

파이썬을 활용한 컴퓨터 비전 입문

Chapter 10. 컬러 영상 처리

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의 『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

❖ 10장 컬러 영상 처리

- 10.1 컬러 영상 다루기
- 10.2 컬러 영상 처리 기법

10.1 컬러 영상 다루기

10.2 컬러 영상 처리 기법

❖ 컬러 영상의 픽셀 값 참조 (1/7)

- OpenCV에서 영상 파일을 불러와서 numpy.ndarray 객체를 생성할 때에는 imread() 함수를 사용함
- 이때 imread() 함수의 두 번째 인자를 IMREAD_COLOR로 설정하면 영상을 3채널 컬러 영상 형식으로 불러옴
- 예를 들어 butterfly.jpg 나비 영상을 3채널 컬러 영상 형식으로 불러오려면 다음과 같이 코드를 작성함

```
img = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)
```

❖ 컬러 영상의 픽셀 값 참조 (2/7)

- 일반적으로 컬러 영상은 흔히 RGB라고 부르는 빨간색(R), 녹색(G), 파란색(B) 색상 성분의 조합으로 픽셀 값을 표현함
- OpenCV의 컬러 영상은 기본적으로 **RGB** 색상 순서가 아니라 **BGR** 색상 순서로 픽셀 값을 표현함
- `imread()` 함수로 영상을 3채널 컬러 영상 형식으로 불러오면 각 픽셀의 색상 값이 파란색(B), 녹색(G), 빨간색(R) 순서로 저장된 `numpy.ndarray` 객체가 생성됨

❖ 컬러 영상의 픽셀 값 참조 (3/7)

- 컬러 영상에서 각각의 R, G, B 색상 성분은 0부터 255 사이의 값을 가질 수 있음
- 색상 성분 값이 0이면 해당 색상 성분이 전혀 없는 상태임
- 255이면 해당 색상 성분이 가득 차 있음을 의미함
- OpenCV에서 각 색상 성분 값은 uint8 자료형을 사용하여 표현함
- 컬러 영상에서 하나의 픽셀은 세 개의 색상 성분을 가지고 있음
- 파이썬 인덱싱(indexing)을 사용하면 세 개의 색상 성분 값에 접근할 수 있음

❖ 컬러 영상의 픽셀 값 참조 (4/7)

- 코드 10-1에 나타난 `color_inverse()` 함수는 butterfly.jpg 컬러 영상의 모든 픽셀 값을 반전시켜 화면에 나타냄
- 컬러 영상을 반전하려면 B, G, R 세 개의 색상 성분 값을 각각 255에서 빼는 연산을 수행해야 함

코드 10-1 컬러 영상의 픽셀 값 반전 (ColorOp.py)

```
1  import numpy as np
2  import cv2
3
4  def color_op():
5      src = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)
6
7      if src is None:
8          print('Image load failed!')
9          return
10
11     print('src.shape:', src.shape)
12     print('src.dtype:', src.dtype)
13
14     # b, g, r = src[0, 0, :]
15     print('The pixel value [B, G, R] at (0, 0) is', src[0, 0, :])
16
```

❖ 컬러 영상의 픽셀 값 참조 (5/7)

코드 10-1 컬러 영상의 픽셀 값 반전 (ColorOp.py)

```
17 def color_inverse():
18     src = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)
19
20     if src is None:
21         print('Image load failed!')
22         return
23
24     dst = np.zeros(src.shape, src.dtype)
25
26     for j in range(src.shape[0]):
27         for k in range(src.shape[1]):
28             p1 = src[j, k, :]
29             p2 = dst[j, k, :]
30
31             p2[0] = 255 - p1[0]
32             p2[1] = 255 - p1[1]
33             p2[2] = 255 - p1[2]
34
35     cv2.imshow('src', src)
36     cv2.imshow('dst', dst)
37     cv2.waitKey()
38     cv2.destroyAllWindows()
39
```


❖ 컬러 영상의 픽셀 값 참조 (6/7)

코드 10-1 컬러 영상의 픽셀 값 반전 (ColorOp.py)

```
40 if __name__ == '__main__':  
41     color_op()  
42     color_inverse()
```

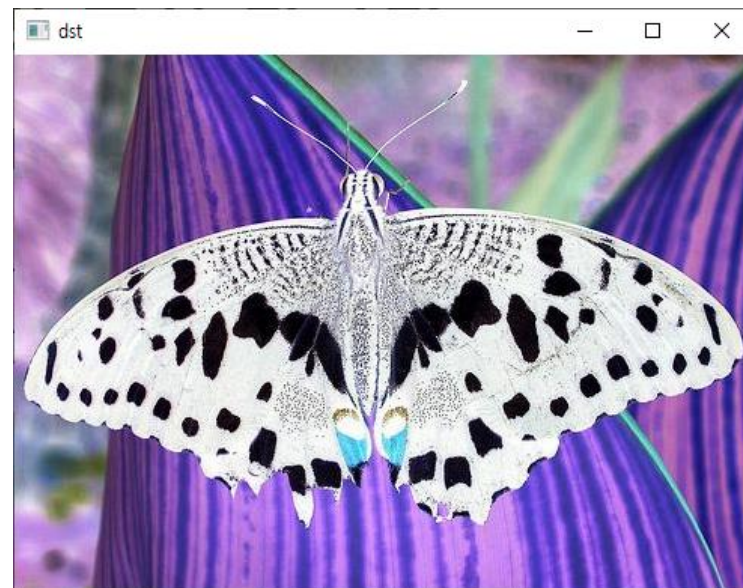
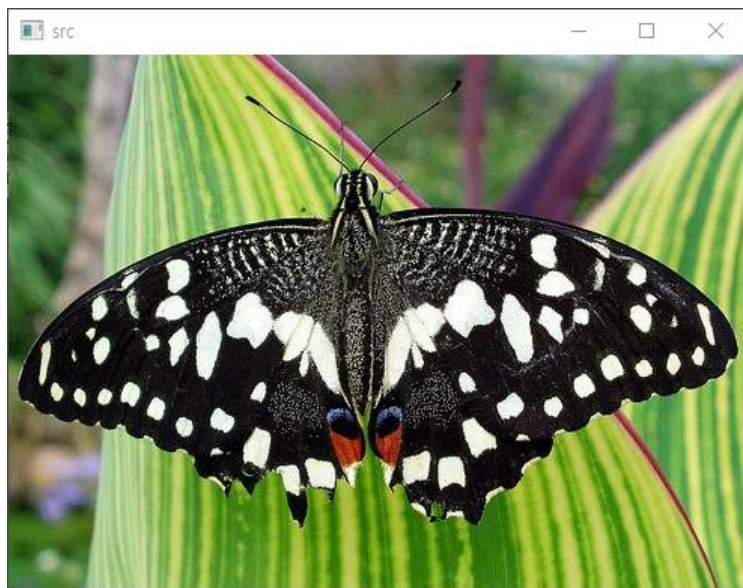
● ColorOp.py 소스 코드 설명

- 18행 butterfly.jpg 파일을 3채널 BGR 컬러 영상으로 불러와서 src에 저장합니다.
- 24행 반전된 영상을 저장할 dst 영상을 생성합니다.
dst 영상의 모든 픽셀 값은 이후 for 반복문에서 설정할 것이므로 초깃값은 따로 지정하지 않습니다.
- 28~29행 src와 dst 영상의 (j, k) 좌표 픽셀 값을 각각 p1과 p2 변수에 참조로 받아 옵니다.
- 31~33행 p1 픽셀의 세 개 색상 성분 값을 모두 반전시켜 p2 픽셀 값으로 설정합니다.

❖ 컬러 영상의 픽셀 값 참조 (7/7)

- 그림 10-1에서 src는 입력 영상 butterfly.jpg 파일이고, dst는 반전된 영상임
- 입력 영상 src에서 검은색 나비 날개 부분이 반전되어 흰색으로 변경된 것을 확인할 수 있음
- 입력 영상에서 배경의 녹색 나뭇잎 영역은 B, G, R 각 채널이 각각 반전되어 결과 영상에서는 보라색으로 변경됨

▼ 그림 10-1 컬러 영상의 반전



❖ 색 공간 변환 (1/4)

- 빨간색, 녹색, 파란색 세 가지 색 성분의 조합으로 색을 표현하는 방식을 **RGB 색 모델(color model)** 또는 **RGB 색 공간(color space)** 표현이라고 함
- 컬러 영상 처리에서는 보통 색상 구분이 용이한 HSV, HSL 색 공간을 사용함
- 휘도 성분이 구분되어 있는 YCrCb, YUV 등 색 공간을 사용하는 것이 유리함
- OpenCV는 **BGR** 순서로 색상이 저장된 컬러 영상의 색 공간을 HSV, YCrCB 등 **다른 색 공간으로 변환하는 인터페이스를 제공함**

❖ 색 공간 변환 (2/4)

- OpenCV에서 영상의 색 공간을 다른 색 공간으로 변환할 때에는 `cvtColor()` 함수를 사용함

```
dst = cv2.cvtColor(src, code)
```

src 입력 영상

dst 결과 영상. src와 크기 및 깊이가 같습니다.

code 색 공간 변환 코드.

ColorConversionCodes 열거형 상수 중 하나를 지정합니다.

❖ 색 공간 변환 (3/4)

- OpenCV는 표 10-1에 나타난 변환 코드 외에도 훨씬 많은 색 공간 변환을 지원함

▼ 표 10-1 주요 색 공간 변환 코드

ColorConversionCodes 열거형 상수	설명
COLOR_BGR2RGB 또는 COLOR_RGB2BGR	BGR 채널 순서와 RGB 채널 순서를 상호 변환합니다.
COLOR_BGR2GRAY	3채널 BGR 컬러 영상을 1채널 그레이스케일 영상으로 변환합니다.
COLOR_GRAY2BGR	1채널 그레이스케일 영상을 3채널 BGR 컬러 영상으로 변환합니다.
COLOR_BGR2XYZ	BGR 색 공간을 CIE XYZ 색 공간으로 변환합니다.
COLOR_XYZ2BGR	CIE XYZ 색 공간을 BGR 색 공간으로 변환합니다.

❖ 색 공간 변환 (4/4)

▼ 표 10-1 주요 색 공간 변환 코드

ColorConversionCodes 열거형 상수	설명
COLOR_BGR2YCrCb	BGR 색 공간을 YCrCb 색 공간으로 변환합니다.
COLOR_YCrCb2BGR	YCrCb 색 공간을 BGR 색 공간으로 변환합니다.
COLOR_BGR2HSV	BGR 색 공간을 HSV 색 공간으로 변환합니다.
COLOR_HSV2BGR	HSV 색 공간을 BGR 색 공간으로 변환합니다.
COLOR_BGR2Lab	BGR 색 공간을 CIE Lab 색 공간으로 변환합니다.
COLOR_Lab2BGR	CIE Lab 색 공간을 BGR 색 공간으로 변환합니다.

❖ BGR2GRAY와 GRAY2BGR (1/3)

- BGR2GRAY 색 공간 변환 코드는 BGR 컬러 영상을 grayscale 영상으로 변환할 때 사용함
- 컬러 영상을 grayscale 영상으로 변환하는 주된 이유는 연산 속도와 메모리 사용량을 줄이기 위함
- 기본적으로 컬러 영상은 grayscale 영상에 비해 3배 많은 메모리를 필요로 함
- 세 개의 채널에 대해 연산을 수행해야 하기 때문에 더 많은 연산 시간이 필요하게 됨
- 입력 영상에서 색상 정보의 활용도가 그리 높지 않은 경우에는 입력 영상을 grayscale 영상으로 변환하여 처리하는 것이 효율적임

❖ BGR2GRAY와 GRAY2BGR (2/3)

- BGR 3채널 컬러 영상을 그레이스케일 영상으로 변환할 때에는 다음 공식을 사용함

$$Y = 0.299R + 0.587G + 0.114B$$

- 앞의 공식에서 R, G, B는 각각 픽셀의 빨간색, 녹색, 파란색 성분의 값을 나타냄
- Y는 해당 픽셀의 grayscale 성분 크기를 나타냄
- BGR2GRAY 색 공간 변환 코드에 의해 만들어지는 결과 영상은 CV_8UC1 타입으로 설정됨

❖ BGR2GRAY와 GRAY2BGR (3/3)

- 반대로 GRAY2BGR 색 공간 변환 코드는 grayscale 영상을 BGR 컬러 영상으로 변환할 때 사용함
- 이 경우 결과 영상은 CV_8UC3 타입으로 결정되고, 각 픽셀의 B, G, R 색상 성분 값은 다음과 같이 결정됨

$$R = G = B = Y$$

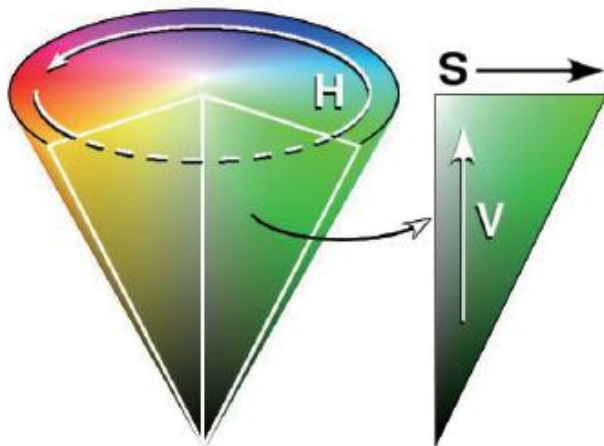
- grayscale 영상 위에 색깔이 있는 선 또는 글씨를 나타내기 위해 미리 grayscale 영상을 BGR 컬러 영상으로 변환함
- 기본적으로 grayscale 영상에는 색깔 있는 선 또는 글씨를 출력할 수 없기 때문임

❖ BGR2HSV와 HSV2BGR (1/3)

- **HSV 색 모델**은 색상(hue), 채도(saturation), 명도(value)로 색을 표현하는 방식임
- 색상은 빨간색, 노란색, 녹색과 같은 색의 종류를 의미합니다. **채도는 색의 순도를 나타냄**
- 빨간색에 대하여 채도가 높으면 맑은 선홍색이고, 채도가 낮으면 탁한 빨간색으로 보이게 됨
- **명도는 빛의 세기를 나타냄**
- 명도가 높으면 밝고, 명도가 낮으면 어둡게 느껴짐

❖ BGR2HSV와 HSV2BGR (2/3)

- HSV 색 공간 모형에서 **색상**은 원뿔을 가로로 잘랐을 때 나타나는 원형에서 **각도로 정의됨**
- 각도가 0° 에 해당할 때 빨간색을 나타냄
- 각도가 증가할수록 노란색, 녹색, 하늘색, 파란색, 보라색을 거쳐 각도가 360° 에 가까워지면 다시 빨간색으로 표현됨
- **채도**는 원뿔을 가로로 잘랐을 때 나타나는 원 모양의 중심에서 최솟값을 가짐
- 원의 중심에서 방사형으로 멀어지는 방향으로 값이 증가함
- **명도**는 원뿔 아래쪽 꼭지점에서 최솟값을 갖고 원뿔의 축을 따라 올라가면서 증가함



◀ 그림 10-2 원뿔 모양의 HSV 색 공간 모형

❖ BGR2HSV와 HSV2BGR (3/3)

- OpenCV에서 BGR2HSV 색 공간 변환 코드를 이용하여 8비트 BGR 영상을 HSV 영상으로 변환할 경우, H 값은 0부터 179 사이의 정수로 표현되고, S와 V는 0부터 255 사이의 정수로 표현함
- 색상 값은 보통 0°부터 360° 사이의 각도로 표현함
- uint8 자료형으로는 256 이상의 정수를 표현할 수 없기 때문에 OpenCV에서는 각도를 2로 나눈 값을 H 성분으로 저장함
- H 값은 0에서 360 사이의 실수로 표현되고 S와 V는 0에서 1 사이의 실수 값으로 표현됨

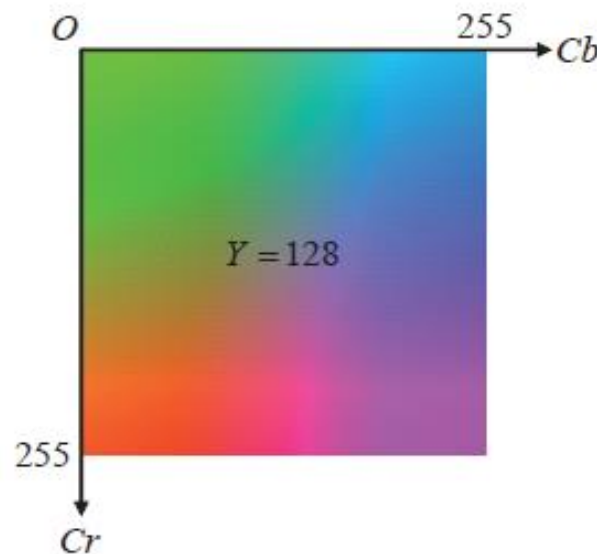
❖ BGR2YCrCb와 YCrCb2BGR (1/5)

- YCrCb 색 공간에서 Y 성분은 밝기 또는 휘도(luminance) 정보를 나타냄
- Cr과 Cb 성분은 색상 또는 색차(chrominance) 정보를 나타냄
- RGB 색상 성분으로부터 Y 성분을 계산하는 공식은 grayscale 계산 공식과 완전히 같음
- Cr과 Cb 성분은 밝기에 대한 정보는 포함하지 않으며 오직 색상에 대한 정보만을 가지고 있음

❖ BGR2YCrCb와 YCrCb2BGR (2/5)

- YCrCb 색 공간은 **영상을 grayscale 정보와 색상 정보로 분리하여 처리할 때 유용함**
- OpenCV에서 BGR2YCrCb 색 공간 변환 코드를 이용하여 8비트 BGR 영상을 YCrCb 영상으로 변환할 경우, Y, Cr, Cb 각각의 성분 값은 0부터 255 사이의 값으로 표현됨
- HSV 색 공간에서는 H 값만을 이용하여 색 종류를 구분할 수 있지만 YCrCb 색 공간에서는 Cr과 Cb를 함께 조합하여 색을 구분할 수 있음

▼ 그림 10-3 CrCb 색 분포(Y=128)



❖ BGR2YCrCb와 YCrCb2BGR (3/5)

- 코드 10-2에 나타난 `color_grayscale()` 함수는 `butterfly.jpg` 영상을 3채널 BGR 컬러 영상 형식으로 불러온 후 그레이스케일 영상으로 변환하여 화면에 나타냄

코드 10-2 컬러 영상을 grayscale 영상으로 변환하기 (ColorOp.py)

```
1  import cv2
2
3  def color_grayscale():
4      src = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)
5
6      if src is None:
7          print('Image load failed!')
8          return
9
10     dst = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
11
12     cv2.imshow('src', src)
13     cv2.imshow('dst', dst)
14     cv2.waitKey()
15     cv2.destroyAllWindows()
16
```

❖ BGR2YCrCb와 YCrCb2BGR (4/5)

코드 10-2 컬러 영상을 grayscale 영상으로 변환하기 (ColorOp.py)

```
17 if __name__ == '__main__':  
18     color_grayscale()
```

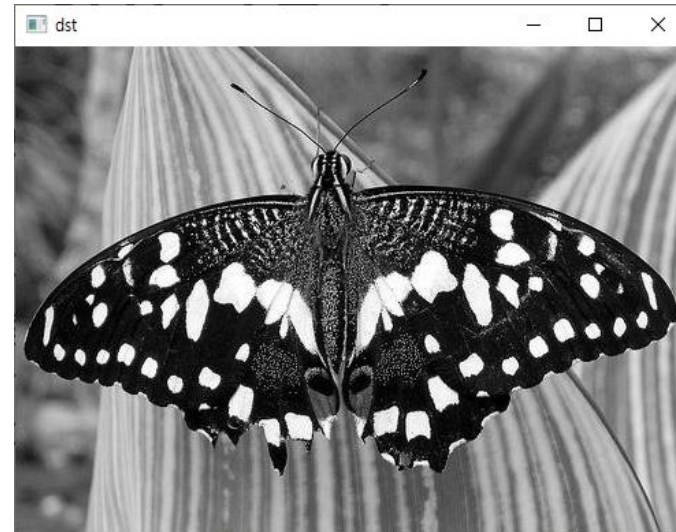
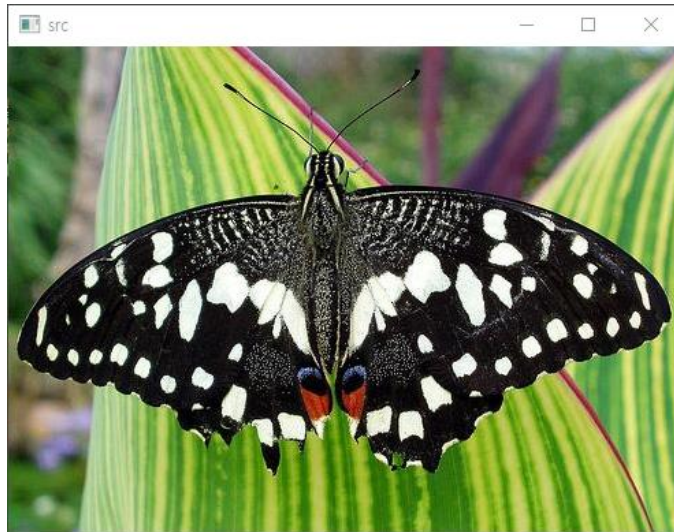
● ColorOp.py 소스 코드 설명

- 4행 `imread()` 함수의 두 번째 인자를 `cv2.IMREAD_COLOR`로 지정하여 3채널 BGR 컬러 영상 형식으로 영상 파일을 불러옵니다.
- 10행 3채널 BGR 컬러 영상 `src`를 grayscale 영상으로 변환하여 `dst`에 저장합니다.

❖ BGR2YCrCb와 YCrCb2BGR (5/5)

- 그림 10-4에서 src는 입력 영상 butterfly.jpg 파일임
- dst는 그레이스케일 형식으로 변경된 영상임
- cvtColor() 함수로 생성한 dst 영상은 `imread("butterfly.jpg", cv2.IMREAD_GRAYSCALE)` 코드를 이용하여 나비 영상을 그레이스케일 영상 형식으로 불러오는 것과 완전히 같음

▼ 그림 10-4 컬러 영상을 그레이스케일 영상으로 변환하기 예제 실행 결과



❖ 색상 채널 나누기 (1/8)

- `imread()` 함수로부터 생성된 컬러 영상은 하나의 픽셀이 파란색(B), 녹색(G), 빨간색(R) 세 개의 색상 정보를 가지고 있음
- OpenCV에서 컬러 영상은 보통 `uint8` 자료형을 사용하고 세 개의 채널을 갖는 `numpy.ndarray` 객체로 표현함
- 컬러 영상을 다루다 보면 빨간색 성분만을 이용하거나 HSV 색 공간으로 변환한 후 H 성분만을 이용하는 경우가 종종 발생함
- 이러한 경우에는 3채널 `numpy.ndarray` 객체를 1채널 `numpy.ndarray` 객체 세 개로 분리해서 다루는 것이 효율적임

❖ 색상 채널 나누기 (2/8)

- OpenCV에서 다채널 행렬을 1채널 행렬 여러 개로 변환할 때에는 `split()` 함수를 사용함

```
planes = cv2.split(src)
```

src	입력 다채널 행렬
planes	분리된 1채널 행렬들을 저장하고 있는 행렬

여기서 `planes`의 자료형은 튜플(tuple)입니다.

❖ 색상 채널 나누기 (3/8)

- `split()` 함수와 반대로 1채널 행렬 여러 개를 합쳐서 다채널 행렬 하나를 생성하려면 `merge()` 함수를 사용함

```
dst = cv2.merge(src)
```

src	1채널 행렬들로 구성된 입력 행렬
dst	출력 다채널 행렬

❖ 색상 채널 나누기 (4/8)

- 코드 10-3에 나타난 `color_split()` 함수는 `candies.png` 파일을 3채널 컬러 영상 형식으로 불러온 후 각 채널을 분리함
- 이때 분리된 각 채널은 `CV_8UC1` 타입의 grayscale 영상이므로 `imshow()` 함수를 이용하여 화면에 나타낼 수 있음

코드 10-3 BGR 컬러 영상의 채널 나누기 (ColorOp.py)

```
1  import cv2
2
3  def color_grayscale():
4      src = cv2.imread('candies.png', cv2.IMREAD_COLOR)
5
6      if src is None:
7          print('Image load failed!')
8          return
9
```

❖ 색상 채널 나누기 (5/8)

코드 10-3 BGR 컬러 영상의 채널 나누기 (ColorOp.py)

```
10      # b_plane, g_plane, r_plane = cv2.split(src)
11      bgr_planes = cv2.split(src)
12
13      cv2.imshow('src', src)
14      cv2.imshow('B_plane', bgr_planes[0])
15      cv2.imshow('G_plane', bgr_planes[1])
16      cv2.imshow('R_plane', bgr_planes[2])
17
18      cv2.waitKey()
19      cv2.destroyAllWindows()
20
21  if __name__ == '__main__':
22      color_split()
```

❖ 색상 채널 나누기 (6/8)

- ColorOp.py 소스 코드 설명

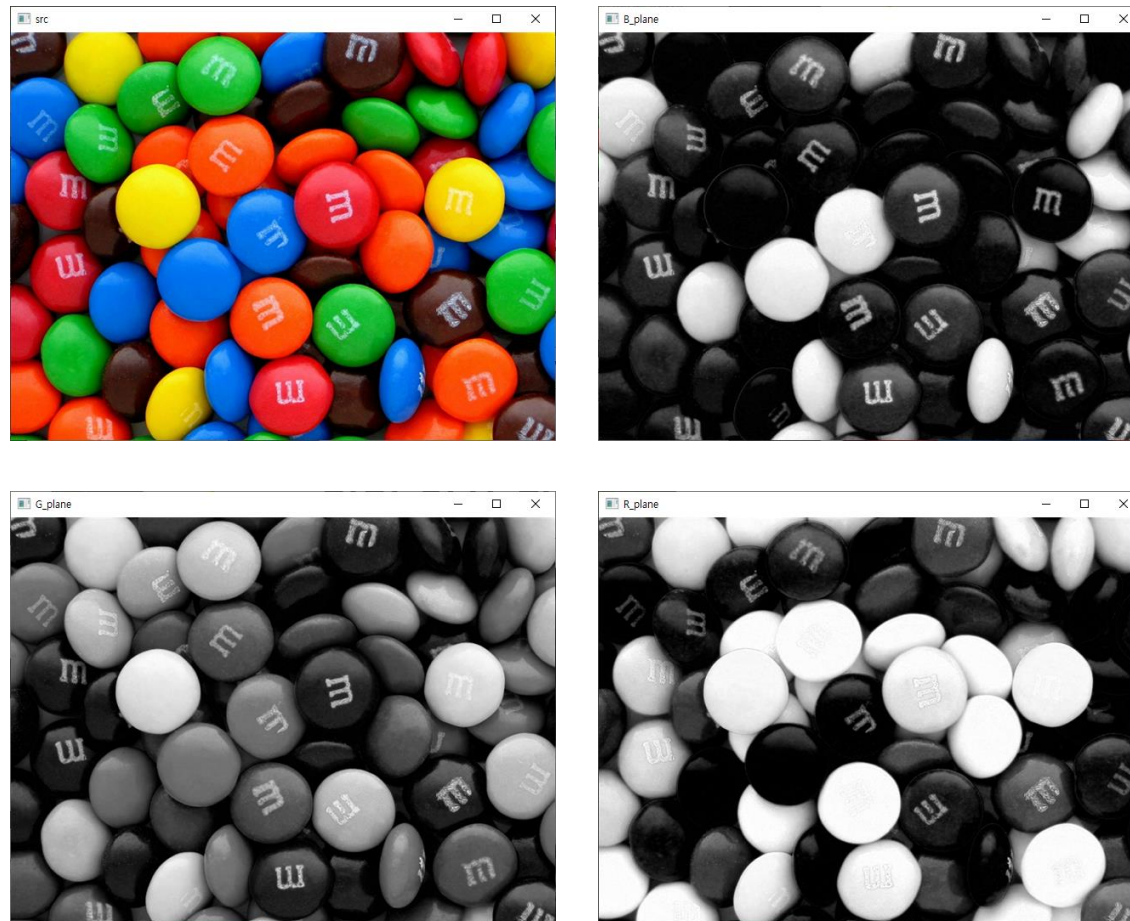
- 4행 candies.png 영상을 3채널 BGR 컬러 영상 형식으로 불러옵니다.
- 11행 src 영상의 채널을 분할하여 bgr_planes 벡터에 저장합니다.
 bgr_planes[0]에는 파란색 색상 평면, bgr_planes[1]에는 녹색 색상 평면, bgr_planes[2]에는
 빨간색 색상 평면이 저장됩니다.

❖ 색상 채널 나누기 (7/8)

- 그림 10-5에서 src는 입력 영상 candies.png 파일이고, B_plane은 파란색 채널, G_plane은 녹색 채널, R_plane은 빨간색 채널을 나타내는 영상임
- 원본 영상 src에서 파란색 초콜릿 영역은 파란색 성분 값이 크기 때문에 B_plane 영상에서 밝은 흰색으로 표시됨
- 원본 영상 src에서 노란색 캔디 영역은 빨간색과 녹색 성분 값이 크기 때문에 R_plane과 G_plane 영상에서 밝게 표현됨
- 반면에 B_plane 영상에서는 어두운 검은색으로 표현되는 것을 확인할 수 있음

❖ 색상 채널 나누기 (8/8)

▼ 그림 10-5 BGR 컬러 영상의 채널 나누기 예제 실행 결과



10.2 컬러 영상 처리 기법

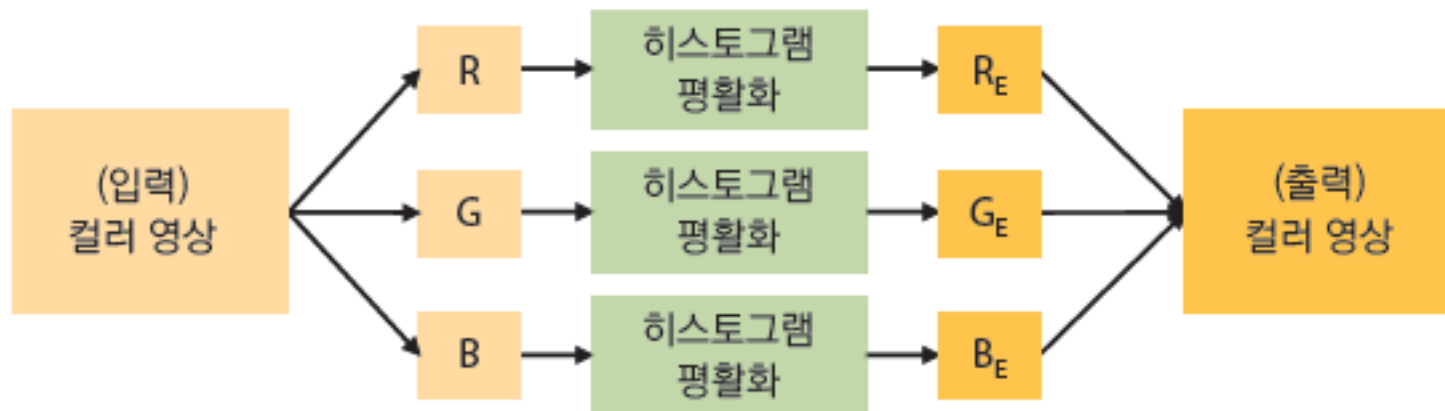
10.1 컬러 영상 다루기

❖ 컬러 히스토그램 평활화 (1/9)

- OpenCV에서는 `equalizeHist()` 함수를 통해 히스토그램 평활화를 수행 할 수 있지만 **`equalizeHist()` 함수는 grayscale 영상만 입력으로 받을 수 있음**
- 3채널 컬러 영상에 대해 히스토그램 평활화를 수행하려면 OpenCV 함수를 조합하여 직접 구현해야 함
- 입력 영상을 R, G, B 각 채널로 나누고, 채널별로 히스토그램 평활화를 수행한 후 다시 채널을 합치는 방식임
- 이러한 방식은 R, G, B 색상 채널마다 서로 다른 형태의 명암비 변환 함수를 사용하게 됨으로써 원본 영상과 다른 색상의 결과 영상이 만들어지는 단점이 있음

❖ 컬러 히스토그램 평활화 (2/9)

▼ 그림 10-6 RGB 각 채널에 히스토그램 평활화 수행하기



❖ 컬러 히스토그램 평활화 (3/9)

- 그림 10-7(a)는 테스트로 사용한 입력 영상 pepper.bmp 파일이며, 이 영상은 전체적으로 녹색 성분이 많이 포함되어 있음
- 이 영상에 대해 R, G, B 각각의 색상 채널에 대해 히스토그램 평활화를 수행하고, 다시 채널을 합쳐서 만든 결과 영상이 그림 10-7(b)임
- 입력 영상에서 상대적으로 적었던 파란색 성분이 강해지면서 원본 영상의 색감과 완전히 다른 결과 영상이 만들어짐

▼ 그림 10-7 RGB 각 채널에 히스토그램 평활화 수행하기 결과



(a)



(b)

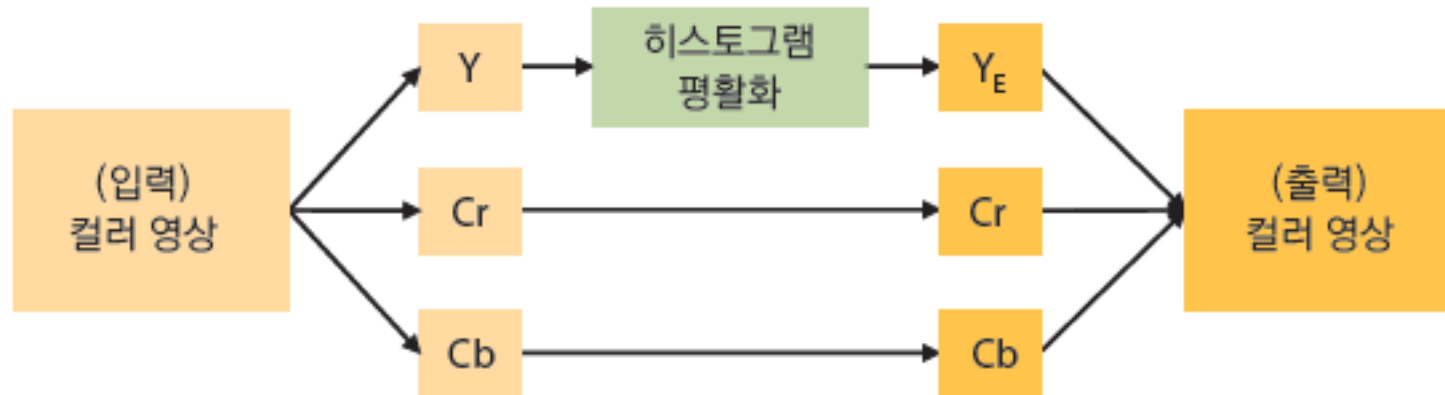
❖ 컬러 히스토그램 평활화 (4/9)

- 컬러 영상의 색감은 변경하지 않고 명암비를 높이려면 영상의 밝기 정보만을 사용해야 함
- 보통 컬러 영상에 대하여 히스토그램 평활화를 수행하려면 입력 영상을 밝기 정보와 색상 정보로 분리함
- 밝기 정보에 대해서만 히스토그램 평활화를 수행함
- 예를 들어 YCrCb 색 공간을 사용할 경우, 입력 영상을 YCrCb 색 공간으로 변환하여 **Y 성분에 대해서만 히스토그램 평활화를 수행**하고 Cr과 Cb 색 성분은 변경하지 않고 그대로 유지함
- 변경된 Y 채널과 Cr, Cb 채널을 다시 합치면 컬러 히스토그램 평활화 결과 영상을 얻을 수 있음

❖ 컬러 히스토그램 평활화 (5/9)

- 이러한 방식의 컬러 히스토그램 평활화 방법을 그림 10-8에 그림으로 나타냄
- 이 방식은 Cr과 Cb 채널의 색상 정보는 전혀 변경하지 않으므로 입력 영상의 색감이 그대로 유지되고, 오직 밝기 성분에 대해서만 명암비가 증가하게 됨

▼ 그림 10-8 컬러 히스토그램 평활화 방법



❖ 컬러 히스토그램 평활화 (6/9)

- 코드 10-4에 나타난 ColorEq 예제 프로그램은 pepper.bmp 영상에 대하여 컬러 히스토그램 평활화를 수행하고 그 결과를 화면에 나타냄

코드 10-4 컬러 영상의 히스토그램 평활화 예제 (ColorEq.py)

```
1  import sys
2  import cv2
3
4  src = cv2.imread('pepper.bmp', cv2.IMREAD_COLOR)
5
6  if src is None:
7      print('Image load failed!')
8      sys.exit()
9
```


❖ 컬러 히스토그램 평활화 (7/9)

코드 10-4 컬러 영상의 히스토그램 평활화 예제 (ColorEq.py)

```
10 src_ycrb = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)
11
12 ycrb_planes = cv2.split(src_ycrb)
13
14 ycrb_planes = list(ycrb_planes)
15
16 ycrb_planes[0] = cv2.equalizeHist(ycrb_planes[0])
17
18 dst_ycrb = cv2.merge(ycrb_planes)
19
20 dst = cv2.cvtColor(dst_ycrb, cv2.COLOR_YCrCb2BGR)
21
22 cv2.imshow('src', src)
23 cv2.imshow('dst', dst)
24 cv2.waitKey()
25 cv2.destroyAllWindows()
```

**ycrb_planes의 자료형은 튜플(tuple)로
값의 변경이 불가능합니다.
따라서 리스트(list)로 자료형을 변경합니다.**

❖ 컬러 히스토그램 평활화 (8/9)

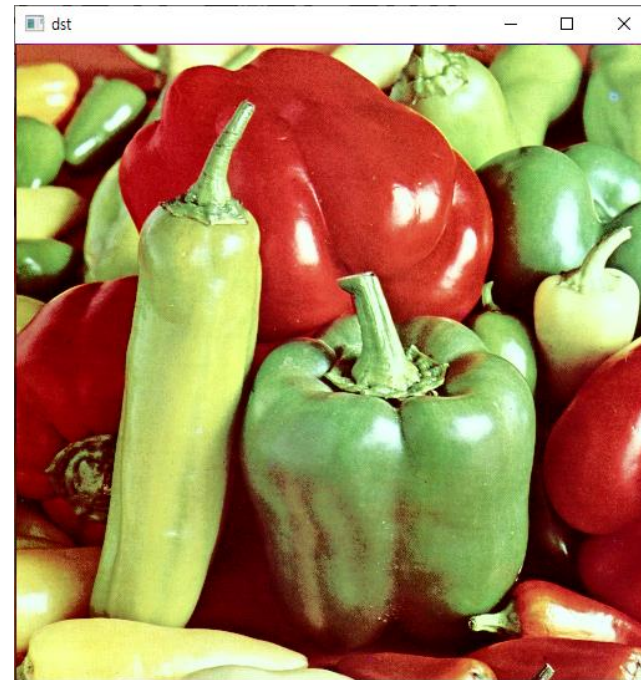
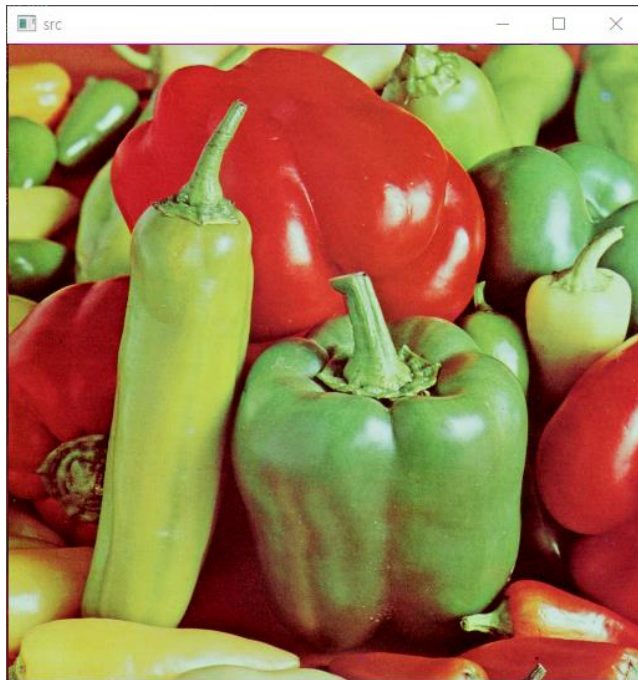
- ColorEq.py 소스 코드 설명

- 4행 `pepper.bmp` 영상을 3채널 BGR 컬러 영상으로 불러와서 `src`에 저장합니다.
- 10행 BGR 색 공간의 `src` 영상을 YCrCb 색 공간으로 변경하여 `src_ycrcb`에 저장합니다.
- 12행 `src_ycrcb` 영상의 채널을 분리하여 `ycrcb_planes`에 저장합니다.
- 16행 Y 성분에 해당하는 `ycrcb_planes[0]` 영상에 대해서만 히스토그램 평활화를 수행합니다.
- 18행 `ycrcb_planes` 벡터에 들어 있는 세 영상을 합쳐서 `dst_ycrcb` 영상을 생성합니다.
- 20행 `dst_ycrcb` 영상의 색 공간을 BGR 색 공간으로 변환하여 `dst`에 저장합니다.

❖ 컬러 히스토그램 평활화 (9/9)

- 그림 10-9에서 src 영상은 pepper.bmp 파일이고, dst 영상은 컬러 히스토그램 평활화가 적용된 결과 영상임
- 원본 영상의 색감은 그대로 유지한 채 명암비가 높아진 것을 확인할 수 있음

▼ 그림 10-9 컬러 영상의 히스토그램 평활화 예제 실행 결과



❖ 색상 범위 지정에 의한 영역 분할 (1/10)

- 컬러 영상을 다루는 응용에서 자주 요구되는 기법은 특정 색상 영역을 추출하는 작업임
- 예를 들어 입력 영상에서 빨간색 픽셀을 모두 찾아내면 빨간색 객체의 위치와 크기를 알 수 있음
- 컬러 영상에서 빨간색, 파란색 등의 대표적인 색상 영역을 구분할 때에는 RGB 색 공간보다 HSV등의 **색상(H) 정보가 따로 설정되어 있는 색 공간을 사용하는 것이 유리함**
- 예를 들어 HSV색 공간에서 녹색은 H 값이 60 근방으로 표현되기 때문에 H 값이 60에 가까운지를 조사하여 녹색 픽셀을 찾아낼 수 있음

❖ 색상 범위 지정에 의한 영역 분할 (2/10)

- OpenCV에서 행렬의 원소 값이 특정 범위 안에 있는지 확인하려면 `inRange()` 함수를 사용할 수 있음

```
dst = cv2.inRange(src, lowerb, upperb)
```

src	입력 영상
lowerb	하한 값
upperb	상한 값
dst	출력 다채널 행렬

❖ 색상 범위 지정에 의한 영역 분할 (3/10)

- `inRange()` 함수는 입력 영상 `src`의 픽셀 값이 지정한 밝기 또는 색상 범위에 포함되어 있으면 흰색임
- 그렇지 않으면 검은색으로 채워진 마스크 영상 `dst`를 반환함
- 입력 영상 `src`에는 그레이스케일 영상 같은 1채널 행렬과 컬러 영상 같은 다채널 행렬을 모두 지정할 수 있음
- 만약 grayscale 영상을 입력 영상으로 사용할 경우, 특정 밝기 값 범위에 있는 픽셀 영역을 추출할 수 있음

❖ 색상 범위 지정에 의한 영역 분할 (4/10)

- 1채널 영상에 대해 `inRange()` 함수의 동작을 수식으로 표현하면 다음과 같음

$$\text{dst}(x, y) = \begin{cases} 255 & \text{lowerb}(x, y) \leq \text{src}(x, y) \leq \text{upperb}(x, y) \text{ 일 때} \\ 0 & \text{그 외} \end{cases}$$

- 만약 `src` 영상의 채널이 두 개 이상이라면 입력 영상의 각 채널 값이 모두 지정된 범위를 만족할 때 `dst` 영상의 픽셀 값이 255로 설정됨

❖ 색상 범위 지정에 의한 영역 분할 (5/10)

- 코드 10-5에 나타난 `inrange` 예제 프로그램은 영상 출력 창에 두개의 트랙바를 붙여서 사용자가 HSV 색 공간에서 색상의 하한 값과 상한 값을 설정할 수 있도록 함
- 사용자가 설정한 색상 값 범위에 해당하는 영역은 흰색, 그 외의 영역은 검은색으로 표현된 마스크 영상을 화면에 출력함

코드 10-5 `inRange()` 함수를 이용한 특정 색상 영역 분할 (`inrange.py`)

```
1  import cv2
2
3  def on_hue_changed(_=None):
4      lower_hue = cv2.getTrackbarPos('Lower Hue', 'mask')
5      upper_hue = cv2.getTrackbarPos('Upper Hue', 'mask')
6
7      lowerb = (lower_hue, 100, 0)
8      upperb = (upper_hue, 255, 255)
9      mask = cv2.inRange(src_hsv, lowerb, upperb)
10
11     cv2.imshow('mask', mask)
12
```


❖ 색상 범위 지정에 의한 영역 분할 (6/10)

코드 10-5 inRange() 함수를 이용한 특정 색상 영역 분할 (inrange.py)

```
13 def main():
14     global src_hsv
15
16     src = cv2.imread('candies.png', cv2.IMREAD_COLOR)
17
18     if src is None:
19         print('Image load failed!')
20         return
21
22     src_hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)
23
24     cv2.imshow('src', src)
25
26     cv2.namedWindow('mask')
27     cv2.createTrackbar('Lower Hue', 'mask', 40, 179, on_hue_changed)
28     cv2.createTrackbar('Upper Hue', 'mask', 80, 179, on_hue_changed)
29     on_hue_changed(0)
30
31     cv2.waitKey()
32     cv2.destroyAllWindows()
33
```

❖ 색상 범위 지정에 의한 영역 분할 (7/10)

코드 10-5 inRange() 함수를 이용한 특정 색상 영역 분할 (inrange.py)

```
34  if __name__ == '__main__':  
35      main()  
36
```

❖ 색상 범위 지정에 의한 영역 분할 (8/10)

- inrange.py 소스 코드 설명

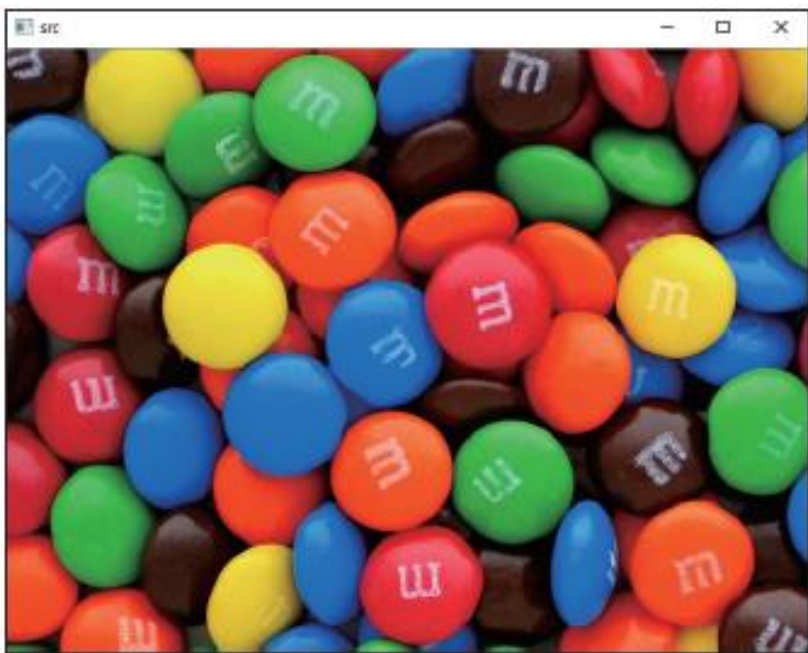
- 4행 두 개의 트랙바 위치를 저장할 변수 `lower_hue`, `upper_hue`를 전역 변수로 선언합니다.
- 16행 `candies.png` 파일을 불러와서 `src` 변수에 저장합니다.
- 22행 `src` 영상을 HSV 색 공간으로 변환하여 `src_hsv`에 저장합니다.
- 27~28행 색상의 하한 값과 상한 값을 조절할 수 있는 두 개의 트랙바를 생성합니다.
 색상의 최댓값을 179로 설정하고, 두 트랙바의 콜백 함수를 모두 `on_hue_change()` 함수로 설정합니다.
- 29행 프로그램이 처음 실행될 때 영상이 정상적으로 출력되도록 트랙바 콜백 함수를 강제로 호출합니다.
- 7~8행 사용자가 지정한 색상의 하한 값과 상한 값을 이용하여 `lowerb`, `upperb` 객체를 생성합니다.
 채도의 범위는 임의로 100부터 255로 설정하였습니다.
 명도의 영향은 무시하도록 범위를 0부터 255로 설정하였습니다.
- 9행 `src_hsv` 영상에서 HSV 색 성분 범위가 `lowerb`부터 `upperb` 사이인 위치의 픽셀만 흰색으로 설정한 `mask` 영상을 생성합니다.

❖ 색상 범위 지정에 의한 영역 분할 (9/10)

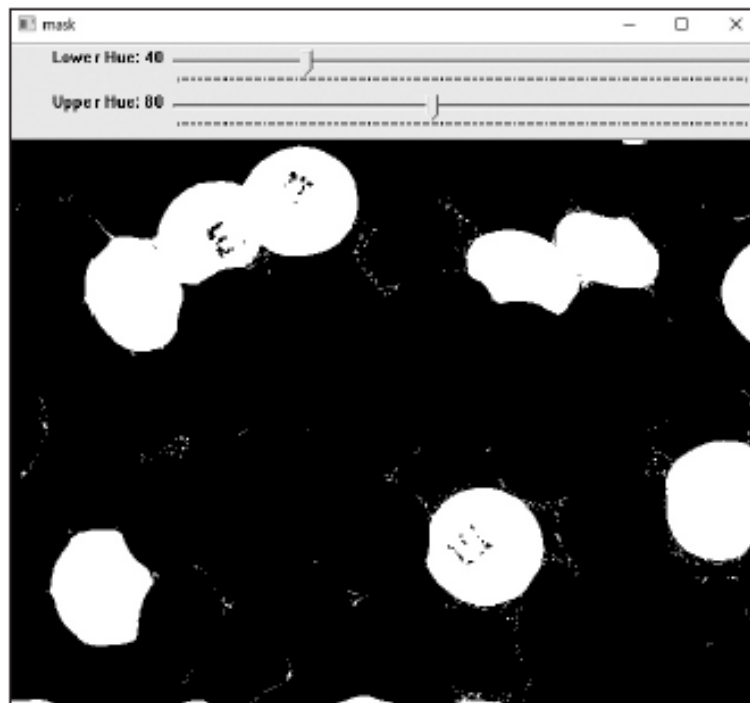
- 그림 10-10(a)는 입력 영상인 candies.png 파일이고, 그림 10-10(b)는 입력 영상에서 녹색 초콜릿 영역만 찾은 결과임
- 그림 10-10(b)는 입력 영상에서 색상 H의 범위가 40에서 80, 채도 S의 범위가 100에서 255, 명도 V의 범위가 0에서 255 사이인 픽셀 위치를 찾은 결과임
- OpenCV의 HSV 색 공간에서 H 값이 60 근방이면 녹색을 나타냄
- 채도 값이 100보다 큰 픽셀만 찾은 것은 충분히 선명한 녹색만 찾기 위함임
- 명도 범위를 0에서 255사이로 지정한 것은 명도 값이 몇이든지 상관하지 않겠다는 의미임
- 만약 상단의 트랙바를 Lower Hue를 100, Upper Hue를 140으로 설정하면 그림 10-10(c)와 같이 파란색 초콜릿 영역만 찾을 수 있음

❖ 색상 범위 지정에 의한 영역 분할 (10/10)

▼ 그림 10-10 inRange() 함수를 이용한 특정 색상 영역 분할 예제 실행 결과

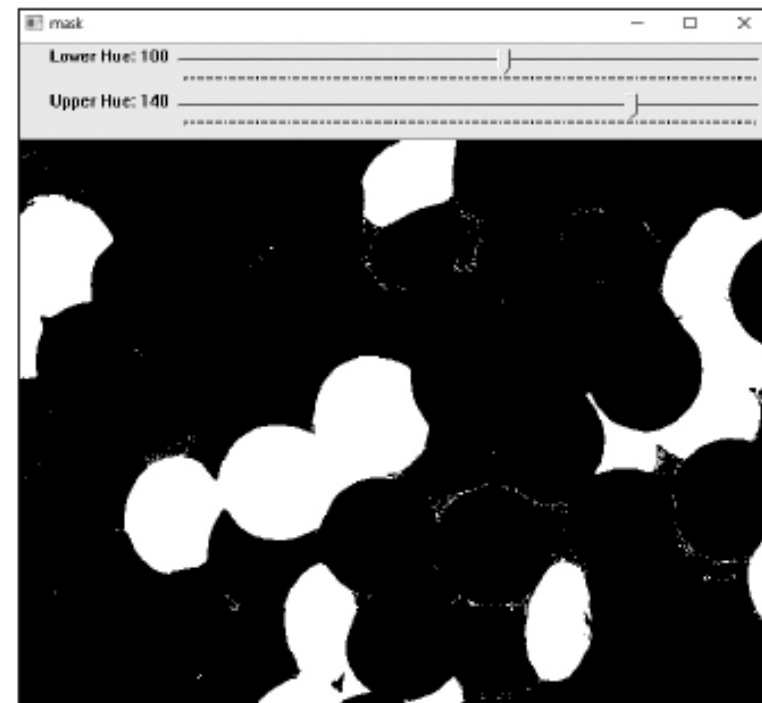


(a)



(b)

H 값이 60 근방이면 녹색



(c)

H 값이 120 근방이면 파란색

❖ 히스토그램 역투영 (1/9)

- HSV 색 공간에서 H 값을 이용하면 간단하게 특정 색상을 골라낼 수 있어서 편리함
- 이러한 방식은 보통 빨간색, 노란색, 녹색, 파란색처럼 원색에 가까운 색상을 찾기에는 효과적임
- 사람의 피부색처럼 미세한 변화가 있거나 색상 값을 수치적으로 지정하기 어려운 경우에는 적합하지 않음

❖ 히스토그램 역투영 (2/9)

- 만약 입력 영상에서 찾고자 하는 객체의 기준 영상을 미리 가지고 있다면 컬러 히스토그램 정보를 이용하여 비슷한 색상 영역을 찾을 수 있음
- 기준 영상으로부터 찾고자 하는 객체의 컬러 히스토그램을 미리 구하고 주어진 입력 영상에서 해당 히스토그램에 부합하는 영역을 찾아내는 방식임
- 이처럼 주어진 히스토그램 모델과 일치하는 픽셀을 찾아내는 기법을 **히스토그램 역투영(histogram backprojection)**이라고 함
- 예를 들어 피부색에 대한 색상 히스토그램을 가지고 있다면 역투영 방법을 사용하여 영상에서 피부색 영역을 검출할 수 있음

❖ 히스토그램 역투영 (3/9)

- OpenCV에서 히스토그램 역투영은 `calcBackProject()` 함수를 이용하여 수행할 수 있음

```
backProject = cv2.calcBackProject(images, channels, hist, ranges, scale, uniform)
```

images	입력 영상의 배열 (모든 영상의 크기와 깊이는 같아야 합니다.)
channels	역투영 계산 시 사용할 채널 번호 배열
hist	입력 히스토그램
backProject	출력 히스토그램 역투영 영상. 입력 영상과 같은 크기, 같은 깊이를 갖는 1채널 행렬
ranges	각 차원의 히스토그램 빈 범위를 나타내는 배열의 배열
scale	히스토그램 역투영 값에 추가적으로 곱할 값
uniform	히스토그램 빈의 간격이 균등한지를 나타내는 플래그

❖ 히스토그램 역투영 (4/9)

- 코드 10-6에 나타난 backproj 예제 프로그램은 기준 영상으로부터 피부색 영역에 대한 히스토그램을 추출하고, 이 히스토그램 정보를 이용하여 입력 영상에서 피부색 영역을 검출함

코드 10-6 히스토그램 역투영을 이용한 피부색 영역 검출 예제 (backproj.py)

```
1  import cv2
2
3  # Calculate CrCb histogram from a reference image
4  ref = cv2.imread('ref.png', cv2.IMREAD_COLOR)
5  mask = cv2.imread('mask.bmp', cv2.IMREAD_GRAYSCALE)
6  ref_ycrcb = cv2.cvtColor(ref, cv2.COLOR_BGR2YCrCb)
7
8  channels = [1, 2]
9  cr_bins = 128
10 cb_bins = 128
11 histSize = [cr_bins, cb_bins]
12 cr_range = [0, 256]
13 cb_range = [0, 256]
14 ranges = cr_range + cb_range
15
16 hist = cv2.calcHist([ref_ycrcb], channels, mask, histSize, ranges)
```

❖ 히스토그램 역투영 (5/9)

코드 10-6 히스토그램 역투영을 이용한 피부색 영역 검출 예제 (backproj.py)

```
17
18 # Apply histogram backprojection to an input image
19 src = cv2.imread('kids.png', cv2.IMREAD_COLOR)
20 src_ycrcb = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)
21
22 backproj = cv2.calcBackProject([src_ycrcb], channels, hist, ranges, 1)
23
24 cv2.imshow('src', src)
25 cv2.imshow('backproj', backproj)
26 cv2.waitKey()
27 cv2.destroyAllWindows()
```

❖ 히스토그램 역투영 (6/9)

● backproj.py 소스 코드 설명

- 4행 피부색 히스토그램 정보를 추출할 기준 영상 ref.png 파일을 불러옵니다.
- 5행 기준 영상에서 피부색이 있는 위치를 흰색으로 표시한 마스크 영상 mask.bmp 파일을 불러옵니다.
- 6행 기준 영상을 YCrCb 색 공간으로 변환합니다.
- 8~16행 기준 영상에서 피부색 영역의 CrCb 2차원 히스토그램을 계산하여 hist에 저장합니다.
- 19~20행 입력 영상 kids.png 파일을 불러와 YCrCb 색 공간으로 변환합니다.
- 22행 앞서 구한 히스토그램 hist를 이용하여 입력 영상에서 히스토그램 역투영을 수행합니다.
역투영 결과는 backproj에 저장됩니다.

❖ 히스토그램 역투영 (7/9)

- 그림 10-11(a)는 피부색이 많이 포함된 기준 영상이고, 그림 10-11(b)는 기준 영상에서 실제로 피부색이 나타난 영역만 흰색으로 표시한 마스크 영상임
- 마스크 영상은 포토샵 같은 영상 편집 툴을 이용하여 미리 생성함
- 이 두 영상을 이용하여 기준 영상에서 피부색 영역의 CrCb 히스토그램을 구하고, 그 결과를 grayscale 영상 형식으로 나타낸 결과가 그림 10-11(c)임
- 그림 10-11(c)에서 밝게 나타나는 부분이 CrCb 평면에서 피부색을 표현하는 영역임

▼ 그림 10-11 기준 영상과 마스크 영상을 이용하여 피부색 CrCb 히스토그램 구하기



(a)



(b)



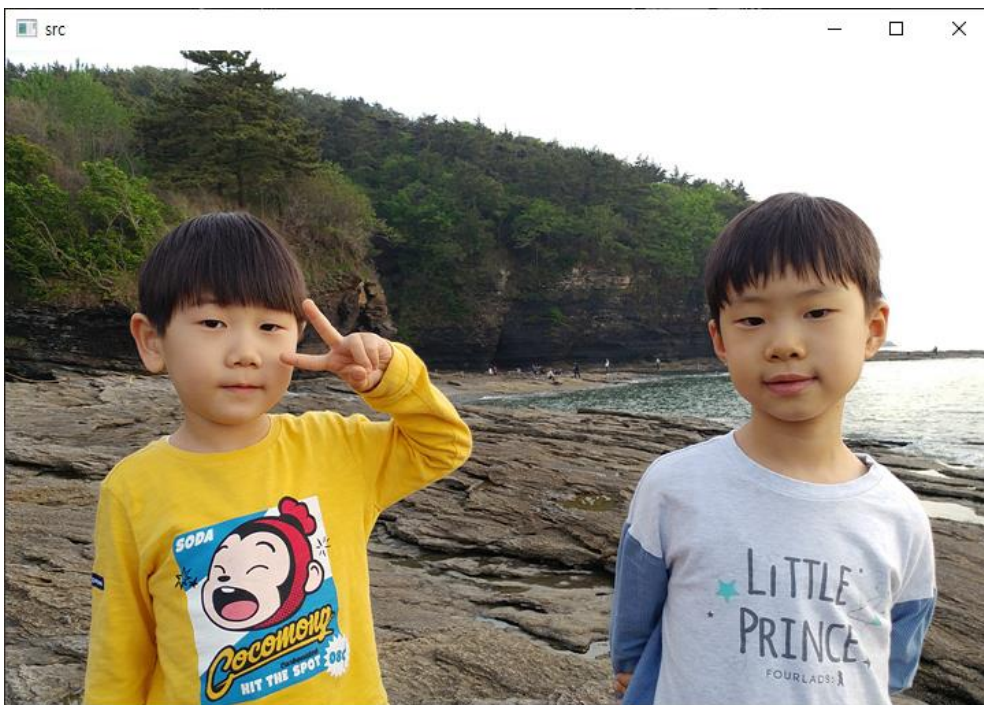
(c)

❖ 히스토그램 역투영 (8/9)

- 그림 10-12에서 src는 입력 영상인 kids.png 파일이고, backproj는 히스토그램 역투영을 통해 구한 피부색 영역임
- calcBackProject() 함수가 반환하는 backproj 영상은 CV_8UC1 타입이므로 imshow() 함수를 이용하여 쉽게 화면에 나타낼 수 있음
- backproj 영상에서 밝은 회색 또는 흰색으로 표시된 영역은 입력 영상의 픽셀 값이 지정한 히스토그램에서 높은 빈도수로 표현됨을 의미함
- 반대로 backproj 영상에서 어두운 회색 또는 검은색으로 표시된 영역은 해당 위치의 입력 영상 픽셀 값이 지정한 히스토그램에서 빈도수가 낮거나 0임을 나타냄

❖ 히스토그램 역투영 (9/9)

▼ 그림 10-12 히스토그램 역투영을 이용한 피부색 영역 검출 예제 실행 결과



THANK YOU!

Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: <https://sites.google.com/view/beomkwon/home>