Chapter

2

Image Processing

1. Image 기본연산
2. Geometric Transformations
3. Image Thresholding

학습 목표

✓ OpenCV를 이용한 영상처리를 한다.

주요 내용

✓ Image 기본 조작
✓ Image 변형
✓ Image 이진화

2

1. Image 기본 연산

- ✓ 이미지 생성
- ✓ 이미지 복사
- ✓ 이미지 연산
- ✓ 픽셀 값 접근 및 수정
- ✓ 이미지 속성 접근
- ✓ 관심 영역 (ROI) 설정
- ✓ 이미지 채널 분할 및 병합

이미지 생성

❖ numpy를 이용하여 이미지 행렬 객체 생성

```
img1 = np.empty((200, 300), np.uint8)       # grayscale image
img2 = np.zeros((200, 300, 3), np.uint8)    # color image
img3 = np.full((200, 300), 100, np.uint8)   # Fill with 100

mat1 = np.array([[10, 120, 130, 140, 150],
                 [210, 202, 230, 240, 250],
                 [310, 320, 330, 340, 350],
                 [110, 120, 130, 140, 150],
                 [60, 70, 80, 90, 100]]).astype(np.uint8)

mat1[0, 1] = 100    # element at x=1, y=0
mat1[2, :] = 200

print("img1=", img1[0,0])
print("img2=", img2[0,0])
print("img3=", img3[0,0])
print("mat1 = ",  mat1)

cv2.imshow('img1', img1)
cv2.imshow('img2', img2)
cv2.imshow('img3', img3)
cv2.imshow('mat1', mat1)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
img1= 80
img2= [0 0 0]
img3= 100
mat1 =  [[100 100 130 140 150]
 [210 202 230 240 250]
 [200 200 200 200 200]
 [110 120 130 140 150]
 [ 60  70  80  90 100]]
```

이미지 복사

Image copy

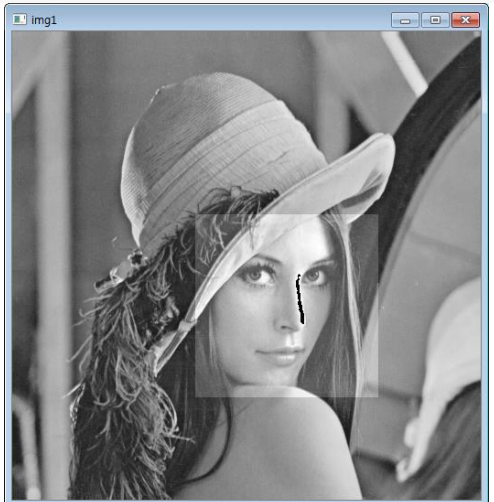
```
img1 = cv2.imread('images/cat.png')

if img1 is None:
        print('Image load failed!')
        sys.exit()

img2 = img1
img3 = img1.copy()

img1[:50, :50] = (0, 255, 255)  # yellow

cv2.imshow('img1', img1)
cv2.imshow('img2', img2)
cv2.imshow('img3', img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

이미지 연산

❖ 이미지 추출 및 연산

```
img1 = cv2.imread('images/lenna.bmp', cv2.IMREAD_GRAYSCALE)

img2 = img1[200:400, 200:400]
img3 = img1[200:400, 200:400].copy()

img2 +=  40

cv2.imshow('img1', img1)
cv2.imshow('img2', img2)
cv2.imshow('img3', img3)

img4 = cv2.add(img3, 40)  # saturated operation
cv2.imshow('img4', img4)

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

$$\text{saturate}(x) = \begin{cases} 0 & x < 0 \text{ 일 때} \\ 255 & x > 255 \text{ 일 때} \\ x & \text{그 외} \end{cases}$$



이미지 연산

❖ 수평 Flip 연산

```
#행렬  flip
mat1 = np.array(np.arange(25).reshape(5, 5))
print('mat1:', mat1)

h, w = mat1.shape[:2]

mat2 = np.zeros(mat1.shape, mat1.dtype)

for j in range(h):
        for i in range(w):
            mat2[j, i] = mat1[h-1-j, i]

print('mat2:')
print(mat2)

# image flip
img1 = cv2.imread('images/cat.png', cv2.IMREAD_GRAYSCALE)

if img1 is None:
        print('Image load failed!')
        sys.exit()

h, w = img1.shape[0:2]

img2 = np.zeros(img1.shape, img1.dtype)
print("img1 type=", img1.dtype)
print("img1 shape=", img1.shape)
print("img2 type=", img2.dtype)
print("img2 shape=", img2.shape)


for j in range(h):
            img2[j, :(w-1)] = img1[h-1-j, :(w-1)]

cv2.imshow('img1', img1)
cv2.imshow('img2', img2)

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

```
mat1: [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
mat2:
[[20 21 22 23 24]
 [15 16 17 18 19]
 [10 11 12 13 14]
 [ 5  6  7  8  9]
 [ 0  1  2  3  4]]
img1 type= uint8
img1 shape= (192, 256)
img2 type= uint8
img2 shape= (192, 256)
```

이미지 속성

❖ 이미지 속성(Image Properties) 확인

```
#칼라 이미지 크기, 유형, 총 픽셀수 확인
print("image.shape = ", image.shape)  #행(높이), 열(폭), 채널수
print("image.shape len = ", len(image.shape) )
print("image.dtype = ", image.dtype)
print("image.size = ", image.size)

#이미지 높이, 폭, 채널수 확인
imgHeight, imgWidth, channel = image.shape
print(fname, imgHeight, imgWidth, channel)
s = imgHeight * imgWidth * channel
print("image.size = ", s)

#그레이스케일 이미지 크기, 유형, 총 픽셀수 확인
print("₩ngray_image.shape = ", gray_image.shape)  #행(높이), 열(폭), 채널수
print("gray_image.shape len = ", len(gray_image.shape) )
print("gray_image.dtype = ", gray_image.dtype)
print("gray_image.size = ", gray_image.size)
```

```
image.shape =  (512, 512, 3)
image.shape len =  3
image.dtype =  uint8
image.size =  786432
lenna.bmp 512 512 3
image.size =  786432

gray_image.shape =  (512, 512)
gray_image.shape len =  2
gray_image.dtype =  uint8
gray_image.size =  262144
```

이미지 픽셀 접근

❖ pixel 직접 접근

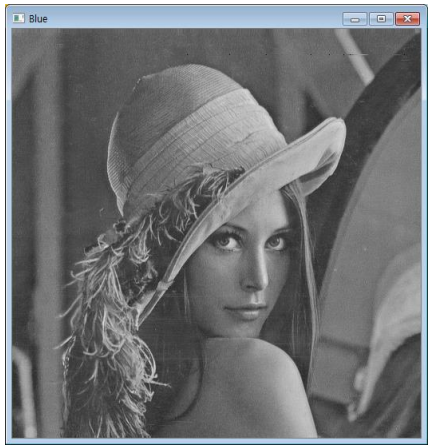
```
# x, y좌표로 직접 접근하여 칼라 픽셀값 얻기
px = image[100,100]
print("image(100,100) = ", px )

# x, y좌표, 칼라채널로 직접 접근하여 각 색상값 얻기
blue = image[100,100,0]
print( "blue = ", blue )
green = image[100,100,1]
print( "green = ", green )
red = image[100,100,2]
print( "red = ", red )

# x, y좌표로 직접 접근하여  그레이스케일 픽셀값 얻기
px = gray_image[100,100]
print("gray image(100,100) = ", px )

gray_image[100,100] = 0
px = gray_image[100,100]
print("gray image(100,100) = ", px )
```

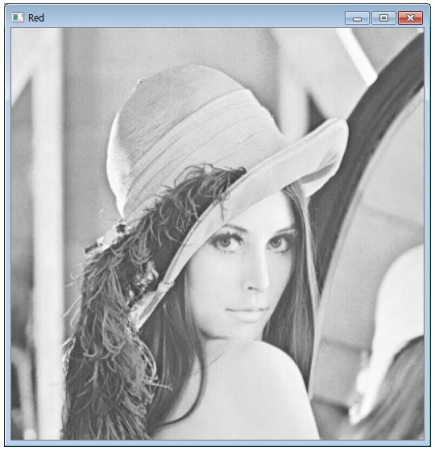
```
image(100,100) =  [ 78  68 178]
blue =  78
green =  68
red =  178
gray image(100,100) =  102
gray image(100,100) =  0
```

채널 분리

❖ 칼라 이미지 채널 분리 및 결합 (Splitting and Merging Image Channels)

```
b,g,r = cv2.split(image)
cv2.imshow("Blue", b)
cv2.imshow("Green", g)
cv2.imshow("Red", r)
newimg = cv2.merge((b,g,r))
cv2.imshow("newImage", newimg)

# ESC 키 입력 시 Windows 닫음
cv2.waitKey(0)
cv2.destroyAllWindows()
```



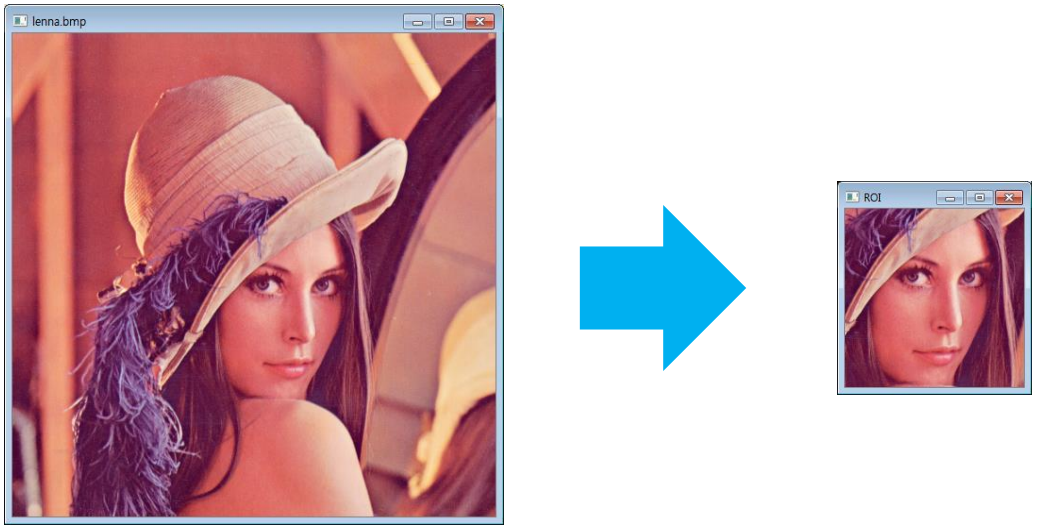


ROI

❖ 이미지 관심 영역 (Image ROI, regions of images)

- 이미지의 특정 영역 좌표를 이용하여 관심 영역추출

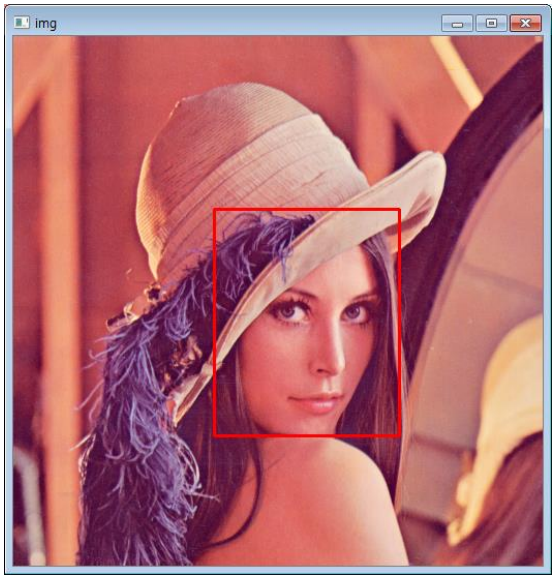
```
#lenna.bmp에서 입 영역 직접 추출
clone = image.copy()  #이미지를 복사
sx = 190        # x 시작좌표
ex = sx + 200    # x 끝좌표
sy = 190        # y 시작좌표
ey = sy + 200    # y 끝좌표
ROI = clone[sy:ey,sx:ex]   #영역 추출
cv2.imshow("ROI", ROI)
# ESC 키 입력 시 Windows 닫음
cv2.waitKey(0)
cv2.destroyAllWindows()
```



ROI

❖ 마우스로 ROI 설정하여 추출

```python
#마우스 이벤트 처리
#왼쪽 버튼을 누르는 위치를 시작점, 버튼을 떼는 위치를 끝점으로 설정
#마우스 이동시 사각형을 그려 영역 표시
isDragging = False
sx, sy, w, h = -1, -1, -1, -1
blue, red = (255, 0, 0), (0, 0, 255)

def onMouse(event, x, y, flags, param):
    global isDragging, sx, sy, image
    if event == cv2.EVENT_LBUTTONDOWN:  #왼쪽 버튼을 누르면 드래그 시작, 시작좌표 설정
        isDragging = True
        sx = x
        sy = y
    elif event == cv2.EVENT_MOUSEMOVE:  #왼쪽 버튼을 누른채 이동하면 이미지에 파란색 사각형 그리기
        if isDragging:
            img_draw = image.copy()
            cv2.rectangle(img_draw, (sx, sy), (x, y), blue, 2)
            cv2.imshow('img', img_draw)
    elif event == cv2.EVENT_LBUTTONUP:  #왼쪽 버튼을 떼면
        if isDragging:
            isDragging = False          #드래그 설정 취소
            w = x - sx                  #현재위치에서 시작위치를 빼서 폭과 높이 계산
            h = y - sy
            if w > 0 and h > 0:
                img_draw = image.copy()
                cv2.rectangle(img_draw, (sx, sy), (x, y), red, 2)  #빨간색 사각형으로 최종 영역 확인
                cv2.imshow('img', img_draw)
                roi = image[sy:sy+h, sx:sx+w]          #관심영역(roi)추출
                cv2.imshow('cropped', roi)
                cv2.moveWindow('cropped', 0, 0)     # 관심 영역의 추출 창의 위치를 (0,0) 화면 왼쪽위로 이동
                cv2.imwrite('images/cropped.png', roi)  #관심영역을 파일로 저장
            else:
                cv2.imshow('img', image)
                print('drag should start from left-top side')

#image를 Widnows로 출력
cv2.imshow('img', image)
#마우스 버튼클릭이벤트 ROI 설정
cv2.setMouseCallback('img', onMouse)
cv2.waitKey()
cv2.destroyAllWindows()
```



이미지 산술 연산

❖ 이미지결합(image addition)

dst res = img1 + img2

- 두 이미지의 유형과 채널이 동일
- cv2.add(x,y)
- saturated operation : 250+10 = 260 => 255

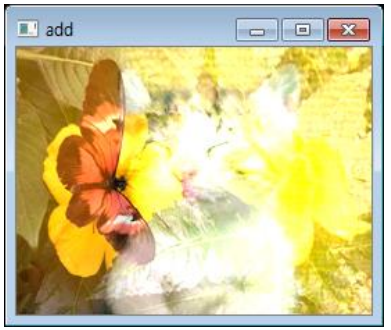
❖ 이미지 혼합(Image Blending)

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

- cv2.addWeighted(img1,alpha,img2,beta, gamma)

Image Addition & Blending

```
import cv2
import numpy as np

img1 = cv2.imread('images/flower1.png')
img2 = cv2.imread('images/cat.png')

dst1 = cv2.add(img1,img2)

a = 0.7
dst2 = cv2.addWeighted(img1, 1-a, img2, a, 0)

cv2.imshow('img1', img1)
cv2.imshow('img2', img2)
cv2.imshow('add', dst1)
cv2.imshow('addWeighted', dst2)

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



QUIZ!

1. ‘cat.png’ 를 그레이스케일로 변환하고 밝기를 +30, -30으로 변환된 이미지를 출력 하세요.
2. ‘cat.png’에 대해 수직 Flip 처리 결과를 출력하세요.
3. ‘lenna.bmp’ 를 채널로 분리하고 R채널 값을 모두 0으로 수정하여 결합한 이미지를 출력하세요.
4. 본인 얼굴과 타인 얼굴 사진을 이용하여 블랜딩 해보세요.
5. 축구장면 이미지에서 축구공만을 ROI로 추출해서 파일로 저장하세요.

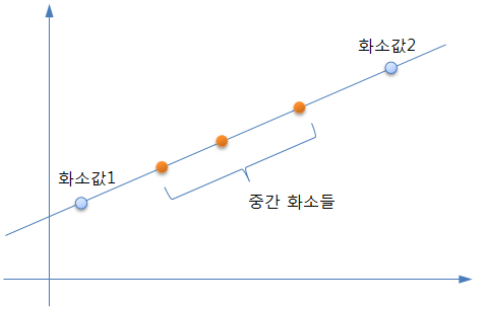
2. Geometric Transformations

- ✓ Scaling
- ✓ Rotation
- ✓ Perspective Transformation

이미지 크기 변환(Scaling)

❖ 이미지의 크기 변환

- cv2.resize(img, dsize, fx, fy, interpolation)
  - dsize : (w,h) size
  - fx, fy : 가로 x 사이즈의 배수, 세로 y 사이즈의 배수
  - Interpolation : 보간법
    - cv2.INTER_LINEAR: default, bilinear interpolation, 직선의 선상에 위치한 중간 화소 들의 값을 직선 방정식을 이용해서 계산하는 방법
    - cv2.INTER_NEAREST: nearest neighbor interpolation,주변 이웃 pixel값을 가져와 빈공간을 채움.
    - cv2.INTER_AREA: 주변 pixel의 관계에 따라 resample하는 방식, 크기 축소에 사용, 확대 시 nearest neighbor와 유사.
    - cv2.INTER_CUBIC - 4x4 이웃 pixel에 대한 bicubic interpolation, bilinear보다 부드럽고 lancozs보다 계산량이 적어 많이 사용.
    - cv2.LANCZOS4 - 8x8 이웃 pixel에 대한 lancozs interpolation, 가장 부드럽게 값을 메우지만 연산이 많이 필요.



이미지 크기 변환(Scaling)

```python
import cv2
import numpy as np

img = cv2.imread('images/cat.png')

#Image 2배 확대
#dest = cv.resize(img,None,fx=2, fy=2, interpolation = cv.INTER_CUBIC)
#OR
height, width = img.shape[:2]
des1 = cv2.resize(img,(2*width, 2*height), interpolation = cv2.INTER_CUBIC)

#Image 1/2 축소
des2 = cv2.resize(img,(width // 2, height // 2), interpolation = cv2.INTER_AREA)

cv2.imshow('img', img)
cv2.imshow('2X INTER_CUBIC', des1)
cv2.imshow('1/2', des2)

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

이미지 회전(Rotation)

❖ Rotation

- 회전할 각도에 맞춰 scaling 정도를 계산할 matrix를 생성
  - `getRotationMatrix2D`(*center*, *angle*, *scale*)
- RotationMatrixx를 사용하여 image를 변형
  - cv2.warpAffine(image, Matrix, (width, height))


```
#이미지의 중심점을 기준으로 90도 회전 하면서 0.5배 Scale
M = cv2.getRotationMatrix2D (((width-1) /2.0, (height-1) /2.0), 90,1)
des3 = cv2.warpAffine (img, M, (width, height))
```

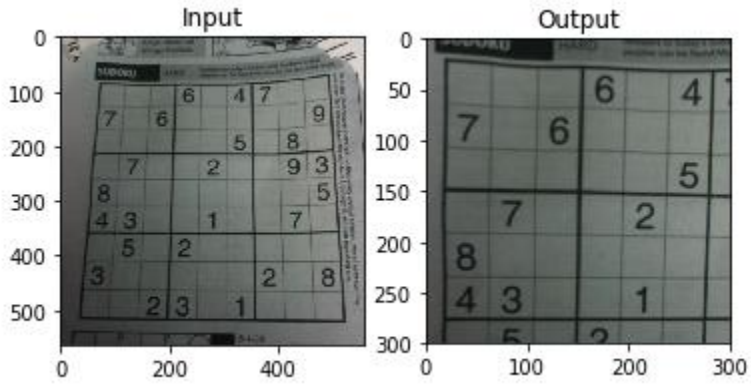
원근변환(Perspective Transformation)

❖ 원근 변환

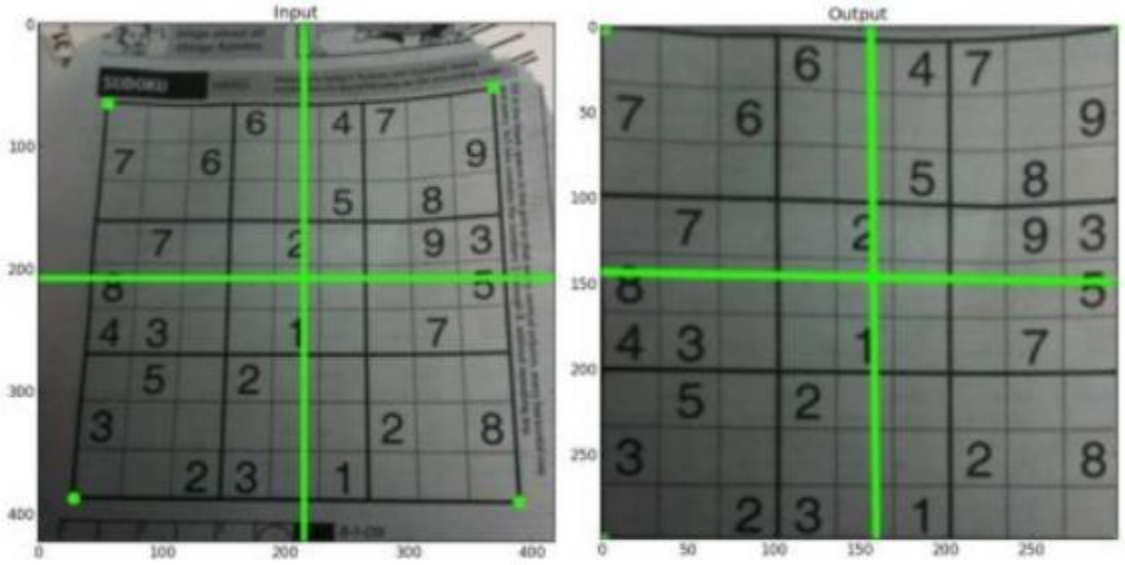
- 3x3 원근 변환 행렬을 찾아서 변환 처리
- 변환 후에도 직선은 유지
- 변환 행렬을 찾으려면 4개의 입력 이미지 점과 출력 이미지 점이 필요 (4 점 중 3 점은 동일 선상에 있지 않아야 함)
- cv.getPerspectiveTransform 함수로 변환 행렬을 찾고, 3x3 변환 행렬로 cv.warpPerspective 를 적용

```
import matplotlib.pyplot as plt

img = cv2.imread('images/sudoku.png')
rows,cols,ch = img.shape

#[x,y] 좌표점을 4x2의 행렬로 작성 (입력 점)
# 좌표점은 좌상->좌하->우상->우하
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
# 좌표의 이동점 (출력 점)
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

M = cv2.getPerspectiveTransform(pts1,pts2)

print(M) #3X3 변환행렬 출력

dst = cv2.warpPerspective(img,M,(300,300))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

```
[[ 1.05587376e+00  9.18151097e-02 -6.50969128e+01]
 [ 4.69010049e-02  1.12562412e+00 -7.57920240e+01]
 [ 1.83251448e-04  5.13337001e-04  1.00000000e+00]]
```



원근변환(Perspective Transformation



image

2

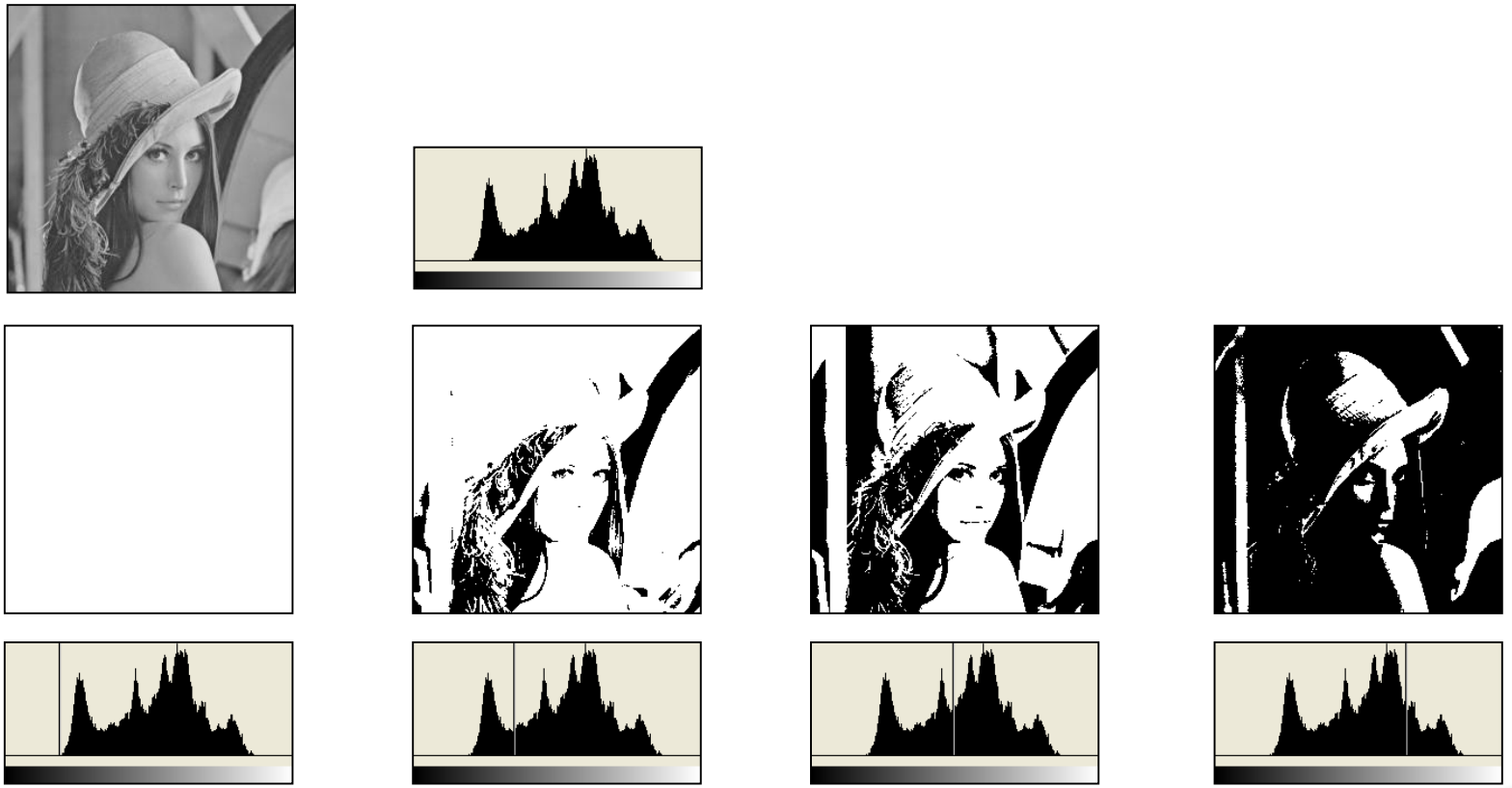
3. Image Thresholding

- ✓ Simple thresholding
- ✓ Adaptive thresholding
- ✓ Otsu's thresholding
- ✓ histogram

Thresholding

❖ Thresholding

- 이진영상(binary images)으로 변환하기 위한 방법
- Threshold 에 따른 이진화 결과 예



Thresholding

❖ 단순 임계값 (Simple Thresholdin

- **global thresholding**
- 특정한 임계값(threshold)을 기준으로 pixel 값을 0 또는 max(255)로 변환

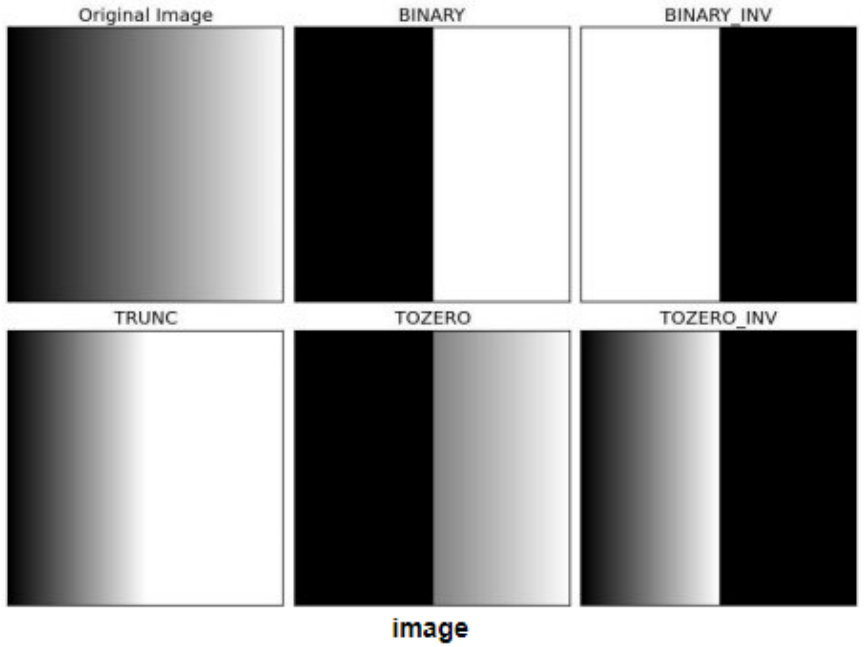
cv2.threshold()

- cv2.THRESH_BINARY: threshold를 기준으로 작으면 0, 크면 max
- cv2.THRESH_BINARY_INV: 위의 반대
- cv2.THRESH_TRUNC: threshold보다 작으면 그대로, 크면 max
- cv2.THRESH_TOZERO: threshold보다 작으면 min, 크면 그대로
- Cv2.THRESH_TOZERO_INV: 위의 반대

Thresholding

- Simple Thresholding

```
img = cv2.imread('images/gradient.png',cv2.IMREAD_GRAYSCALE)
ret,thresh1 = cv2.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



image

Thresholding

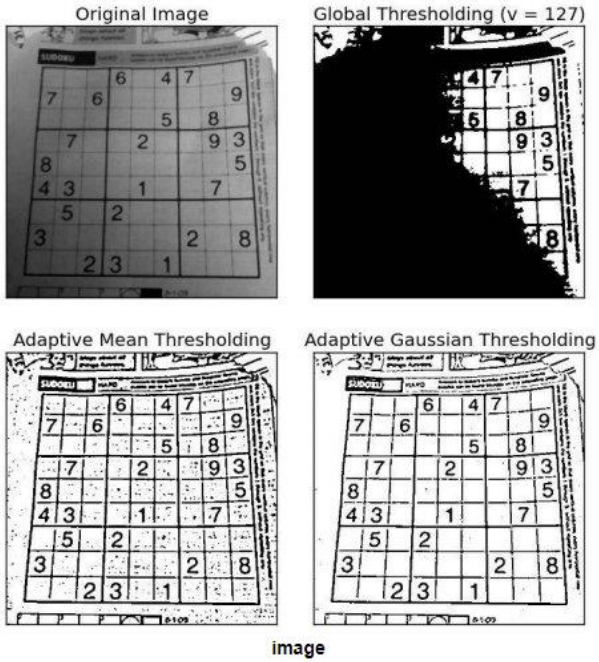
❖ 적응 형 임계 값(Adaptive Thresholding)

- 단순 임계값 방법은 하나의 전역 값을 임계 값으로 사용하므로 영역별 다른 조명 조건을 갖는 경우 좋은 결과를 얻을 수 없다.
- 적응 임계 값은 주변의 작은 영역을 기반으로 픽셀의 임계 값을 결정하는 방법
- cv2.adaptiveThreshold()
  - cv2.ADAPTIVE_THRESH_MEAN_C : 임계 값은 주변 지역의 평균에서 상수 C 를 뺀 값을 이용 .
  - cv2.ADAPTIVE_THRESH_GAUSSIAN_C : 임계 값은 주변 값의 가우스 가중치 합계에서 상수 C 를 뺀 값을 이용 . .

Thresholding

❖ Adaptive Thresholding

```
img = cv2.imread('images/sudoku.png',0)
img = cv2.medianBlur(img,5)
ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,₩
            cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,₩
            cv2.THRESH_BINARY,11,2)
titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
plt.figure(figsize=(8,8))
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



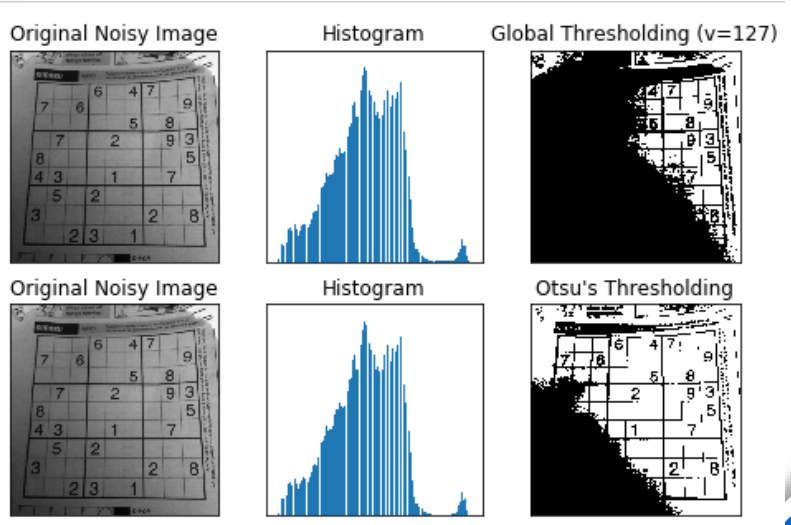
image

Thresholding

❖ Otsu's Binarization

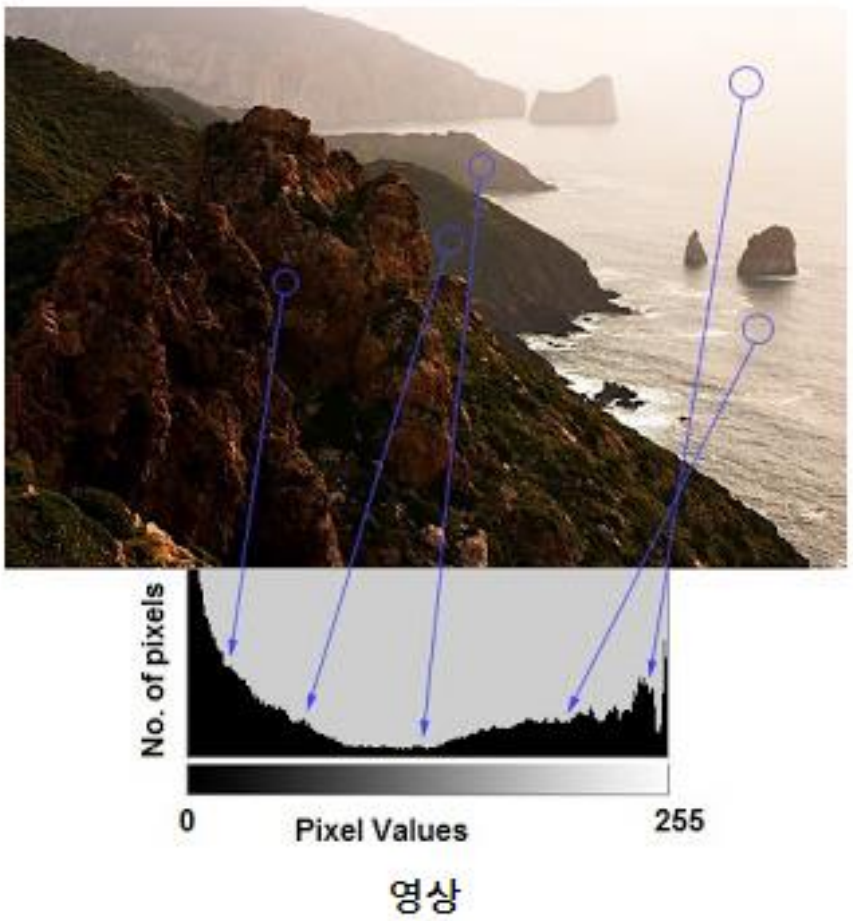
- 이미지의 히스토그램이 두 개의 피크로만 구성된 경우 두 피크값의 중간에 적절한 임계 값이 있다는 가정으로 이미지 히스토그램에서 최적의 글로벌 임계 값을 결정하는 방법

```
img = cv2.imread('images/sudoku.png',0)
# global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding"]
plt.figure(figsize=(8,8))
for i in range(2):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```



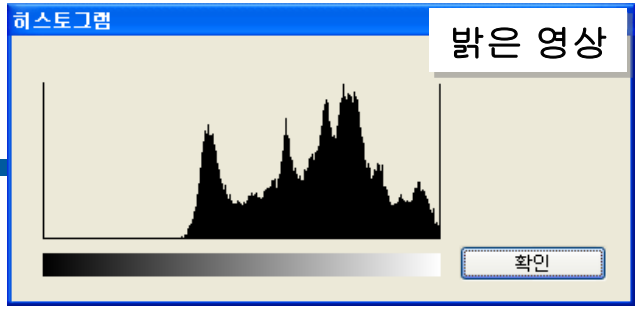
Histogram

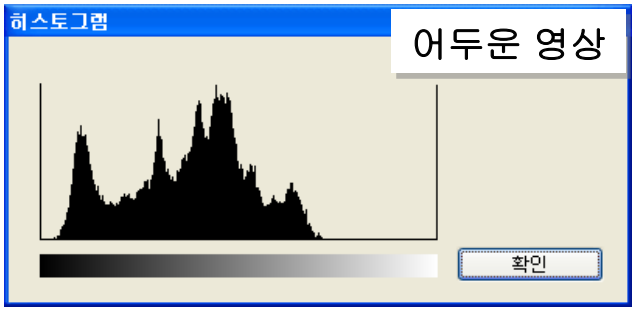
❖ 이미지의 밝기에 대한 분포도

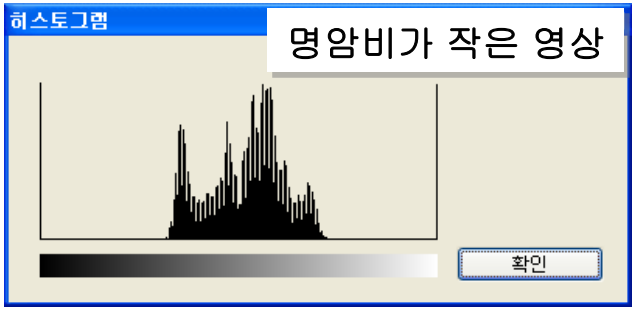


Histogram

❖ 다양한 영상에 대한 히스토그램 분석







Histogram

❖ Histogram Calculation in OpenCV

- cv.calcHist(images, channels, mask, histSize, ranges)
  - channels :이미지의 칼라 채널 ([0]:B, [1]:G, [2]:R)
  - mask : 특정 영역에 대한 막대 그래프를 찾으려면 해당 마스크 이미지를 만들어 마스크로 제공
  - histSize : BIN 개수
  - ranges : [0,256].

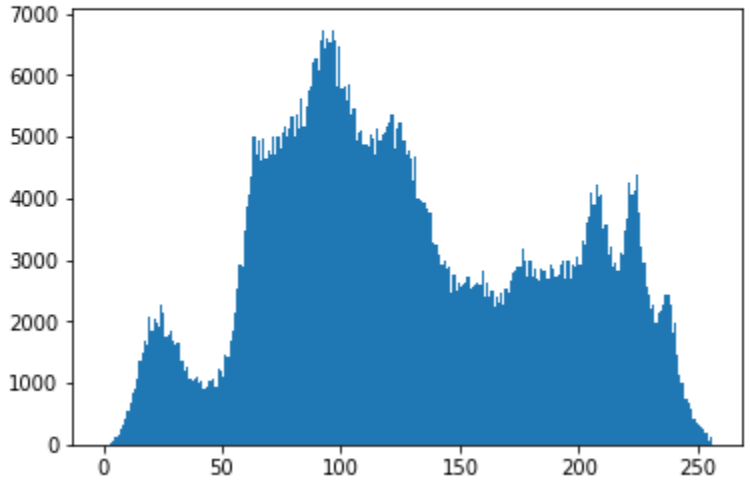
❖ Histogram Calculation in Numpy

- np.histogram(images, histSize, ranges)

Histogram

❖ Plotting Histograms

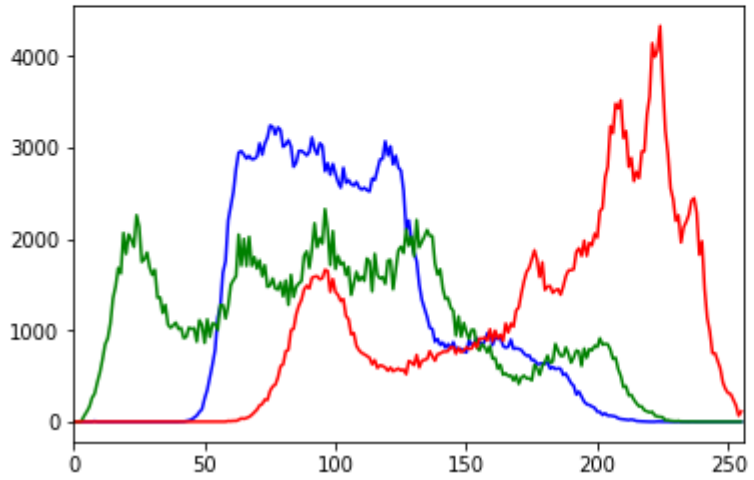
```
img = cv2.imread('images/lenna.bmp')
plt.hist(img.ravel(),256,[0,256])
plt.show()
```



Histogram

❖ Histogram Calculation in OpenCV

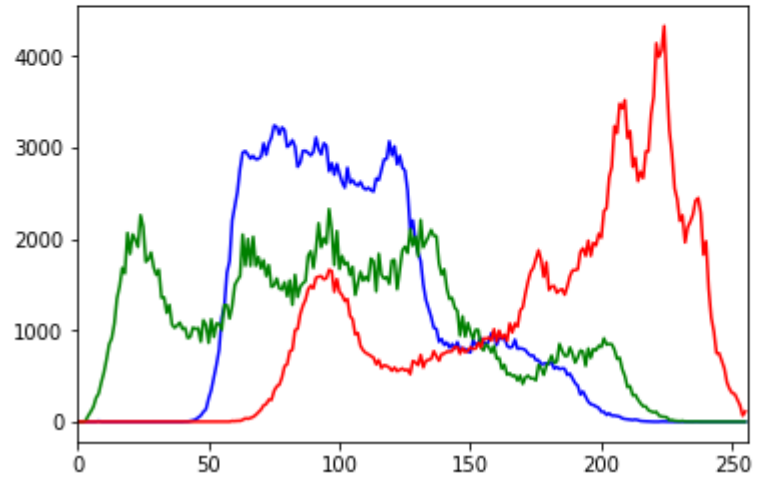
```
#채널별 히스토그램 계산하여 출력
#opencv 함수로 히스토그램 계산 (cv.calcHist)
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256]) #채널별 히스토그램 계산
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```



Histogram

❖ Histogram Calculation in Numpy

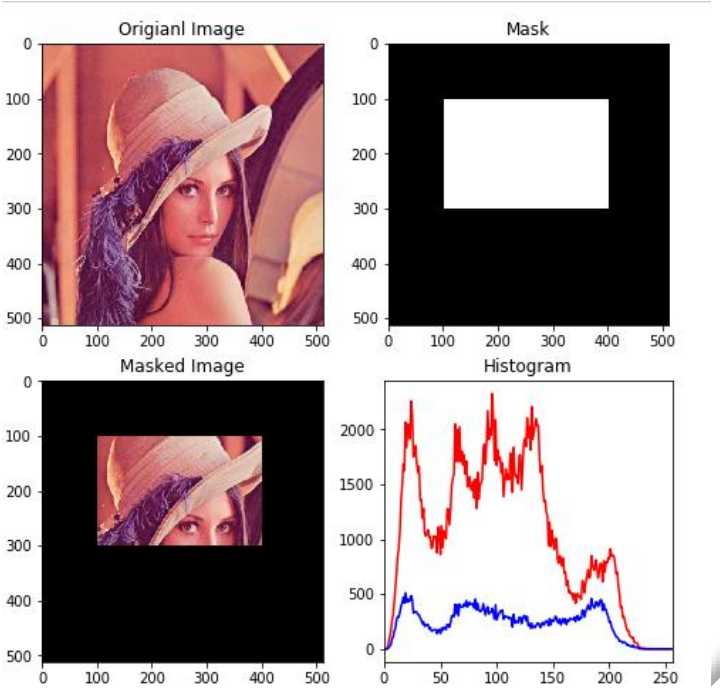
```
#numpy 함수로  채널별 히스토그램 계산하여 출력
#이미지 채널 분리
bi,gi,ri = cv2.split(img)
img = [bi,gi,ri]
for i,col in enumerate(color):
    hist, bins = np.histogram (img[i].ravel (), 256, [0,256])
    plt.plot(hist,color = col)
    plt.xlim([0,256])
plt.show()
```



Histogram

❖ Mask를 적용한 Histogram

```
img = cv2.imread('images/lenna.bmp');
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# mask생성
mask = np.zeros(img.shape[:2],np.uint8)
mask[100:300,100:400] = 255

# 이미지에 mask가 적용된 결과
masked_img = cv2.bitwise_and(img,img,mask=mask)

# 원본 이미지의 히스토그램
hist_full = cv2.calcHist([img],[1],None,[256],[0,256])

# mask를 적용한 히스트로그램
hist_mask = cv2.calcHist([img],[1],mask,[256],[0,256])

plt.figure(figsize=(8,8))

plt.subplot(221),plt.imshow(img,'gray'),plt.title('Origianl Image')
plt.subplot(222),plt.imshow(mask,'gray'),plt.title('Mask')
plt.subplot(223),plt.imshow(masked_img,'gray'),plt.title('Masked Image')

# red는 원본이미지 히스토그램, blue는 mask적용된 히스토그램
plt.subplot(224),plt.title('Histogram')
plt.plot(hist_full,color='r'),plt.plot(hist_mask,color='b')
plt.xlim([0,256])

plt.show()
```



QUIZ!

1. 'cat.png'를 4배 확대하여 180도 회전처리 결과를 출력하세요.
2. 'rice.bmp' 이미지에서 쌀알이 가장 잘 보이도록 thresholding 처리를 해보세요.
3. 'opencv-logo.png' 이미지를 채널로 분리하여 채널 별 히스토그램을 확인하세요.