manual')
    # OpenCV API로 구현한 함수 ---⑪
    def backProject_cv(hist_roi):
        # 역투영 함수 호출 ---⑫
         p = cv2.calcBackProject([hsv_img], [0, 1], hist_roi, [0, 180, 0, 256], 1)
```

Histogram

❖ Back Projection

- HSV 역투영으로 마스킹 <3/3>

```
    # 역 투영 결과로 마스킹해서 결과 출력 ---⑬
    masking(bp,'result_cv')
    # ROI 선택 ---①
    (x,y,w,h) = cv2.selectROI(win_name, img, False)
    if w > 0 and h > 0:
        roi = draw[y:y+h, x:x+w]
        cv2.rectangle(draw, (x, y), (x+w, y+h), (0,0,255), 2)
        #--② 선택한 ROI를 HSV 컬러 스페이스로 변경
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
        #--③ H,S 채널에 대한 히스토그램 계산
        hist_roi = cv2.calcHist([hsv_roi],[0, 1], None, [180, 256], [0, 180, 0, 256] )
        #--④ ROI의 히스토그램을 매뉴얼 구현함수와 OpenCV 이용하는 함수에 각각 전
    달
        backProject_manual(hist_roi)
        backProject_cv(hist_roi)
    cv2.imshow(win_name, draw)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Histogram

❖ Back Projection

- HSV 역투영으로 마스킹

[그림 4-41] [예제 4-32]의 실행 결과

Histogram

❖ Compare Histogram

- 히스토그램을 비교해서 영상간의 비슷한 정도 파악
- cv2.compareHist(hist1, hist2, method)
  - hist1, hist2 : 비교할 2개의 히스토그램, 크기와 차원이 같아야 한다.
  - method : 비교 알고리즘 선택 플래그 상수
    - cv2.HISTCMP_CORREL : 상관관계
      - 1 : 완전 일치, -1 : 최대 불일치, 0: 무관계
    - cv2.HISTCMP_CHISQR : 카이제곱
      - 0 : 완전 일치, 큰 값(미정) : 최대 불일치
    - cv2.HISTCMP_INTERSECT : 교차
      - 1 : 완전 일치, 0 : 최대 불일치(1로 정규화 한 경우)
    - cv2.HISTCMP_BHATTACHARYYA : 바타차야
      - 0: 완전 일치, 1 : 최대 불일치
    - cv2.HISTCMP_HELLINGER : HISTCMP_BHATTACHARYYA와 동일

Histogram

❖ Compare Histogram <1/2>

```
import cv2, numpy as np
import matplotlib.pylab as plt
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/taekwonv2.jpg')
img3 = cv2.imread('../img/taekwonv3.jpg')
img4 = cv2.imread('../img/dr_ochanomizu.jpg')
cv2.imshow('query', img1)
imgs = [img1, img2, img3, img4]
hists = []
for i, img in enumerate(imgs) :
    plt.subplot(1,len(imgs),i+1)
    plt.title('img%d'% (i+1))
    plt.axis('off')
    plt.imshow(img[:,:,::-1])
    #---① 각 이미지를 HSV로 변환
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #---② H,S 채널에 대한 히스토그램 계산
    hist = cv2.calcHist([hsv], [0,1], None, [180,256], [0,180,0, 256])
```

[예제 4-2] 관심영역 복제 및 새 창 띄우기(roi_copy .py)

Histogram

❖ Compare Histogram <2/2>

```
        #---③ 0~1로 정규화
        cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
        hists.append(hist)
query = hists[0]
methods = {'CORREL' :cv2.HISTCMP_CORREL, 'CHISQR':cv2.HISTCMP_CHISQR,
                  'INTERSECT':cv2.HISTCMP_INTERSECT,
                  'BHATTACHARYYA':cv2.HISTCMP_BHATTACHARYYA}
for j, (name, flag) in enumerate(methods.items()):
    print('%-10s'%name, end='\t')
    for i, (hist, img) in enumerate(zip(hists, imgs)):
        #---④ 각 메서드에 따라 img1과 각 이미지의 히스토그램 비교
        ret = cv2.compareHist(query, hist, flag)
        if flag == cv2.HISTCMP_INTERSECT: #교차 분석인 경우
            ret = ret/np.sum(query) #비교대상으로 나누어 1로 정규화
        print("img%d:%7.2f"% (i+1 , ret), end='\t')
    print()
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```

❖ Compare Histogram

[그림 4-42] [예제 4-33]의 실행 결과

```
CORREL          img1:   1.00    img2:   0.70    img3:   0.56    img4:   0.23
CHISQR          img1:   0.00    img2:  67.33    img3:  35.71    img4:1129.49
INTERSECT       img1:   1.00    img2:   0.54    img3:   0.40    img4:   0.18
BHATTACHARYYA   img1:   0.00    img2:   0.48    img3:   0.47    img4:   0.79
```

출력 결과

세부목차

1. ROI(Region Of Interest)
2. Color Space
3. Threshold
4. Image Arithmetic
5. Histogram
6. **Workshop**

Workshop

❖ Monster Face Synthesis

- 두 얼굴의 절반씩 하나의 얼굴로 자연스럽게 나타나게 합성하세요.
- 사용 파일
  - img/man_face.jpg
  - img/skull.jpg
- 결과 예시

[그림 4-43] 반해골 괴물 얼굴 합성

- 힌트
  - 두 얼굴이 접하는 부분을 0%~50%, 50%~0% Alpha blending

❖ Motion Detecting CCTV

- 움직임을 검출하는 카메라를 만들어 보세요
  .
- 결과 예시

[그림 4-44] 모션 감지 CCTV

Workshop

❖ Motion Detecting CCTV

- 카메라로 움직이는 물체나 사람을 인식해서 그 영역을 표시하세요.
- 움직임을 감지한 경우 빨강색으로 "Motion Detected"라는 메시지를 화면에 출력하세요.
- 카메라 영상에 움직이는 영역을 표시한 영상과 움직이는 픽셀만 표시한 화면을 좌우로 배치해서 하나의 화면으로 출력하세요.
- 힌트
  - 차영상을 구해서 차이가 있는 부분을 찾으세요.
  - 이전/현재 두 영상의 차이로는 해결 할 수 없습니다.
  - A와 B의 차이와 B와 C의 차이 모두 있는 경우 움직임으로 판단
  - 설정 가능 기준치 2가지
    - 차이나는 픽셀 값의 크기
    - 차이나는 픽셀 값의 갯수