색상 변경 / 이미지

이미지 출력 및 저장

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

img = cv2.imread("barbara.jpg") # 이미지 불러오기

plt.grid(None)
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(img) # 이미지 출력

cv2.imwrite('out_img.jpg', img) # 이미지 저장
```

이미지 색상 변경

사전 정의된 함수 사용

```
# BGR to RGB
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.grid(None)
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(rgb)

# Gray
gray_img =cv2.imread('barbara.jpg', 0)

plt.grid(None)
plt.xticks([])
plt.yticks([])
grayplot = plt.imshow(gray_img) # 흑백 아니고 viridis로 출력됨

plt.grid(None)
plt.xticks([])
plt.yticks([])
grayplot = plt.imshow(gary_img, cmap = plt.cm.gray)
```

Raw Level로 변경

```
height, width, channels = img.shape
print('세로:', height, "가로:", width, "채널:", channels)

# Convert BGR to RGB
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
R_channel = RGB_img.copy()

R_channel[:,:,1] = 0 # G 채널 값을 모두 0으로
R_channel[:,:,2] = 0 # B 채널 값을 모두 0으로
```

```
plt.grid(None)
plt.xticks([])
plt.yticks([])
plt.imshow(R_channel)
```

이미지를 4분할로 각각 채널 다르게 만들기

```
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
Mix_channel = RGB_img.copy()

# R 채널만 살림 (우상단)
Mix_channel[0:height//2, width//2:, 1] = 0  #
Mix_channel[0:height//2, width//2:, 2] = 0

# G 채널만 살림 (우하단)
Mix_channel[height//2:, width//2:, 0] = 0
Mix_channel[height//2:, width//2:, 2] = 0

# B 채널만 살림 (좌하단)
Mix_channel[height//2:, :width//2, 0] = 0
Mix_channel[height//2:, :width//2, 1] = 0

plt.grid(None)
plt.xticks([])
plt.yticks([])

plt.imshow(Mix_channel)
plt.show()
```

```
for j in range(height):
    for i in range(width):
        # R 채널
        if j < height//2 and i > width//2:
            Mix_channel[j,i,1] = 0
            Mix_channel[j,i,2] = 0
        # G 채널
        if j > height//2 and i > width//2:
            Mix_channel[j,i,0] = 0
            Mix_channel[j,i,2] = 0
        # B 채널
        if j > height//2 and i < width//2:
            Mix_channel[j,i,0] = 0
            Mix_channel[j,i,1] = 0

plt.grid(None)
plt.xticks([])
plt.yticks([])

plt.imshow(Mix_channel)
plt.show()
```

Histogram

명암 영상에서 히스토그램 계산

- 입력 : 명암 영상 $f(j,i), 0 \leq j \leq M-1, 0 \leq j \leq N-1$
- 출력 : 히스토그램 $h(l)$과 정규 히스토그램 $\hat{h}(l), 0 \leq l \leq L-1$
- Pseudocode

```
for (l = 0 to L-1)
    h(l) = 0;           // 초기화
for (j = 0 to M-1)
    for (i = 0 to N-1)  // f의 화소(j,i) 각각에 대해
        h(f(j,i))++;    // 그곳 명암값에 해당하는 히스토그램 칸을 1만큼 증가

for (l = 0 to L-1)
    (hat)h(l) = h(l) / (M * N) // 정규화
```

- Python

```
import numpy as np
import matplotlib.pyplot as plt

def histogram(img):
    bins = np.zeros(256, np.int32) // 초기화

    height, width = img.shape // M, N값 받아옴

    for i in range(height):
        for j in range(width):
            bins[img[i][j]] += 1 // 그 곳 명암값에 해당하는 히스토그램 칸 1 증가

    bins = bins/(width*height)  // 히스토그램 정규화

    return bins

# 그레이 스케일로 read
img = cv2.imread('barbara.jpg',0)

bins = histogram(img)

plt.plot(bins)
plt.xlim([0,256])
plt.show()
```

- Python - Opencv calcHist 함수 활용

```
import cv2
# images(uint8 또는 float32 타입의 이미지 - 대괄호안에), channels(채널 인덱스),
#   mask(전체 이미지일시 None), histSize(계산할 막대 개수), ranges(계산 범위)
# 채널인덱스 : 그레이스케일 => 0, RGB : 0(R), 1(G), 2(B)
hist = cv2.calcHist([img],[0],None,[256],[0,256]) # 그냥 히스토그램

height, width = img.shape
hist = hist / (width*height) # 정규화
```

칼라 이미지 히스토그램

```
import cv2
color_img = cv2.imread('barbara.jpg')
# cvtColor : BGR 이미지를 RGB로 바꿔줌
RGB_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)

# 각 채널별로 histogram 추출
R_bins = histogram(RGB_img[:,:,0])
G_bins = histogram(RGB_img[:,:,1])
B_bins = histogram(RGB_img[:,:,2])
```

Histogram - 10 bins

```
# 10레벨 영상으로 표현하기
def histogram_10bins(img):
    bins = np.zeros(10, np.int32)

    height, width = img.shape

    for i in range(0, height):
        for j in range(0, width):
            bins[int(img[i][j]/26)] += 1    # int() 함수 써줘야함

    return bins

color_img = cv2.imread('barbara.jpg')
RGB_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)

R_bins = histogram_10bins(RGB_img[:,:,0])
G_bins = histogram_10bins(RGB_img[:,:,1])
B_bins = histogram_10bins(RGB_img[:,:,2])

# 0 ~ len(R_bins) 값을 1 간격을 두고 각각 저장
x = np.arange(len(R_bins))
width = 0.2
# 시작위치, 데이터, 차지하는 width, 색깔
plt.bar(x - width, R_bins, width, color='r')
plt.bar(x, G_bins, width, color='g')
plt.bar(x + width, G_bins, width, color='b')

plt.show()
```

히스토그램 평활화

$$l_{out} = T(l_{in}) = round(c(l_{in}) * (L-1))$$

- 이 때, $c(l_{in}) = \sum_{l=0}^{l_{in}} \hat{h}(l)$

```
import copy
import cv2
import numpy as np

def equalizeHist(img):
    new_img = copy.deepcopy(img)
    height, width = new_img.shape
```

```
    hist = cv2.calcHist([img],[0],None,[256],[0,256]) # 히스토그램
    hist = hist/(height*width)  # 정규화

    # 누적 히스토그램 c
    c = np.cumsum(hist)

    for i in range(height):
        for j in range(width):
            new_img[i][j] = round(c[new_img[i][j]] * (255))

    return new_img
```

- opencv 활용

```
img = cv2.imread('barbara.jpg',0)
# cv2.equalizeHist(이미지)
equalized_img = cv2.equalizeHist(img)

plt.imshow(equalized_img, cmap = 'gray')

plt.show()
```

히스토그램 Stretching

```
img = cv2.imread('barbara.jpg', 0)
hist = cv2.calcHist([img],[0],None,[256],[0,256])

stretched_img = (img - hist.min())/(hist.max()-hist.min())*255
```

임계값 설정(Otsu)

```
img = cv2.imread('barbara.jpg', 0)

# histogram and CDF
hist = cv2.calcHist([img], [0], None, [256], [0,256])
hist_norm = hist.ravel() / hist.max()
CDF = hist_norm.cumsum()

# initialization
bins = np.arange(256) # 0 ~ 255
fn_min = np.inf
thresh = -1

# Otsu algorithm operation
for i in range(1, 256):
    # hsplit : 인덱스 i를 기준으로 배열을 분할
    p1, p2 = np.hsplit(hist_norm, [i]) # probabilities
    q1, q2 = CDF[i], CDF[255]-CDF[i] # cum sum of classes

    if q1 == 0:
        q1 = 0.00000001
    if q2 == 0:
        q2 = 0.00000001
    b1, b2 = np.hsplit(bins, [i]) # weights
```

```
    # finding means and variances
    m1, m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
    v1, v2 = np.sum(((b1-m1)**2)*p1)/q1, np.sum(((b2-m2)**2)*p2)/q2

    # calculates the minimization function
    fn = v1 * q1 + v2 * q2
    if fn < fn_min:
        fn_min = fn
        thresh = i

# find otsu's threshold value with OpenCV function
# src(이미지 소스), thresh(임계값), maxval(임계값 넘었을 때 적용할 value), type
# type : cv2.THRESH_BINARY, THRESH_BINARY_INV, THRESH_TRUNC, THRESH_TOZERO
#           THRESH_TOZERO_INV, OTSU를 적용하기 위해 THRES_BINARY + THRESH_OTSU
ret, otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print("thresh: {} ret : {}".format(thresh,ret))

binary_img = np.zeros((img.shape[1], img.shape[0]), np.uint8)
for i in range(0, img.shape[1]):
    for j in range(0, img.shape[0]):
        if img[i][j] < thresh:
            binary_img[i][j] = 0
        else:
            binary_img[i][j] = 255

plt.figure(figsize = (14,7))

plt.subplot(121)
plt.imshow(otsu, cmap = 'gray')

plt.subplot(122)
plt.imshow(binary_img, cmap = 'gray')

plt.show()
```

연결성 / 라벨링

4연결성

```
def four_connect(img) :
    height, width = img.shape

    # 라벨값을 저장할 마스크
    mask = [[0 for x in range(width)] for y in range(height)]

    # 그룹에 부여할 라벨
    current_label = 1

    # 전체 이미지 배열을 돌며 checkFour 수행
    for i in range(0, height):
        for j in range(0, width):
            # 라벨링이 됐으면 current_label값을 다음으로 갱신
            if (checkFour(i, j, img, mask, 0, current_label)):
                current_label += 1
```

```
    # 랜덤 색깔 부여, 이미 색깔이 부여된 label은 Dictionary에 기억해놓은 값을 사용
    color_dict = {}
    labeled_img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    for i in range(height):
        for j in range(width):
            label = mask[i][j]
            if label != 0:
                # 해당 라벨이 dictionary에 없으면 새로 생성
                if label not in color_dict:
                    r = random.randint(0,255)
                    g = random.randint(0,255)
                    b = random.randint(0,255)
                    color_tuple = (r,g,b)
                    color_dict[label] = color_tuple
                # dictionary 참조 색깔 부여
                for k in range(3):
                    labeled_img[i][j][k] = color_dict[label][k]

    return labeled_img

# 재귀적으로 상하좌우를 탐색하는 함수
## mode값 0 : 상하좌우 다 체크 / 1 : 하 제외 / 2 : 우 제외 / 3 : 상 제외 / 4 : 좌 제외
def checkFour(i, j, img, mask, mode, label):
    # 해당 픽셀의 값이 255이고, 라벨링 돼있지 않을 때 진입
    if (img[i][j] == 255 and mask[i][j] == 0) :
        mask[i][j] = label
        # 아래쪽 체크, 현재 함수를 아래쪽에서 호출했거나 끝 쪽인 경우 진입 X
        if (mode != 1 and i != len(img)-1):
            checkFour(i+1,j, img, mask, 3, label)
        # 오른쪽 체크, 현재 함수를 오른쪽에서 호출했거나 끝 쪽인 경우 진입 X
        if (mode != 2 and j != (len(img[0])-1)):
            checkFour(i,j+1, img, mask, 4, label)
        # 위쪽 체크, 현재 함수를 위쪽에서 호출했거나 끝 쪽인 경우 진입 X
        if (mode != 3 and i != 0):
            checkFour(i-1, j, img, mask, 1, label)
        # 왼쪽 체크, 현재 함수를 왼쪽에서 호출했거나 끝 쪽인 경우 진입 X
        if (mode != 4 and j != 0):
            checkFour(i,j-1, img, mask, 2, label)
        return True

    # 라벨링이 되지 않았으면 False 리턴
    return False

img = cv2.imread('sample.png', 0)

labeled_img = four_connect(img)

## image 출력
plt.imshow(labeled_img)
plt.grid(None)
plt.xticks([])
plt.yticks([])
plt.show()
```

8 연결성

```
def eight_connect(img) :
```

```
    height, width = img.shape

    # 라벨값을 저장할 마스크
    mask = [[0 for x in range(width)] for y in range(height)]

    # 그룹에 부여할 라벨
    current_label = 1

    # 전체 이미지 배열을 돌며 checkFour 수행
    for i in range(0, height):
        for j in range(0, width):
            # 라벨링이 됐으면 current_label값을 다음으로 갱신
            if (checkEight(i, j, img, mask, 0, current_label)):
                # print ("labeled", current_label)
                current_label += 1


    # 랜덤 색깔 부여, 이미 색깔이 부여된 label은 Dictionary에 기억해놓은 값을 사용
    color_dict = {}
    labeled_img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    for i in range(height):
        for j in range(width):
            label = mask[i][j]
            if label != 0:
                # 해당 라벨이 dictionary에 없으면 새로 생성
                if label not in color_dict:
                    r = random.randint(0,255)
                    g = random.randint(0,255)
                    b = random.randint(0,255)
                    color_tuple = (r,g,b)
                    color_dict[label] = color_tuple
                # dictionary 참조 색깔 부여
                for k in range(3):
                    labeled_img[i][j][k] = color_dict[label][k]

    return labeled_img

# 재귀적으로 8 연결성을 탐색하는 함수
def checkEight(i, j, img, mask, mode, label):
    # 해당 픽셀의 값이 255이고, 라벨링 돼있지 않을 때 진입
    if (img[i][j] == 255 and mask[i][j] == 0) :
        mask[i][j] = label
        if (mode != 1 and i != len(img)-1):
            checkEight(i+1,j, img, mask, 3, label)
        if (mode != 2 and j != (len(img[0])-1)):
            checkEight(i,j+1, img, mask, 4, label)
        if (mode != 3 and i != 0):
            checkEight(i-1, j, img, mask, 1, label)
        if (mode != 4 and j != 0):
            checkEight(i,j-1, img, mask, 2, label)
        if (mode != 5 and i != len(img)-1 and j != len(img[0])-1):
            checkEight(i+1, j+1, img, mask, 7, label)
        if (mode != 6 and i != len(img)-1 and j != 0):
            checkEight(i+1, j-1, img, mask, 8, label)
        if (mode != 7 and i != 0 and j != 0):
            checkEight(i-1, j-1, img, mask, 5, label)
        if (mode != 8 and i != 0 and j != len(img[0])-1):
            checkEight(i-1, j+1, img, mask, 6, label)
```

```
            return True

        # 라벨링이 되지 않았으면 False 리턴
        return False

img = cv2.imread('sample.png', 0)

labeled_img = eight_connect(img)

# image 출력
plt.imshow(labeled_img)
plt.grid(None)
plt.xticks([])
plt.yticks([])
plt.show()
```