



Day66; 20221208

| | |
|------|---------------|
| 📅 날짜 | @2022년 12월 8일 |
| 🕒 유형 | @2022년 12월 8일 |
| ☰ 태그 | |

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/ai>

u8yes/AI



1 Contributor 0 Issues 0 Stars 0 Forks

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e429d3d4-ace8-44fb-9357-c5cf7b68ee64/04_%ED%99%94%EC%86%8C_%EC%B2%98%EB%A6%AC.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/83cf3bed-5c2d-4ea8-9910-bfda7327a290/05_%EC%98%81%EC%97%AD_%EC%B2%98%EB%A6%AC.pdf

(빈)도수 분포표 - 히스토그램- 범주형

나의 강화 · 나의 소식 1 · 나의 말 일 0

1. 벡터

▶ 선택대수학을 위한 벡터란? 업데이트: 2022.12.08 | 41

▶ 실좌표공간 업데이트: 2022.11.30 | 26

▶ 대수와 그래프를 이용한 벡터의 개념 업데이트: 2022.12.05 | 26

▶ 벡터와 스칼라의 곱셈 업데이트: 2022.11.30 | 18

▶ 벡터 예제 업데이트: 2022.11.30 | 24

▶ 단위벡터란? 업데이트: 2022.11.30 | 20

[칸아카데미] 모두를 위한 선형대수학
[칸아카데미] 모두를 위한 선형대수학

👉 <https://www.boostcourse.org/ai151/joinLectures/194187?isDesc=false>

boostcourse

나의 강좌 | 나의 소식 1 | 나의 할 일 0 | |

all 입문
[하버드] 확률론 기초: Statistics 110

Joe Blitzstein

좋아요 803 | 수강생 11658

공지게시판
수강생 토론게시판
오리엔테이션
학률론 기초
코스를 마치며
성적조회

오리엔테이션

01. [하버드] 확률론 기초 : Statistics 110 코스란?
업데이트: 2022.12.01 | 119

02. 이 코스를 개발한 전문가
업데이트: 2022.11.30 | 76

03. 부스트코스 강의 학습 가이드
업데이트: 2022.11.30 | 72

04. 시작합니다!
업데이트: 2022.11.30 | 61

[하버드] 확률론 기초: Statistics 110

하버드] 확률론 기초: Statistics 110



<https://www.boostcourse.org/ai152/joinLectures/195039>

MinMaxScale과 형태가 비슷함.

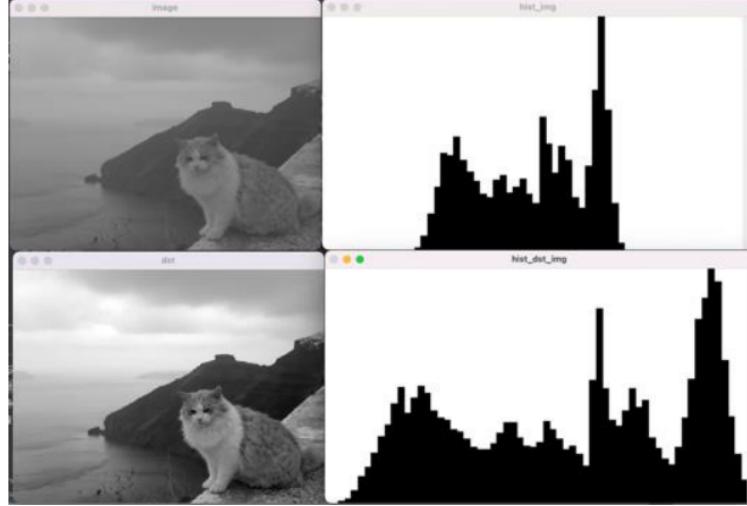
$$\text{새 픽셀 값} = ((\text{픽셀 값} - \text{low}) / (\text{high} - \text{low})) * 255$$

히스토그램

❖ Histogram Stretching

- ✓ 히스토그램을 스트레칭하고자 하는 경우 빈도 값이 존재하는 가장 낮은 픽셀 값과 가장 높은 빈도 값이 존재하는 픽셀 값을 알아야 함
- ✓ 가장 낮은 픽셀 값은 0으로 당기고 가장 높은 픽셀 값을 255로 당긴 후 중간의 픽셀 값들은 각각의 비율에 따서 픽셀 값의 위치를 조정

$$\text{새 픽셀 값} = ((\text{픽셀 값} - \text{low}) / (\text{high} - \text{low})) * 255$$



09_histogram_stretching.py

```

import numpy as np, cv2
from Common.histogram import draw_histo

def search_value_idx(hist, bias = 0): # bias라는 매개변수 선언(디폴트는 0으로)
    for i in range(hist.shape[0]):
        idx = np.abs(bias - i)
        # 움직이면서 한칸씩 상승(bias가 0) or 하강(bias가 1)하면서 최소, 최대값 찾아냄
        if hist[idx] > 0:
            return idx
    return -1

image = cv2.imread("dst.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류")

bsize, ranges = [64], [0, 256] # size를 64가지, 256/64하면 '4'가 나옴.
hist = cv2.calcHist([image], [0], None, bsize, ranges)
# channel(BGR)값이 흑백일 때 0번 채널은 BLUE

bin_width = ranges[1] / bsize[0]
high = search_value_idx(hist, bsize[0] - 1) * bin_width # 최대값
low = search_value_idx(hist, 0) * bin_width # 최소값

idx = np.arange(0,256)
idx = (idx - low) * 256 / (high - low) # 정규화의 기본 개념에서 255로 펼쳐줌
idx[0:int(low)] = 0
idx[int(high+1):] = 255

```

look up

1. (사전·참고 자료·컴퓨터 등에서 정보를) 찾아보다

```

dst = cv2.LUT(image, idx.astype('uint8')) # LookUp Table 사용
# 최소, 최대값을 찾아서 0~255사이로 펼쳐주게끔 테이블로 만들어줌.
# dst - destination
# 현재는 dst가 스트레칭 되서 짜악 퍼져있는 상태가 돼있음.
hist_dst = cv2.calcHist([dst],[0], None, bsize, ranges)

hist_img = draw_hist(hist, (200,360))
hist_dst_img = draw_hist(hist_dst, (200, 360))

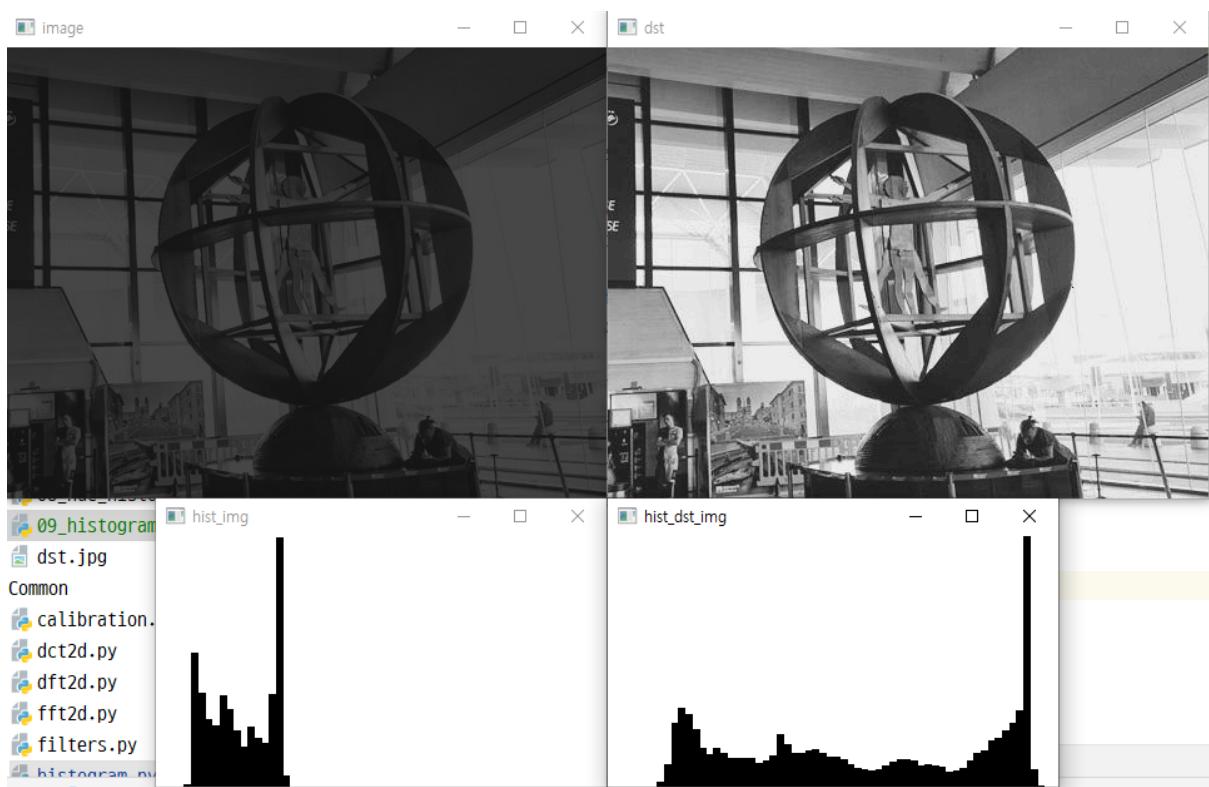
print("high_value=", high)
print("low_value=", low)

cv2.imshow("image", image)
cv2.imshow("hist_img", hist_img)

cv2.imshow("dst", dst)
cv2.imshow("hist_dst_img", hist_dst_img)

cv2.waitKey(0)

```



```
C:\ProgramData\Anacond  
high_value= 76.0  
low_value= 12.0
```

최소값이 낮게 나온다.(어둡게 나온다)

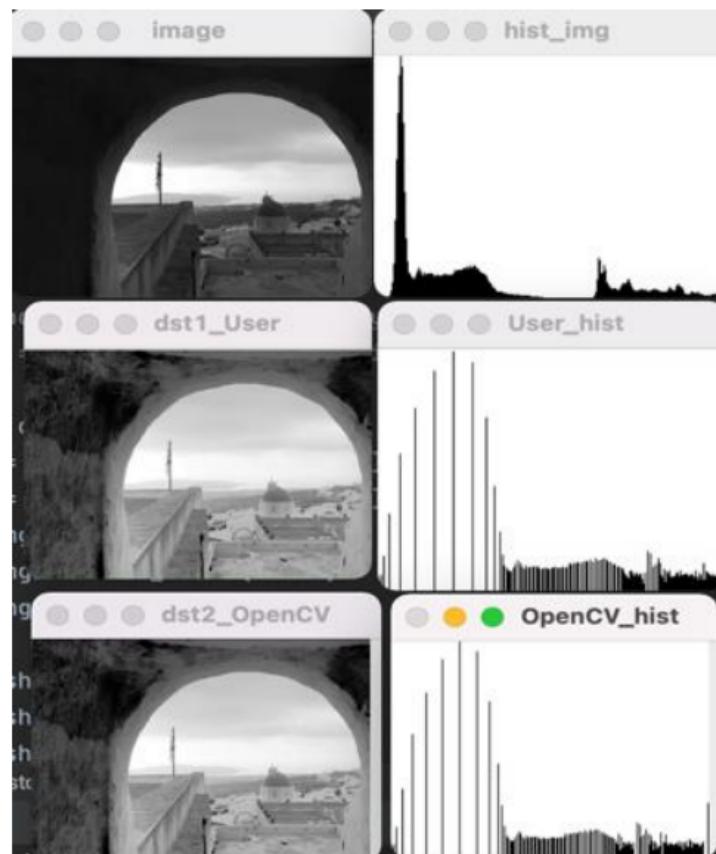
이퀄라이저 기능 - 평활화시키는 것.

분포는 돼있으나 빈도가 몰려있는 것을 어느 정도 펼쳐줌. 빈도를 손상시키지 않는 선에서 품질을 살려주는 방법으로 만들어줌.

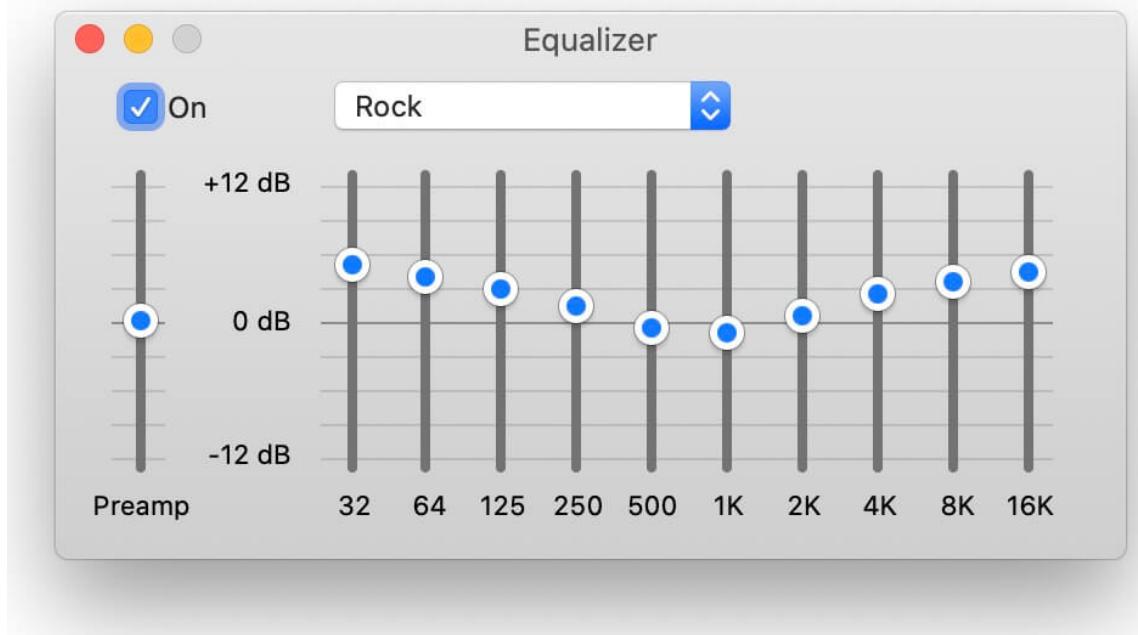
스트레칭이 효과가 없을 때 사용하면 좋다.

히스토그램

- ❖ Histogram Equalization(평활화)
 - ✓ 평활화



음악에서는



히스토그램 평활화를 수행하는 전체 과정

- 영상의 히스토그램을 계산
- 히스토그램 빈도 값에서 누적 빈도수(누적합)를 계산
- 누적 빈도수를 정규화(정규화 누적합)
- 결과 픽셀값 = 정규화 누적합 * 최대 픽셀값

Histogram Equalization(평활화)

- ✓ 평활화 계산 과정

| | | | |
|---|---|---|---|
| 0 | 2 | 2 | 1 |
| 1 | 2 | 3 | 2 |
| 1 | 2 | 3 | 2 |
| 1 | 3 | 1 | 7 |

입력 영상 화소값

| | | | |
|---|---|---|---|
| 0 | 5 | 5 | 3 |
| 3 | 5 | 7 | 5 |
| 3 | 5 | 7 | 5 |
| 3 | 7 | 3 | 7 |

평활화 완료 영상 화소값

| 화소값 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----------|--------|-------|-------|--------|--------|--------|--------|-------|
| 빈도수 | 1 | 5 | 6 | 3 | 0 | 0 | 0 | 1 |
| 누적 빈도수 | 1 | 6 | 12 | 15 | 15 | 15 | 15 | 16 |
| 정규화누적합 | 1/16 | 6/16 | 12/16 | 15//16 | 15//16 | 15/16 | 15/16 | 16/16 |
| | 0.0625 | 0.375 | 0.75 | 0.9375 | 0.9375 | 0.9375 | 0.9375 | 1 |
| 누적합 * 최댓값 | 0.4375 | 2.625 | 5.25 | 6.5625 | 6.5625 | 6.5625 | 6.5625 | 7 |
| 평활화 결과 | 0 | 3 | 5 | 7 | 7 | 7 | 7 | 7 |

| 화소값 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----------|--------|-------|-------|--------|--------|--------|--------|-------|
| 빈도수 | 1 | 5 | 6 | 3 | 0 | 0 | 0 | 1 |
| 누적 빈도수 | 1 | 6 | 12 | 15 | 15 | 15 | 15 | 16 |
| 정규화누적합 | 1/16 | 6/16 | 12/16 | 15//16 | 15//16 | 15/16 | 15/16 | 16/16 |
| | 0.0625 | 0.375 | 0.75 | 0.9375 | 0.9375 | 0.9375 | 0.9375 | 1 |
| 누적합 * 최댓값 | 0.4375 | 2.625 | 5.25 | 6.5625 | 6.5625 | 6.5625 | 6.5625 | 7 |
| 평활화 결과 | 0 | 3 | 5 | 7 | 7 | 7 | 7 | 7 |

10_histogram_equalization.py

```
import numpy as np, cv2
from Common.histogram import draw_histo

image = cv2.imread("images/equalize.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류")

# 영상의 히스토그램 계산
bins, ranges = [256], [0, 256]
hist = cv2.calcHist([image], [0], None, bins, ranges)

# 히스토그램 누적합 계산
accum_hist = np.zeros(hist.shape[:2], np.float32) # histogram의 행과 열
accum_hist[0] = hist[0]
for i in range(1, hist.shape[0]):
    accum_hist[i] = accum_hist[i - 1] + hist[i]

# 누적합의 정규화
accum_hist = (accum_hist / sum(hist)) * 255
dst1 = [[accum_hist[val] for val in row] for row in image]
dst1 = np.array(dst1, np.uint8)
```

```

dst2 = cv2.equalizeHist(image) # histogram으로 평활화시킴

hist1 = cv2.calcHist([dst1], [0], None, bins, ranges) # 내가 계산한 것
hist2 = cv2.calcHist([dst2], [0], None, bins, ranges) # cv2 평활화 기능을 사용해본 것

hist_img = draw_histo(hist) # 원 이미지에 대한 시각화
hist1_img = draw_histo(hist1)
hist2_img = draw_histo(hist2)

cv2.imshow("image", image)
cv2.imshow("hist_img", hist_img)

cv2.imshow("dst1_directly", dst1)
cv2.imshow("hist1_directly", hist1_img)

cv2.imshow("dst2_cv", dst2)
cv2.imshow("hist2_cv", hist2_img)

cv2.waitKey(0)

```



magenta

미국·영국 [məˈdʒentə] [

마젠타색, 자홍색

<https://www.thefreedictionary.com/magenta>

가시광선, 적외선, 자외선

생활 속 태양광 자외선 측정

자외선 강도는 오후 5경이 되어서야 다시 2.06mW/cm 으로 떨어졌네요. 즉, 1시~3시 기준으로 가장 높은 수준 4.5~5mW/cm 으로 올라가는 것으로 보입니다. 그 각각 두시간 전후에는 2~2.5mW/cm 수준으로

 https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=consumer_quotient&logNo=221040297317



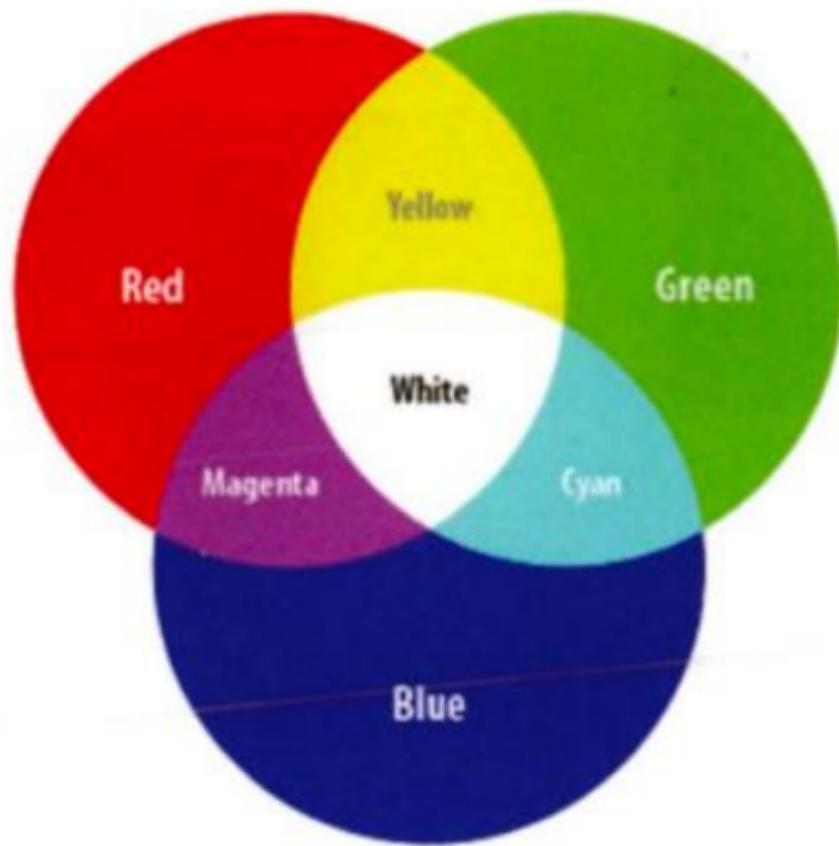
메모

- 계급은 클래스(class)라고 부른다. 또 히스토그램의 세로축에 도수를 나타내는 'f'를 쓰기도 한다. 이것은 도수를 뜻하는 영단어 frequency의 앞글자다.

가산 혼합

RGB 컬러 공간

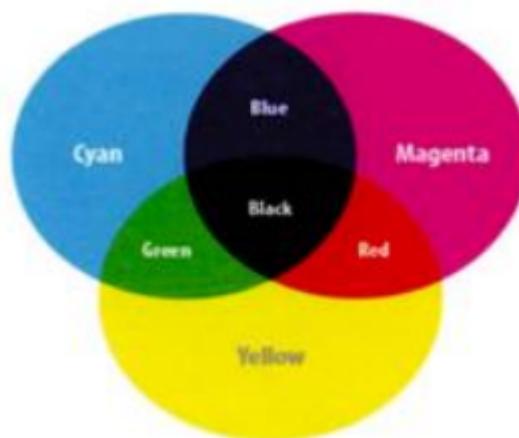
- ✓ 빛을 이용해서 색을 만들려면 빨강 빛, 초록 빛, 파랑 빛이 반드시 필요한데 이것을 빛의 삼원색이라 부름
- ✓ 원색이란 더이상 분해할 수 없는 최소한의 색을 의미하며 이 색들을 조합해서 다른 색을 표현할 수 있음
- ✓ 이 원색의 조합은 그림과 같이 섞을 수록 밝아지기 때문에 가산 혼합(additive mixture)이라 함



감산 혼합

그림과 같이 색의 삼원색은 빛의 삼원색과 보색 관계에 있는 청록색(Cyan), 자홍색(Magenta), 노랑색(Yellow)를 의미

이 원색들은 흰색으로부터 감산되어 색이 만들어지기 때문에 감산 혼합(subtractive mixture)



순수한 검은색을 출력하기 위해서 CMY 컬러 모델에 검은색(black)을 추가하여 CMYK 컬러 공간으로 사용하는 것이 일반적

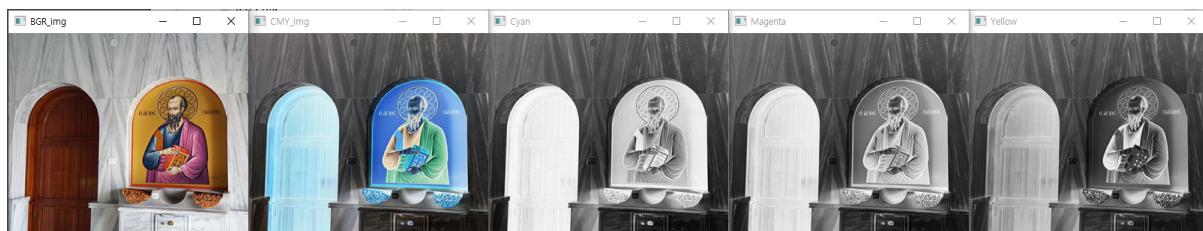
11_convert_cmy.py

```
import numpy as np, cv2

BGR_img = cv2.imread("images/color_model.jpg", cv2.IMREAD_COLOR)
if BGR_img is None: raise Exception("영상 파일 읽기 오류")

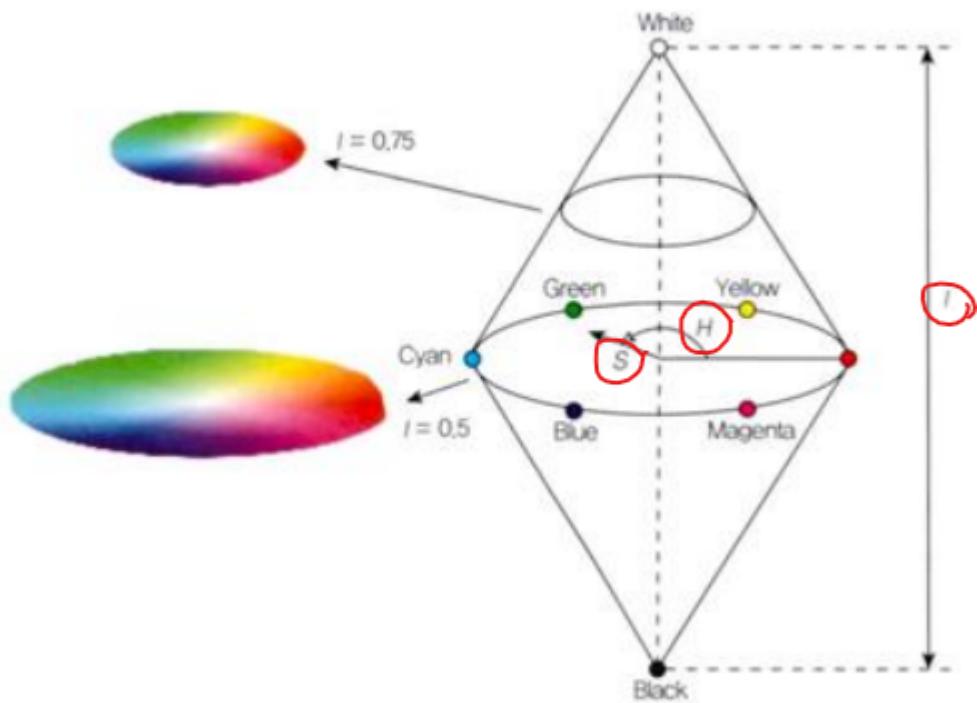
white = np.array([255, 255, 255], np.uint8)
CMY_img = white - BGR_img
Cyan, Magenta, Yellow = cv2.split(CMY_img)

titles = ['BGR_img', 'CMY_img', 'Cyan', 'Magenta', 'Yellow']
[cv2.imshow(t, eval(t)) for t in titles]
cv2.waitKey(0)
```



HSI 컬러 공간

- ✓ 인간이 컬러 영상 정보를 인지하는 방법은 색상(Hue), 채도(Saturation), 명도(Intensity)라는 세 가지 지각 변수로 분류



색상(Hue) - 각도

채도(Saturation) - 불순물의 정도.

명도(Intensity) - 흰색, 검은색까지 표현. 빛의 밝기, 세기

$HSI = HSV(\text{Value})$

HSL Color Picker

The HSL Color Picker is a simple, yet effective color workbench to generate color output and also to import colors for analysis and comparison. It has all the features you need for experimenting with the seven color contrasts in mind.



<http://www.workwithcolor.com/hsl-color-picker-01.htm>

12_convert_hsv.py

```

import numpy as np, cv2, math

def calc_hsi(bgr):
    B, G, R = float(bgr[0]), float(bgr[1]), float(bgr[2])
    bgr_sum = (R+G+B)

    # 색상 계산
    tmp1 = ((R-G)+(R-B))*0.5 # 분자
    tmp2 = math.sqrt((R-G)*(R-G)+(R-B)*(G-B)) # 분모
    angle = math.acos(tmp1 / tmp2) * (180 / np.pi) if tmp2 else 0

    H = angle if B <= G else 360 - angle
    S = 1.0 - 3 * min([R,G,B]) / bgr_sum if bgr_sum else 0
    I = bgr_sum / 3

    return (H/2, S * 255, I)

# BGR 컬러 -> HSI 컬러
def bgr2hsd(image):
    hsv = [[calc_hsi(pixel) for pixel in row] for row in image]
    return (np.array(hsv)).astype('uint8')

BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR)
if BGR_img is None: raise Exception("영상 파일 읽기 오류")

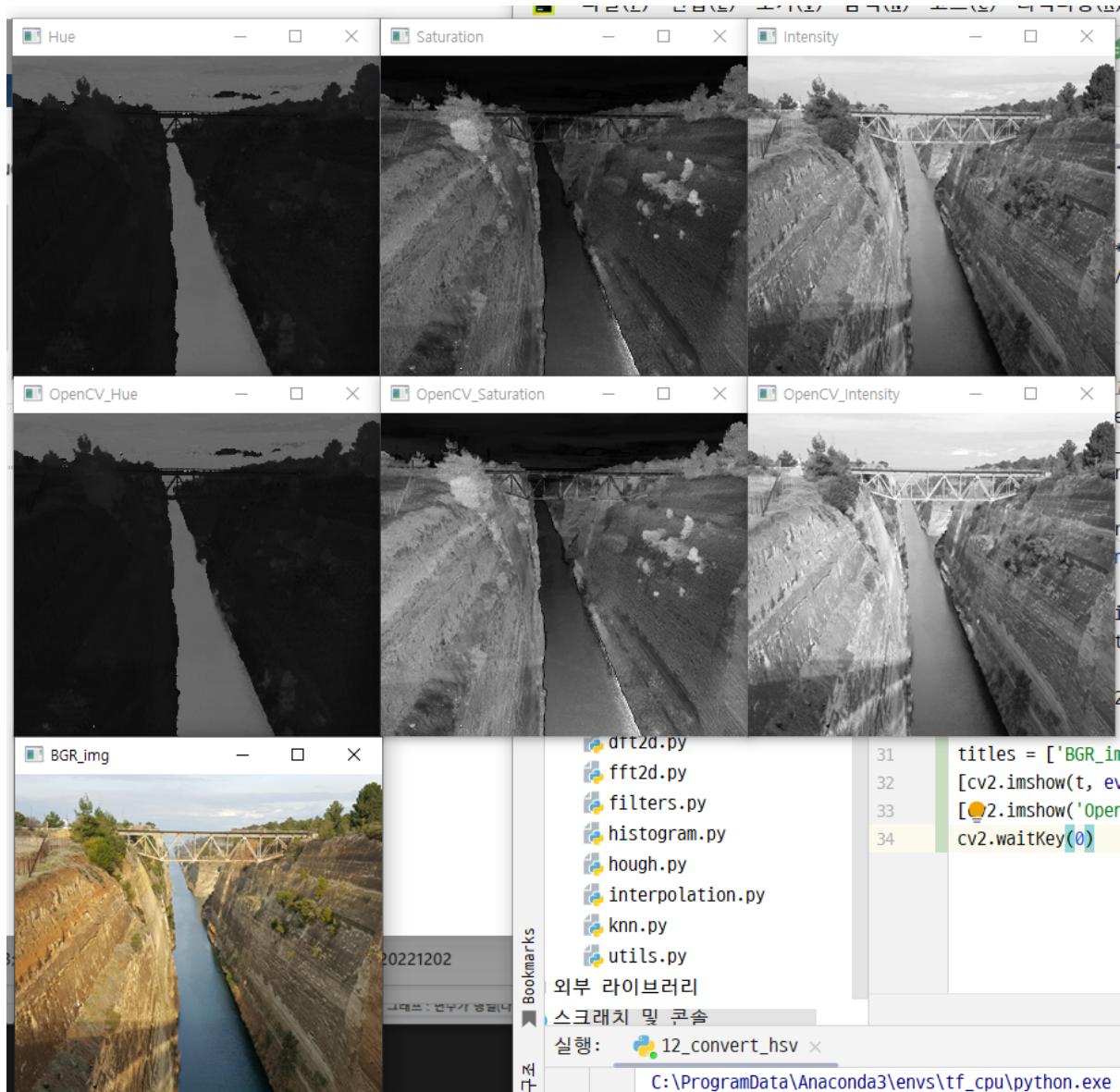
HSI_img = bgr2hsd(BGR_img)
HSV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2HSV)
Hue, Saturation, Intensity = cv2.split(HSI_img)
Hue2, Saturation2, Intensity2 = cv2.split(HSV_img)

```

```

titles = ['BGR_img', 'Hue', 'Saturation', 'Intensity']
[cv2.imshow(t, eval(t)) for t in titles]
[cv2.imshow('OpenCV_'+t, eval(t+'2')) for t in titles[1:]]
cv2.waitKey(0)

```

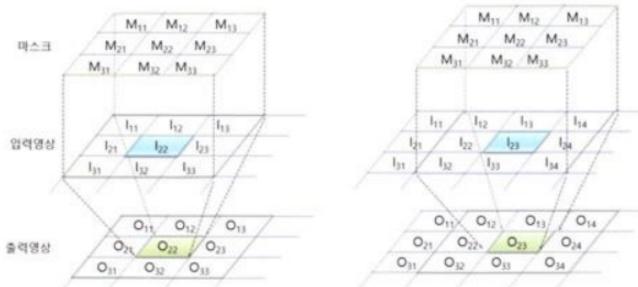


영역 처리

회선(Convolution)

❖ 공간 영역의 개념과 회선(Convolution)

- ✓ 영상 처리에서 영역에 대한 하나의 의미는 두 개의 다른 범위(domain)의 구분으로 공간 영역(spatial domain)이며 다른 하나는 주파수 영역(frequency domain)
- ✓ 영역에 대한 다른 의미는 영역 기반 처리(area based processing)라는 표현에서 사용하는 영역인데 픽셀이 모인 특정 범위(영역)의 픽셀 배열을 의미
- ✓ 픽셀 기반 처리가 픽셀 값 각각에 대해 여러 가지 연산을 수행하는 것이라면 영역 기반 처리는 마스크(mask)라 불리는 규정된 영역을 기반으로 연산이 수행되기 때문에 영역 기반 처리를 마스크 기반 처리라고도 함
- ✓ 마스크 기반 처리는 마스크 내의 픽셀 값과 공간 영역에 있는 입력 영상의 픽셀 값을 대응되게 곱하여 출력 픽셀 값을 계산하는 것을 의미하는데 이러한 처리를 모든 출력 픽셀 값에 대해 이동하면서 수행하는 것을 회선(convolution)이라고 함
- ✓ 입력 영상에 곱해지는 이 마스크는 커널(kernel), 윈도우(window), 필터(filter) 등의 이름으로도 부름



Convolution(합성곱) = 회선

회선(Convolution)

❖ 블러링(blurring)

- ✓ 블러링은 영상에서 픽셀값이 급격하게 변하는 부분들을 감소시켜 점진적으로 변하게 함으로써 영상이 전체적으로 부드러운 느낌이 나게 하는 기술인데 스무딩(smoothing)이라 하기도 함
- ✓ 급격히 변화하는 것을 점진적으로 변하게 하려면 커널 마스크를 이용해서 회선을 수행하면 됨

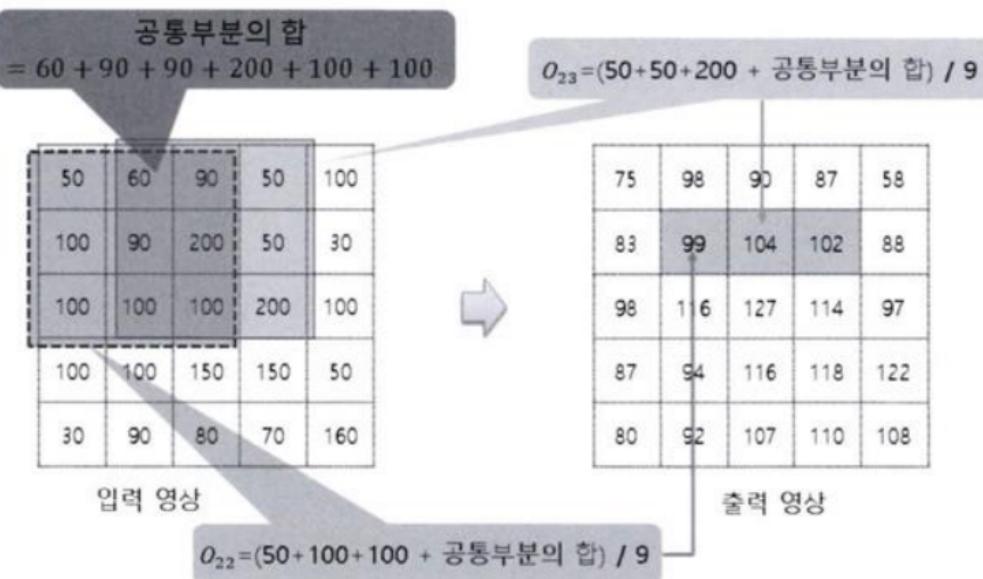
| | | |
|---------------|---------------|---------------|
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |
| $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ |

| | | | | |
|----------------|----------------|----------------|----------------|----------------|
| $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ |
| $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ |
| $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ |
| $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ |
| $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ | $\frac{1}{25}$ |

회선(Convolution)

❖ 블러링

- ✓ 3×3 마스크로 회선



chap_01_blurring.py

```
import numpy as np, cv2, time ▲ 1

def blur_convolution(image, filter):
    rows, cols = image.shape[:2]
    # height, width로 생각하는 게 더 쉽다.
    # 300, 270
    dst = np.zeros((rows, cols), np.float32)
    xcenter, ycenter = filter.shape[1]//2, filter.shape[0] // 2

    for i in range(ycenter, rows - ycenter):
        for j in range(xcenter, cols - xcenter):
            y1, y2 = i - ycenter, i + ycenter + 1 # 1, 3 저장 돼있음
            x1, x2 = j - xcenter, j + xcenter + 1 # 1, 3 저장 돼있음
            roi = image[y1:y2, x1:x2].astype("float32")

            tmp = cv2.multiply(roi, filter)
            dst[i, j] = cv2.sumElems(tmp)[0]
    return dst

image = cv2.imread("images/filter.blur.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류 발생")

# 블러링 마스크 원소 지정
filter = [1/9, 1/9, 1/9,
          1/9, 1/9, 1/9,
          1/9, 1/9, 1/9]

mask = np.array(filter, np.float32).reshape(3, 3)
blur = blur_convolution(image, mask)
```

```
cv2.imshow("image", image)
cv2.imshow("blur", blur.astype("uint8"))

cv2.waitKey(0)
```

