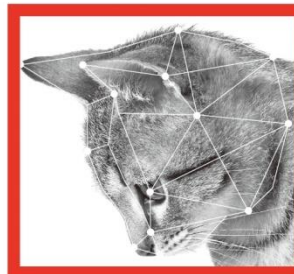


COMPUTER VISION



DEEP
LEARNING



컴퓨터 비전과 딥러닝

Chapter 03 영상 처리

차례

3.1 디지털 영상 기초

3.2 이진 영상

3.3 점 연산

3.4 영역 연산

3.5 기하 연산

3.6 OpenCV의 계산 효율

PREVIEW

■ 영상 처리

- 주어진 목적을 달성하기 위해 원래 영상을 새로운 영상으로 변환
- 컴퓨터 비전의 전처리 과정



(a) 원래 영상



(b) 어둡게



(c) 블러링



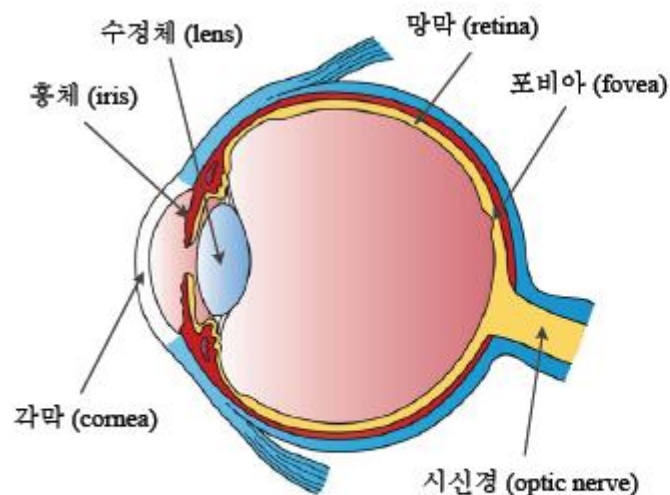
(d) 회전

그림 2-1 원래 영상과 영상 처리를 적용한 영상

획득과 표현

■ 사람의 눈과 카메라

- 수정체는 렌즈, 망막은 CCD 센서 (필름)에 해당



(a) 사람의 눈 구조



(b) 카메라의 구조

그림 2-3 사람의 눈과 카메라의 구조

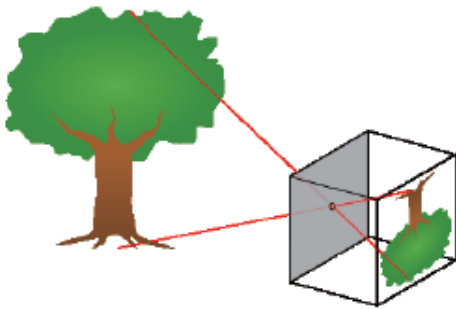
CCD 센서: 빛을 전기적 신호로 바꿔주는 광센서(optical sensor) 반도체 (semiconductor)임. 빛은 CCD에 붙어있는 RGB 색필터에 의해 각기 다른 색(色)으로 분리된 후 ADC(Analog Digital Converter)라는 변환 장치를 통해 디지털 신호로 변환된 후 이미지 파일로 저장

획득과 표현

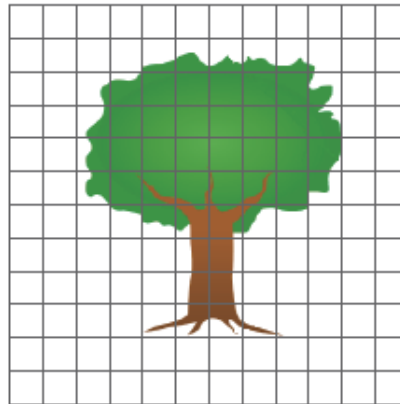
■ 샘플링과 양자화

CCD 소자의 수

- 2차원 영상 공간을 $M \times N$ 개의 점으로 샘플링 ($M \times N$ 을 해상도, 각 점은 화소(pixel)라 부름)
- 화소의 명암을 L 단계로 양자화 (L 을 명암 단계라 부름, 즉 명암은 $[0, L-1]$ 사이 분포)
- 아래 예) $M=12, N=12, L=10$ 인 경우(보통 화소에 1바이트 배정, $L=256$)



(a) 핀홀 카메라 모델



(b) 샘플링과 양자화

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	3	4	2	3	4	3	0	0	0
0	0	3	7	8	8	8	7	6	3	0	0
0	0	4	8	9	9	9	8	7	5	1	0
0	0	4	7	8	9	9	8	7	5	0	0
0	0	3	6	7	8	8	7	7	3	0	0
0	0	0	2	4	7	8	4	3	0	0	0
0	0	0	0	0	4	7	0	0	0	0	0
0	0	0	0	0	5	6	0	0	0	0	0
0	0	0	0	2	3	4	2	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(c) 디지털 영상

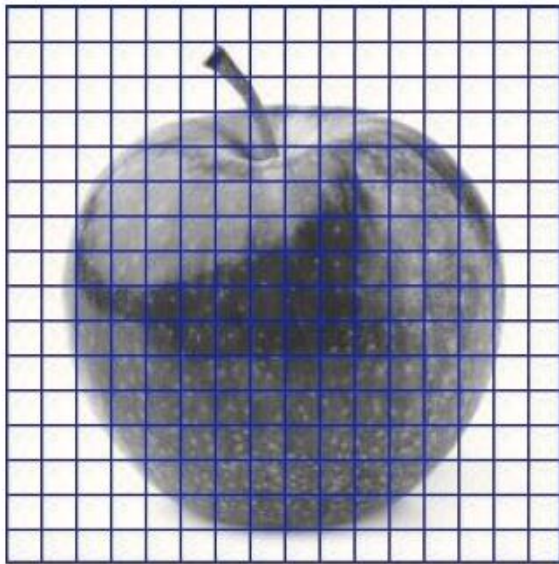
그림 2-4 디지털 영상 획득

pixel : picture element

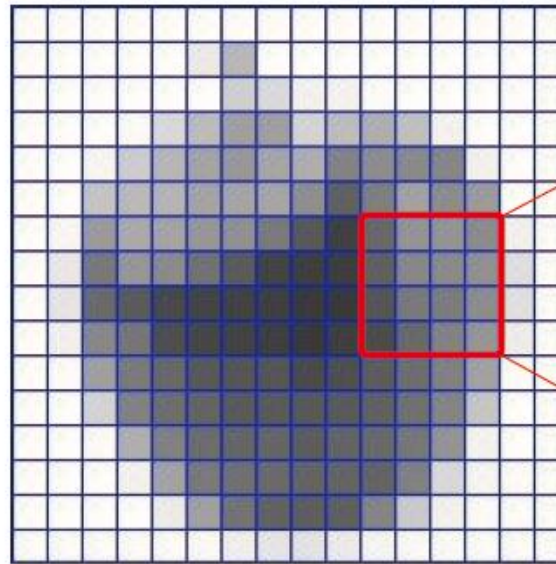
3.1.1 영상 획득과 표현

■ 디지털 변환

- $M \times N$ 영상으로 샘플링_{sampling}
- L 단계로 양자화_{quantization}
- $M=N=16, L=256$



(a) 샘플링



105	149	149	141
97	137	139	146
86	123	126	142
76	106	132	150

(b) 양자화

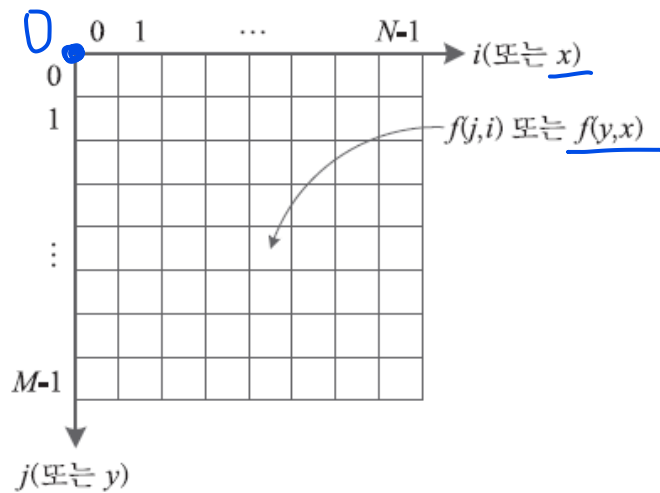
그림 3-3 피사체가 반사하는 빛 신호를 샘플링과 양자화를 통해 디지털 영상으로 변환

TIP 엄밀히 말해 해상도는 물리적 단위 공간에서 식별 가능한 점의 개수를 뜻한다. 예를 들어 인치 당 점의 개수를 뜻하는 dpi(dot per inch)는 해상도다. 이 책에서는 화소의 개수를 해상도라고 부른다.

획득과 표현

■ 영상 좌표계

- 화소 위치는 $\mathbf{x}=(j,i)$ 또는 $\mathbf{x}=(y,x)$ 로 표기
- 영상은 $f(\mathbf{x})$ 또는 $f(j,i)$, $0 \leq j \leq M-1$, $0 \leq i \leq N-1$ 로 표기
- OpenCV는 함수에 따라 (x,y) 표기 사용하니 주의할 필요. 예) `cv.line` 함수

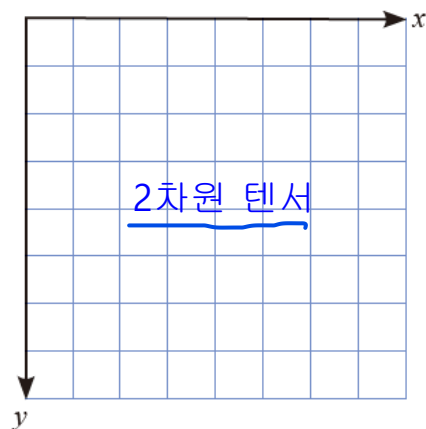


■ 컬러 영상은 한 화소가 $f_r(\mathbf{x})$, $f_g(\mathbf{x})$, $f_b(\mathbf{x})$ 의 세 채널로 구성

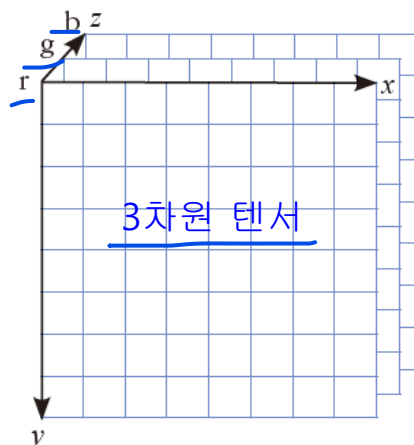
■ OpenCV는 `numpy.ndarray`로 영상 표현

- `numpy.ndarray`가 지원하는 다양한 함수를 사용할 수 있다는 큰 장점
- 예) `min`, `max`, `argmin`, `argmax`, `mean`, `sort`, `reshape`, `transpose`,

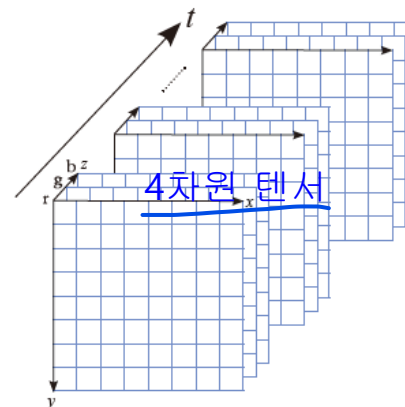
다양한 종류의 영상



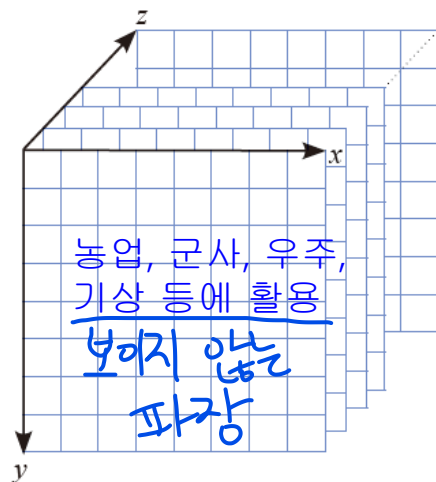
(a) 명암 영상



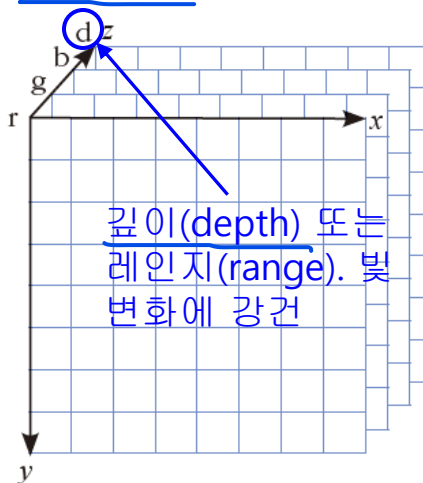
(b) 컬러 영상



(c) 컬러 동영상



(d) 다분광/초분광/MR/CT 영상



(e) RGB-D 영상



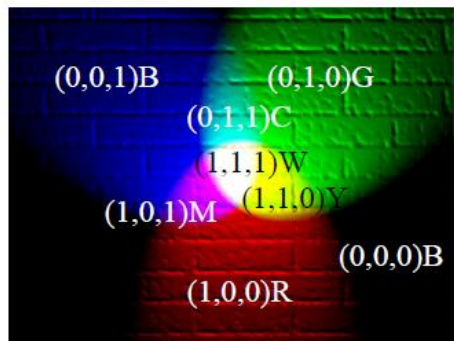
(f) 점 구름 영상

그림 3-5 다양한 형태의 디지털 영상

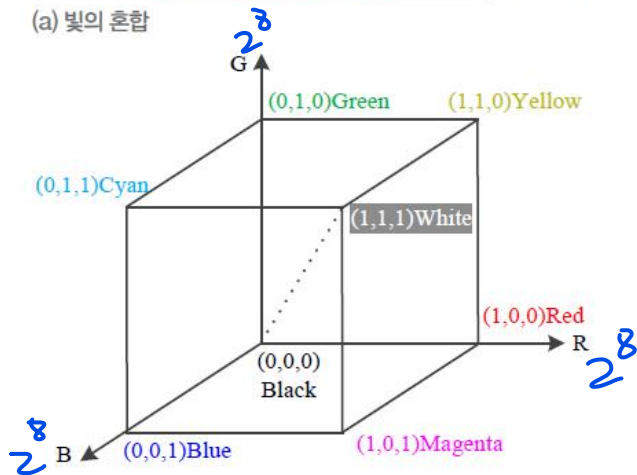
컬러

■ RGB 모델

- 길이가 1인 정육면체로 색을 표현

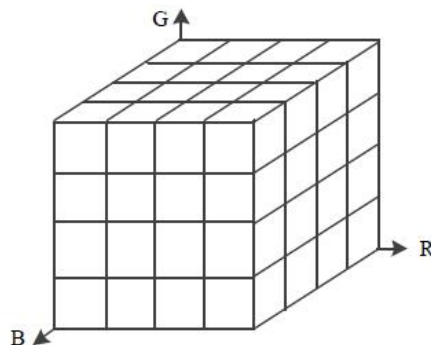


(a) 빛의 혼합

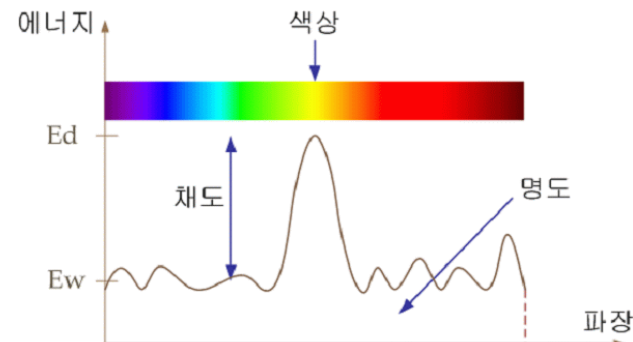
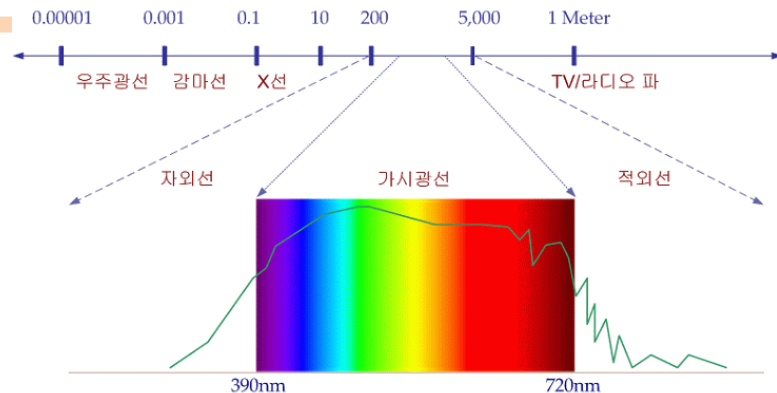


(b) RGB 큐브

그림 2-43 RGB 컬러 모델



(c) 양자화된 4×4×4 RGB 큐브

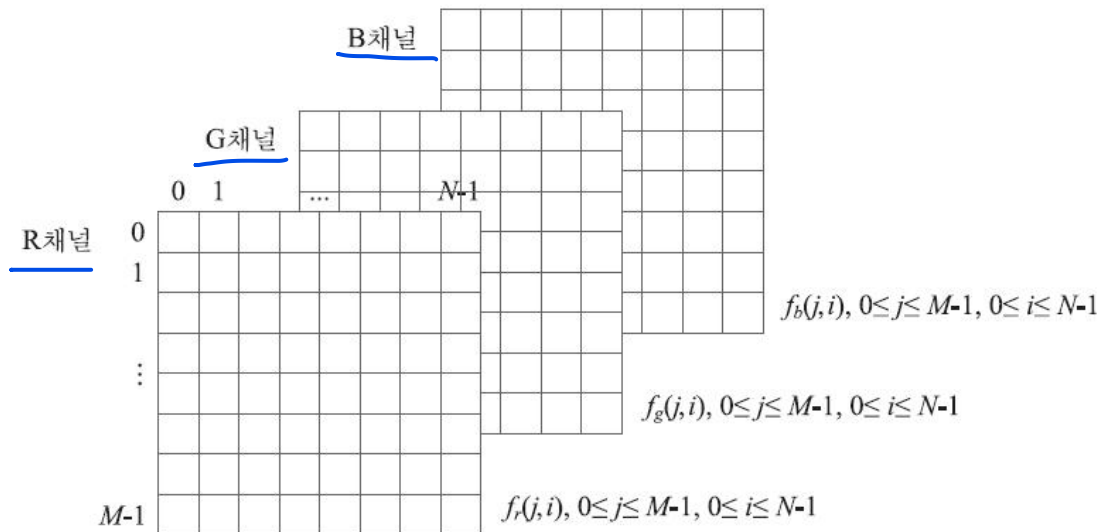


2^8
양자화는 256개의 구간으로 함

컬러

■ RGB 모델로 영상 표현

- f_r f_g f_b 의 세 채널로 표현



> RGB 영상(원래 영상)



> f_r



> f_g



> f_b

그림 2-44 RGB 컬러 영상

컬러

■ 컬러 영상 처리

- 가장 단순한 방법은 세 채널을 독립적으로 처리
- 예: 가우시안 스무딩
 - 각 채널에서 가우시안 스무딩 한 후 새로운 RGB 영상 생성

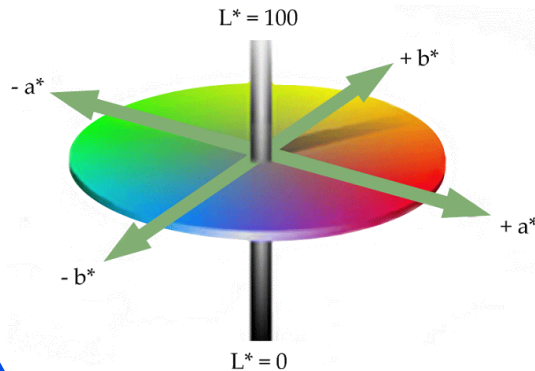
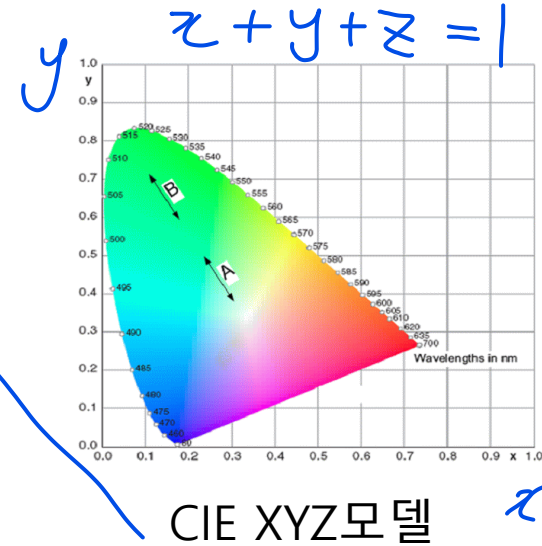


그림 2-46 RGB 영상에 가우시안 스무딩($\sigma=2.0$)을 적용한 결과

컬러 표현 변환

■ CIE L*a*b

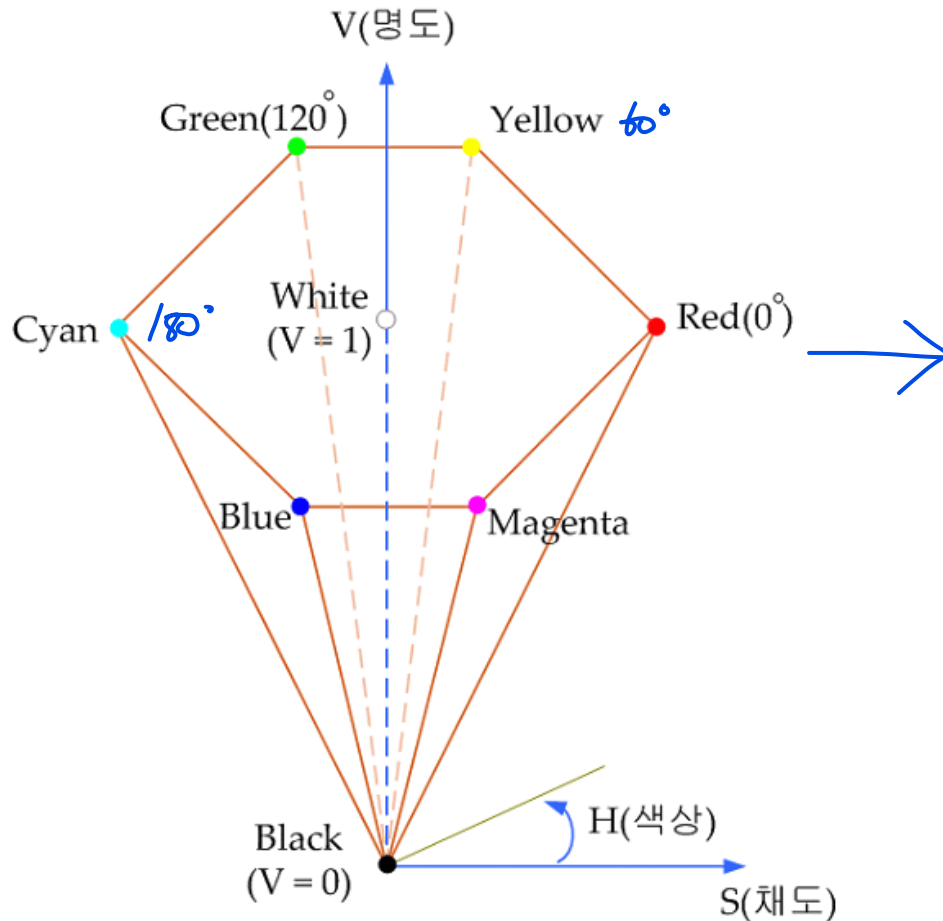
- RGB 공간 상에서 컬러 간의 거리
 - 두 컬러 간의 유사도 측정으로는 좋지 않음
 - 동일한 거리라도 서로 다른 컬러인 경우가 있음
- CIE의 변형
 - 인지 컬러모델(Perceptual Color Model)
 - 인지된 색차가 맵상의 거리에 비례하도록
 - 두 컬러 간의 거리
 - 두 컬러 간의 시각적 유사도를 반영하는 컬러 공간




CIE XYZ와 CIE L*a*b*와의 비교
(a) (b)

컬러 표현 변환

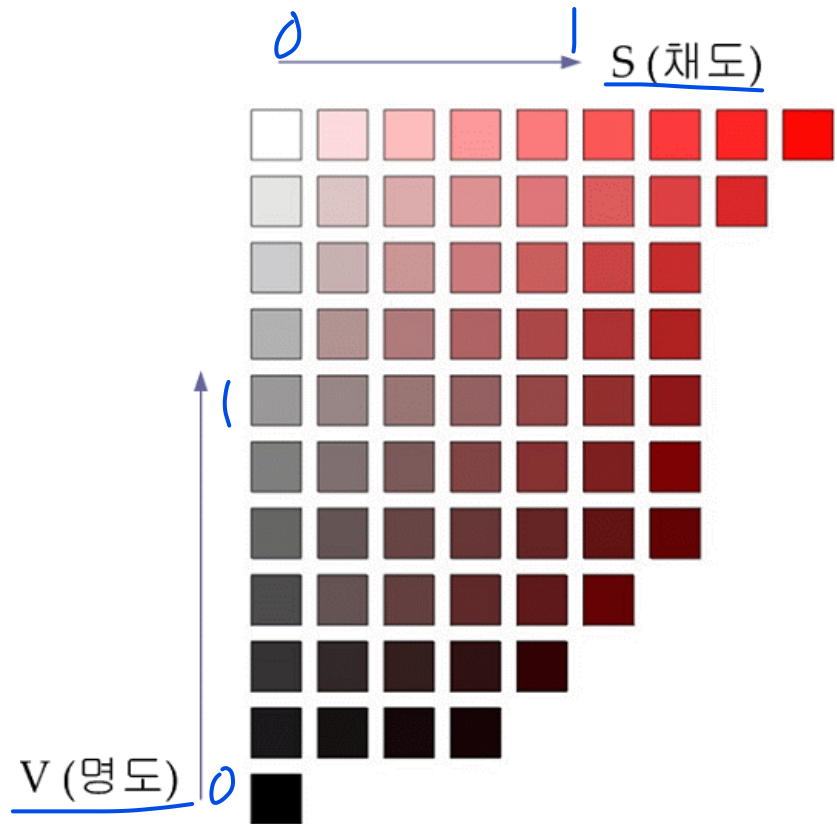
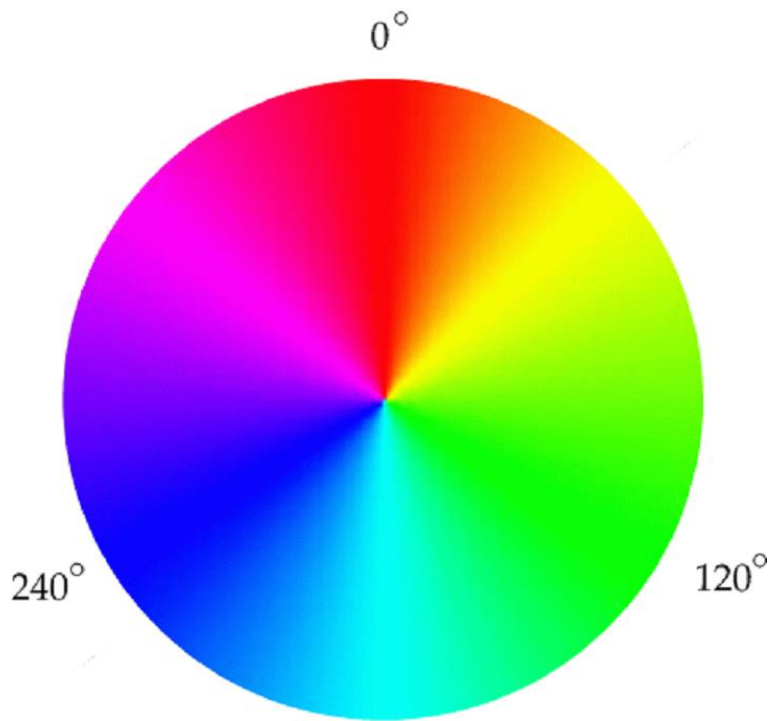
- HSV(Hue, Saturate, Value): 색상, 채도, 밝기
 - `cv::cvtColor(image, converted, COLOR_BGR2HSV);`



H	S	V	색
0	1.0	1.0	Red
120	1.0	1.0	Green
240	1.0	1.0	Blue
	0.0	1.0	White
		0.0	Black
90	0.5	0.25	

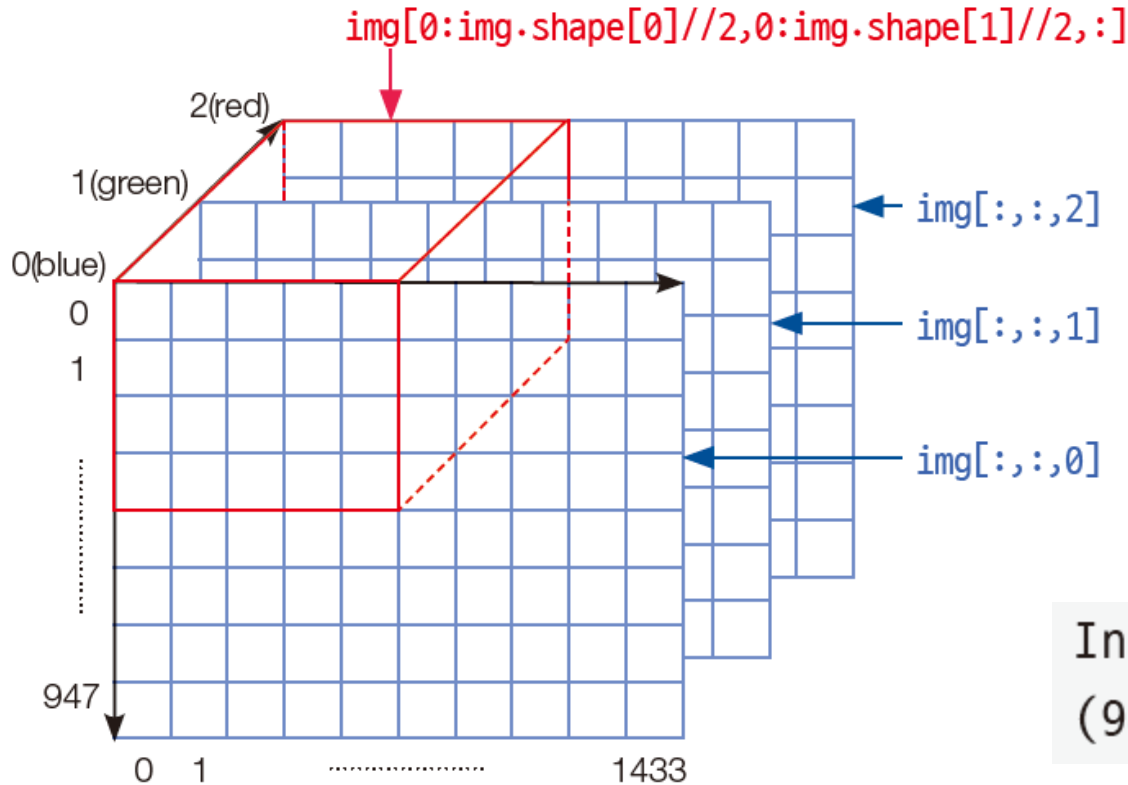
컬러 표현 변환

- ▶ HSV(Hue, Saturate, Value): 색상, 채도, 밝기



RGB 채널별로 디스플레이

■ numpy의 슬라이싱 기능을 이용하여 RGB 채널별로 디스플레이



```
In [0]: img.shape  
(948, 1434, 3)
```

그림 3-8 numpy.ndarray의 슬라이싱을 이용한 영상 일부분 자르기([프로그램 3-1]의 10행)

OpenCV의 가산 혼합의 삼원색 기본 배열순서는 **BGR**

TIP 온라인 부록 A에서 이 책을 공부하는 데 필요한 최소한의 numpy 지식을 제공한다.

RGB 채널별로 디스플레이

프로그램 3-1

RGB 컬러 영상을 채널별로 구분해 디스플레이하기

```
01 import cv2 as cv
02 import sys
03
04 img=cv.imread('soccer.jpg')
05
06 if img is None:
07     sys.exit('파일을 찾을 수 없습니다.')
08
09 cv.imshow('original_RGB',img)
10 cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
11 cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,img.
    shape[1]//4:3*img.shape[1]//4,:])
12
13 cv.imshow('R channel',img[:, :,2])
14 cv.imshow('G channel',img[:, :,1])
15 cv.imshow('B channel',img[:, :,0])
16
17 cv.waitKey()
18 cv.destroyAllWindows()
```

bgr=cv.split(img) #혹은 b, g, r
cv.imshow("B channel", bgr[0])
cv.imshow("G channel", bgr[1])
cv.imshow("R channel", bgr[2])
cv.waitKey()

◆ imread()

```
Mat cv::imread ( const String & filename,  
                int flags = IMREAD_COLOR  
                )
```

Python:

```
cv.imread( filename[, flags] ) -> retval
```

이미지 파일은 Numpy array 형태로 숫자

cv2.IMREAD_UNCHANGED or -1 : image 파일 변형 없이 원본 읽기

cv2.IMREAD_COLOR or 1 : BGR 색으로 읽기 (default)

cv2.IMREAD_GRAYSCALE or 0 : 회색으로 이미지 출력하기

cv2.IMREAD_REDUCED_GRAYSCALE_2 : 회색 출력, 사이즈 반으로 줄이기

cv2.IMREAD_REDUCED_COLOR_2 : BGR 출력, 사이즈 반으로 줄이기

cv2.IMREAD_REDUCED_GRAYSCALE_4 : 회색 출력, 사이즈 1/4로 줄이기

cv2.IMREAD_REDUCED_COLOR_4 : BGR 출력, 사이즈 1/4로 줄이기

cv2.IMREAD_ANYDEPTH : 8/16/32비트 변경

cv2.IMREAD_ANYCOLOR : 어떤 색으로든 출력 가능

cv2.IMREAD_LOAD_GDAL : gdal 드라이브로 이미지 읽기

cv2.IMREAD_IGNORE_ORIENTATION : EXIF flag에 따라 이미지 회전 하지 않음

In the case of color images, the decoded images will have the channels stored in B G R order.

컬러 표현 변환

◆ cvtColor()

```
void cv::cvtColor ( InputArray  src,  
                   OutputArray dst,  
                   int          code,  
                   int          dstCn = 0  
                 )
```

Python:

```
cv.cvtColor( src, code[, dst[, dstCn]] ) -> dst
```

- src: source image, dst: destination image,
 - 같은 크기, 같은 깊이, 채널은 다를 수 있음
- code: color space conversion code
- COLOR_src-color2dst-color
 - COLOR_BGR2GRAY, COLOR_BGR2YCrCb, COLOR_BGR2HSV, COLOR_BGR2Luv, COLOR_BGR2XYZ, COLOR_BGR2Lab반대로도 성립 COLOR_Lab2BGR...
- dstCn: dst 영상의 채널 수
 - 0 이면 src와 code에 의해 자동으로 결정
- 예제) `converted = cv2.cvtColor(image, cv2.COLOR_BGR2Lab);`

"soccer.jpg" 영상의 HSV채널 별로 출력

```
import cv2 as cv
import sys
import numpy as np

img=cv.imread('soccer.jpg')
if img is None:
    sys.exit("No File exists.")

cv.imshow("Original", img); cv.waitKey()
#%%
hsv=cv.cvtColor(img, cv.COLOR_BGR2HSV)
cv.imshow('Hsv', hsv) ; cv.waitKey()
#%%
h, s, v = cv.split(hsv)
type(h)
cv.imshow('Hue', h) ; cv.waitKey()
#np.set_printoptions(threshold=np.inf, linewidth=np.inf)
#print(h)
print(np.max(h))

cv.imshow('Sat', s) ; cv.waitKey()
cv.imshow('Value', v) ; cv.waitKey()
```

채널 별 작업 후 합병

■ 명도 channel의 값을 모두 255로 변환 후 채널 합병

```
print(v.ndim)
print(v.shape)
print(v)
v=np.zeros((948, 1434), np.uint8)+255
dst=cv.merge([h,s,v])
new=cv.cvtColor(dst,cv.COLOR_HSV2BGR)
cv.imshow('New', new)
cv.waitKey()
cv.destroyAllWindows()
```

◆ merge() [2/2]

```
void cv::merge ( InputArrayOfArrays mv,
                 OutputArray      dst
                )
```

Python:

```
cv.merge( mv[, dst] ) -> dst
```

- void merge(InputArrayOfArrays **mv**, OutputArray **dst**)
 - **mv** – input array or vector of matrices to be merged; all the matrices in mv must have the same size and the same depth.
 - **dst** – output array of the same size and the same depth as mv[0]; The number of channels will be the total number of channels in the matrix array

HSV 채널 분리

■ 어두운 영역의 채도 값은 신뢰할 수 없다

- boldt.jpg 채도 영상에서 지붕의 한쪽 부분, 물과 경계 부분이 하얗게 보임
- (R,G,B) = (0.5,0,0), (1,0,0) 모두 채도=1
- $S = \frac{\max(R,G,B) - \min(R,G,B)}{\max(R,G,B)}$
채도

■ 낮은 채도를 갖는 컬러를 평가할 때 색상(Hue)에 대한 신뢰도가 떨어짐

- OpenCV는 hue 값을 8비트에 표현하기 위해 0~360의 범위를 2로 나누어 0~180 범위로 조정하였음

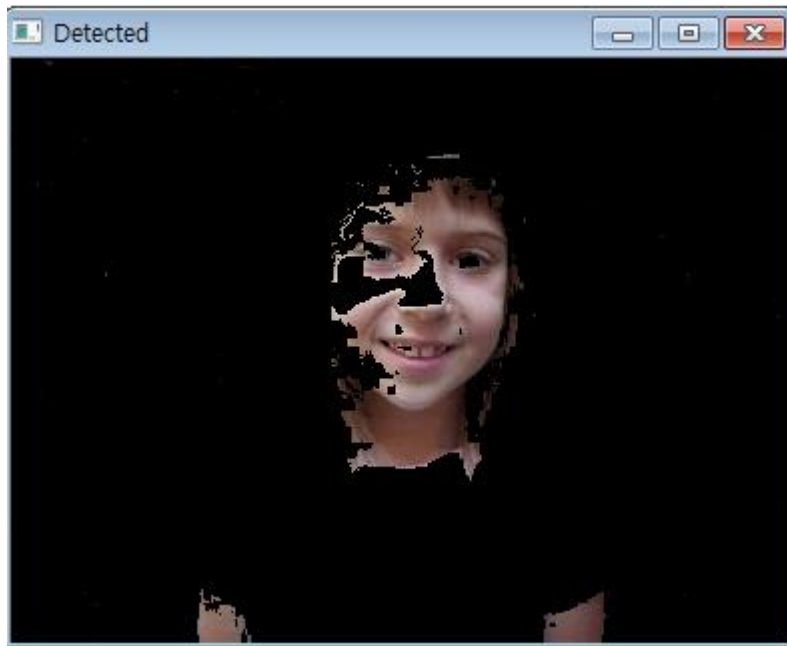
■ BALLOON.bmp를 이용해서 HSV 채널 분리해 볼 것

지정한 크기로 새 영상 생성하기

- numpy 명령어를 이용해 배열을 생성하여 영상으로 이용.
 - `numpy.empty(shape, dtype=np.uint8, ...)` -> arr : 임의의 값으로 초기화된 배열을 생성
 - `numpy.zeros(shape, dtype=np.uint8, ...)` -> arr : 0으로 초기화된 배열을 생성
 - `numpy.ones(shape, dtype=np.uint8, ...)` -> arr : 1로 초기화된 배열을 생성
 - `numpy.full(shape, fill_value, dtype=np.uint8, ...)` -> arr : fill_value로 초기화된 배열을 생성
- parameters
 - shape : 각 차원의 크기. (h,w) 또는 (h,w,3)
 - dtype : 원소의 데이터 타입. 일반적인 영상이면 `numpy.uint8` 지정
 - arr : 생성된 영상(`numpy.ndarray`)
- 예제:
 - `img1 = np.empty((240, 320), dtype=np.uint8)` # *grayscale image 임의의 value*
 - `img2 = np.zeros((240, 320, 3), dtype=np.uint8)` # *color image 모든 픽셀이 0*
 - `img3 = np.ones((240, 320), dtype=np.uint8) * 128` # *dark gray 모든 픽셀이 1 * 128*
 - `img4 = np.full((240, 320, 3), (0, 255, 255), dtype=np.uint8)` # *yellow 픽셀을 지정*

피부색 검출 : 색상과 채도 사용

- 컬러 정보는 특정 객체의 초기 검출에 유용



```
cv2.inRange(src, lowerb, upperb, dst=None) -> dst
```

OpenCV에서 제공하는 cv2.inRange 함수를 사용하여 특정 색상 영역을 추출

- src: 입력 행렬
- lowerb: 하한 값 행렬 또는 스칼라
- upperb: 상한 값 행렬 또는 스칼라
- dst: 입력 영상과 같은 크기의 마스크 영상. (numpy.uint8) 범위 안에 들어가는 픽셀은 255, 나머지는 0으로 설정

피부색 검출 : 색상과 채도 사용

```
import cv2 as cv
import sys
import numpy as np
#%%%
img=cv.imread('girl.jpg') ; type(img)
if img is None:
    sys.exit("No File exists.")
cv.imshow("Original", img) ; cv.waitKey()
#%%%
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
mask1 = cv.inRange(hsv, (0, 25, 0),(10,166,255))
cv.imshow("mask1", mask1)
mask2 = cv.inRange(hsv, (160,25, 0),(180,166,255))
cv.imshow("mask2", mask2)
mask = mask1 | mask2
cv.imshow("Mask", mask) ; cv.waitKey()
#%%%
detected = cv.bitwise_and(img, img, mask=mask)
cv.imshow("Result", detected); cv.waitKey()
cv.destroyAllWindows()
```


과제

- 다음 영상에서 파란색 풍선을 찾아서 파란색 영역만 출력하시오(BALLOON.bmp). 결과 영상의 배경은 흰색으로하시오



3.2 이진화와 오츠크 알고리즘

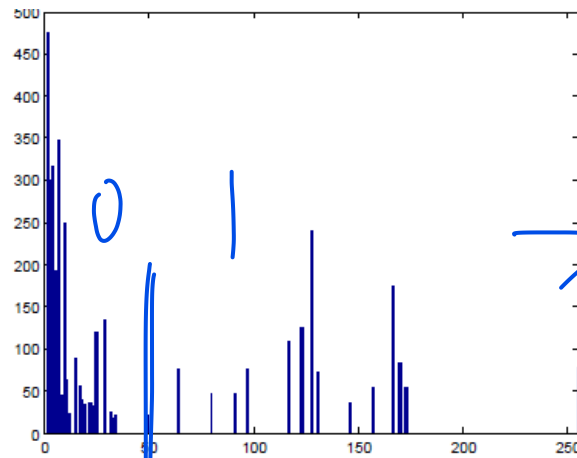
■ 이진화

- 명암 영상을 흑과 백만 가진 이진 영상으로 변환

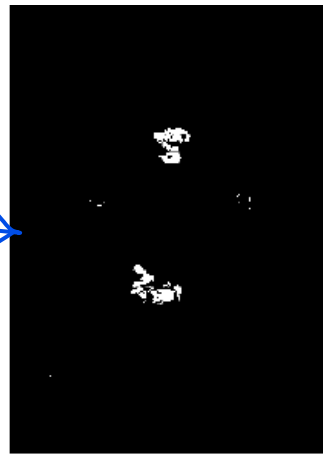
$$b(j, i) = \begin{cases} 1, & \text{if } f(j, i) \geq T \\ 0, & f(j, i) < T \end{cases} \quad (2.5)$$

■ 임계값 방법

- 히스토그램을 분석하여 두 봉우리 사이의 계곡을 임계값 T 로 결정
- 자연 영상에서는 계곡 지점 결정이 어려움
- 컴퓨터비전에서는 임계값을 자동으로 처리해야 함



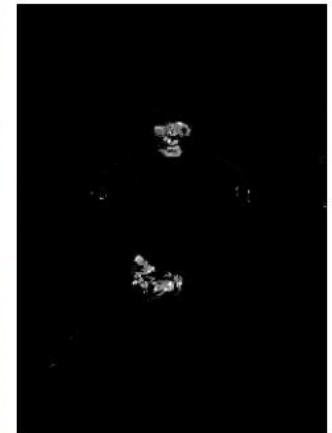
(a) 히스토그램
그림 2-14 이진화



(b) 임계값을 이용하여 구한 이진 영상($T=50$)



(a) 입력 영상



(b) 역투영 영상

그림 2-13 히스토그램 역투영을 이용한 얼굴 검출

그림 2-13 (b)의 영상에서 얼굴위치를 찾기 위해 이진화 한 후 연결요소를 찾고 연결요소의 중심을 얼굴위치로 선정

3.2.1 이진화

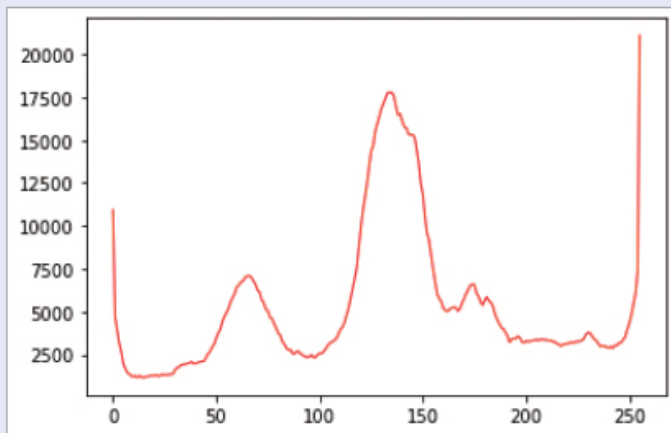
■ 알고리즘 (계속.....)

- 실제 영상에서는 계곡이 아주 많이 나타나서 구현이 쉽지 않음

프로그램 3-2

실제 영상에서 히스토그램 구하기

```
01 import cv2 as cv
02 import matplotlib.pyplot as plt
03
04 img=cv.imread('soccer.jpg')
05 h=cv.calcHist([img],[2],None,[256],[0,256]) # 2번 채널인 R 채널에서 히스토그램 구함
06 plt.plot(h,color='r',linewidth=1)
```



◆ calcHist() [3 / 3]

```
void cv::calcHist ( InputArrayOfArrays      images,  
                   const std::vector< int > & channels,  
                   InputArray              mask,  
                   OutputArray            hist,  
                   const std::vector< int > & histSize,  
                   const std::vector< float > & ranges,  
                   bool                    accumulate = false  
                   )
```

Python:

```
cv.calcHist( images, channels, mask, histSize, ranges[, hist[, accumulate]] ) -> hist
```

- images: 입력 영상 리스트
- channels: 히스토그램을 구할 채널을 나타내는 리스트
- mask: 마스크 영상. 입력 영상 전체에서 히스토그램을 구하려면 None 지정
- histSize: 히스토그램 각 차원의 크기(빈(bin)의 개수)를 나타내는 리스트
- ranges: 히스토그램 각 차원의 최솟값과 최댓값으로 구성된 리스트
- hist: 계산된 히스토그램 (numpy.ndarray)
- accumulate: 기존의 hist 히스토그램에 누적하려면 True, 새로 만들려면 False

3.2.2 오츠크 알고리즘

■ 오츠크 알고리즘 [Otsu79]

- 이진화 했을 때 흑 그룹과 백 그룹 각각이 그룹내에서 균일할수록 좋다는 원리에 근거
- 균일성은 분산으로 측정 (분산이 작을수록 균일성 높음: 명암의 분포가 뭉쳐 있음)
- 분산의 가중치 합 $v_{within}(\cdot)$ 을 목적 함수로 이용한 최적화 알고리즘
- 시간복잡도: $O(L^2)$ - 실시간 적용에 부담=>효율적 버전 사용

$$T = \underset{t \in \{0, 1, \dots, L-1\}}{\operatorname{argmin}} \frac{v_{within}(t)}{\quad} \text{이 값이 최소가 되게} \quad (2.6)$$

할 때 T가 임계값

$$\begin{aligned} w_0(t) &= \sum_{i=0}^t \hat{h}(i), & w_1(t) &= \sum_{i=t+1}^{L-1} \hat{h}(i) \\ \mu_0(t) &= \frac{1}{w_0(t)} \sum_{i=0}^t i \hat{h}(i), & \mu_1(t) &= \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} i \hat{h}(i) \\ v_0(t) &= \frac{1}{w_0(t)} \sum_{i=0}^t \hat{h}(i) (i - \mu_0(t))^2, & v_1(t) &= \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} \hat{h}(i) (i - \mu_1(t))^2 \end{aligned} \quad (2.7)$$

3.2.2 오츠크 알고리즘

■ 프로그래밍 실습

프로그램 3-3

오츠크 알고리즘으로 이진화하기

```
01  import cv2 as cv
02  import sys
03
04  img=cv.imread('soccer.jpg')
05
06  t,bin_img=cv.threshold(img[:,:,:2],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
07  print('오츠크 알고리즘이 찾은 최적 임계값=',t) ①
08
09  cv.imshow('R channel',img[:,:,:2])                # R 채널 영상
10  cv.imshow('R channel binarization',bin_img)        # R 채널 이진화 영상
11
12  cv.waitKey()
13  cv.destroyAllWindows()
```

threshold()

◆ threshold()

```
double cv::threshold ( InputArray    src,  
                      OutputArray dst,  
                      double          thresh,  
                      double          maxval,  
                      int              type  
                      )
```

Python:

```
cv.threshold( src, thresh, maxval, type[, dst] ) -> retval, dst
```

retval=임계값

■ 영상의 이진화를 처리.

- 임계값(threshold)을 받아 픽셀 값 이진화
- 임계값 이하면 0으로, 임계값 이상이면 maxVal로 처리해준다.

■ Parameters

- src: 원본 이미지.
- dst: 결과 출력 이미지.
- threshold: 임계값, retval
- maxVal: 임계값보다 큰 픽셀 값을 어떤 값으로 설정할 것인지 정함.
- thresholdType: 이진화 방식을 결정.
 - cv2.THRESH_BINARY: 임계값 이하 = 0, 임계값 초과 = maxVal.
 - cv2.THRESH_BINARY_INV: 임계값 이하 = maxVal, 임계값 초과 = 0.
 - cv2.THRESH_TRUNC: 임계값 이하 = 그대로, 임계값 초과 = threshold.
 - cv2.THRESH_TOZERO: 임계값 이하 = 0, 임계값 초과 = src(x,y).
 - cv2.THRESH_TOZERO_INV: 임계값 이하 = src(x,y), 임계값 초과 = 0.

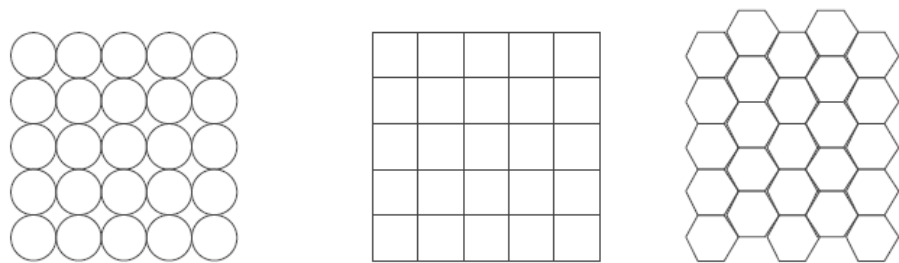
3.2.2 오츠크 알고리즘

오츠크 알고리즘이 찾은 최적 임계값= 113.0 ①



3.3.2 연결요소(connected component) 화소의 이웃들의 집합

■ 화소의 모양과 연결성

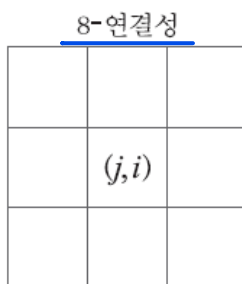
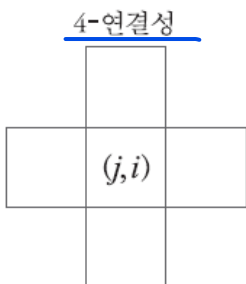


(a) 생각해 볼 수 있는 화소의 여러 가지 모양

$i-1$	i	$i+1$	
NW	N	NE	$j-1$
W	(j,i)	E	j
SW	S	SE	$j+1$

(b) 화소의 연결성

그림 2-16 화소의 모양과 연결성



연결요소 번호 붙이기

0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	1	0	1	1	0
0	1	0	1	0	1	1	0	1	0
0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	0	1	0
0	1	1	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0

(a) 입력 이진 영상

0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	2	2	0	0	1	0	3	3	0
0	2	0	4	0	1	1	0	3	0
0	2	0	4	0	1	0	0	3	0
0	2	0	4	0	1	0	0	3	0
0	2	2	0	0	1	0	0	3	0
0	0	0	0	0	0	0	0	0	0

(b) 번호 붙이기(4-연결성)

그림 2-17 연결요소 번호 붙이기

0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	2	2	0	0	1	0	1	1	0
0	2	0	2	0	1	1	0	1	0
0	2	0	2	0	1	0	0	1	0
0	2	0	2	0	1	0	0	1	0
0	2	2	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0

(c) 번호 붙이기(8-연결성)

4-연결성: 4개 연결요소
8-연결성: 2개 연결요소

OpenCV에서는 connectedComponents를 사용해서 찾을 수 있다.