

강원대학교  
AI 소프트웨어학과

---

# 데이터 전처리

## - 이미지 데이터 -

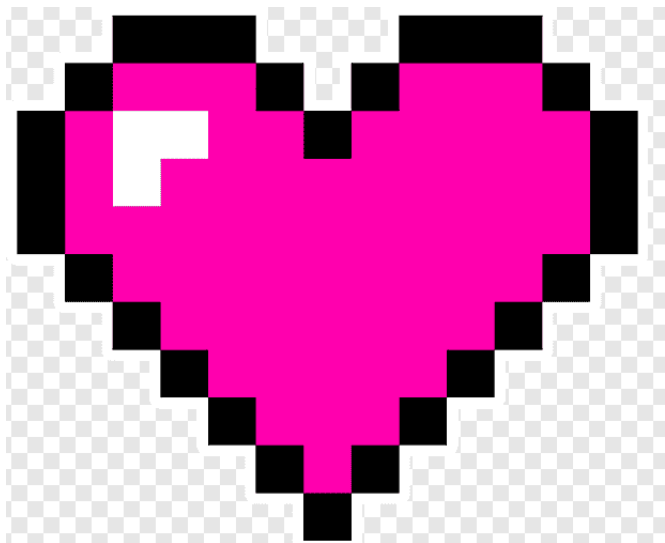
---

### 이미지 데이터

- 이미지 데이터는 일반적으로 픽셀(pixel)의 그리드로 구성되며, 픽셀(pixel)은 이미지의 가장 작은 구성 단위임
- 픽셀 구성
  - 이미지는 픽셀(pixel)이라 불리는 작은 점들의 집합으로 구성됨
  - 각 픽셀은 이미지의 한 부분의 색과 밝기를 나타냄
- 픽셀 배열
  - 픽셀들은 일반적으로 2차원 격자 또는 그리드 형태의 배열로 구성됨
  - 이미지의 해상도는 그리드의 크기로 결정되며, 예를 들어 1920x1080 해상도는 가로 1920픽셀, 세로 1080픽셀을 의미함

### 이미지 데이터

- 이미지 데이터는 일반적으로 픽셀의 그리드로 구성되며, 각 픽셀은 이미지의 가장 작은 구성 단위임



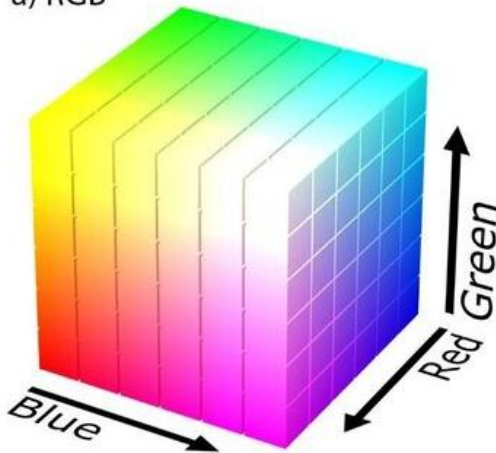
**픽셀 한칸 : 하나의 네모난 점**

**그리드 or window : 전체 이미지의 가로 세로 길이 정보**

### 이미지 데이터

- 대부분의 디지털 화면과 카메라는 RGB 색상 모델을 기반으로 작동
- RGB는 Red (빨강), Green (녹색), Blue (파랑)의 약자로, 이 세 가지 기본 색을 조합하여 다양한 색을 표현함
- 0부터 255의 색상의 전체 강도를 2차원의 배열에 나타내며, 이를 통해 색을 표현할 수 있음(R, G, B)

a) RGB

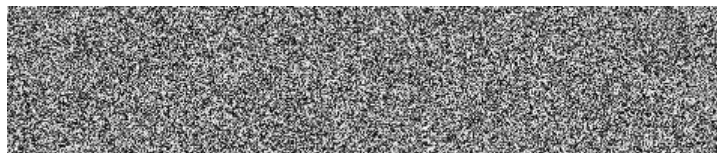


서브 픽셀의 조합을 이용한 색 표현



### 이미지 데이터

- 컴퓨터로 표현 하는 이미지 데이터



무작위 생성값으로 이미지 만들기

```
array([[186,  0, 61, ..., 192, 72, 176],  
       [ 74, 241, 155, ..., 93, 28, 70],  
       [192, 43, 236, ..., 167, 154,  0],  
       ...,  
       [207, 22, 85, ..., 25, 173, 38],  
       [ 7, 242, 151, ..., 89, 99, 186],  
       [ 17, 227, 70, ..., 250, 196, 99]], dtype=uint8)
```

해당 이미지의 수치 값

### 이미지 데이터

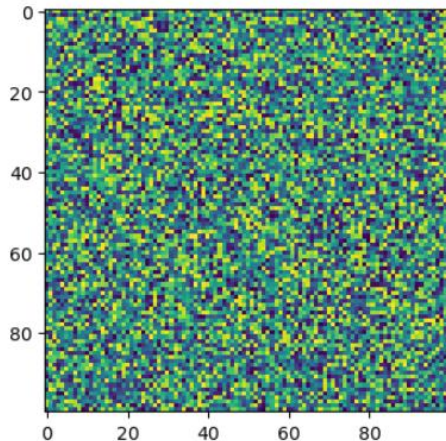
- 컴퓨터로 표현 하는 이미지 데이터

Python version = 3.11.3

Opencv 설치 → pip install opencv-python, 그래프를 그려주는 패키지 → pip install matplotlib

```
import cv2
```

```
import matplotlib.pyplot as plt
```



무작위 생성값으로 이미지 만들기(BGR)

```
random_image = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
plt.figure(figsize=(12, 4))
plt.imshow(random_image) # 보라 → 파랑 → 초록 → 노랑
plt.show()
```

```
[[206 188 110 ... 44 232 237]
 [165 125 74 ... 218 125 170]
 [ 79 39 232 ... 202 219 107]
 ...
 [ 48 118 192 ... 107 70 136]
 [165 19 101 ... 75 40 220]
 [103 22 18 ... 116 160 106]]
```

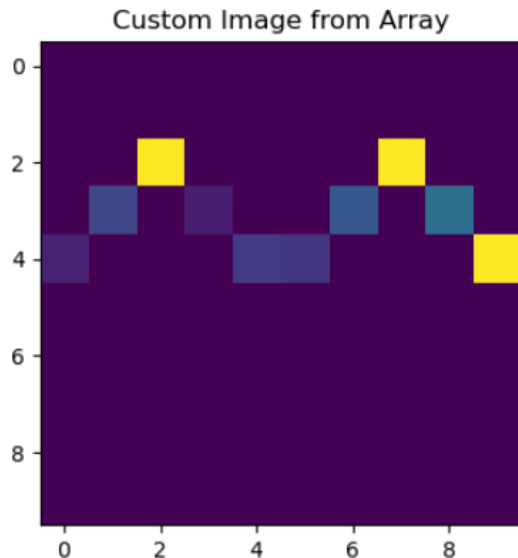
해당 이미지의 수치 값

### 이미지 데이터

- 배열을 활용해 이미지 생성

```
data = np.array([
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 255, 0, 0, 0, 0, 255, 0, 0],
    [0, 55, 0, 21, 0, 0, 70, 0, 92, 0],
    [25, 0, 0, 0, 45, 41, 0, 0, 0, 255],
    [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, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
])
```

```
plt.figure(figsize=(12, 4)) #공간 부여(inch)
plt.imshow(data)
plt.title("Custom Image from Array")
plt.show()
```

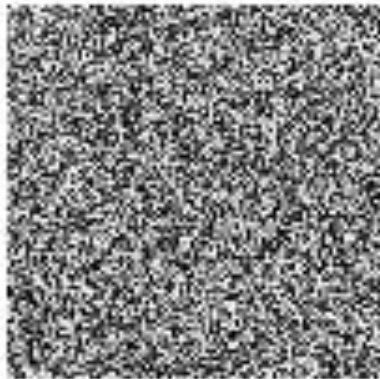


데이터 값	컬러맵
0	어두운 파랑
10	파랑
128	연두~노랑
255	밝은 노랑

### 이미지 데이터

- 컴퓨터로 표현 하는 이미지 데이터

```
random_image = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
RGB_image = cv2.cvtColor(random_image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12, 4))
plt.imshow(RGB_image)
plt.show()
```



```
array([[ 5,   4,  45, ..., 158, 148,  99],
       [ 52, 189, 205, ...,  94,   5, 252],
       [164, 117, 137, ...,  61, 223, 128],
       ...,
       [ 64, 161, 239, ...,  29, 159,  95],
       [ 94, 139, 156, ..., 143,  78,  17],
       [155,  84, 143, ...,  51, 211, 186]], dtype=uint8)
```

무작위 생성값으로 이미지 만들기(RGB)

해당 이미지의 수치 값



### 이미지 데이터

- 배열을 활용해 이미지 생성

```
H, W = 10, 10
```

```
R = np.zeros((H, W), dtype=np.uint8)  
R[0:5, 0:5] = 255
```

```
# 1. R 채널 (Grayscale)  
plt.figure(figsize=(20, 4))  
plt.imshow(R, cmap='gray')  
plt.title('R-Channel')  
plt.show()
```

R-Channel

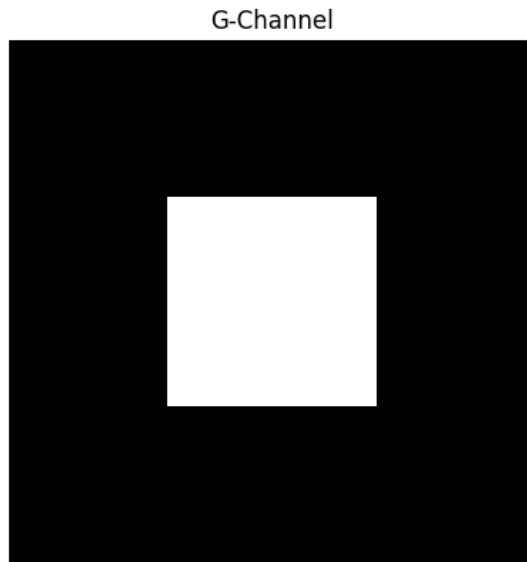


## 이미지 데이터

- 배열을 활용해 이미지 생성

```
G = np.zeros((H, W), dtype=np.uint8)  
G[3:7, 3:7] = 255
```

```
# 2. G 채널 (Grayscale)  
plt.figure(figsize=(20, 4))  
plt.imshow(G, cmap='gray')  
plt.title('G-Channel')  
plt.show()
```



## 이미지 데이터

- 배열을 활용해 이미지 생성

```
B = np.zeros((H, W), dtype=np.uint8)  
B[5:10, 5:10] = 255
```

```
# 3. B 채널 (Grayscale)  
plt.figure(figsize=(20, 4))  
plt.imshow(B, cmap='gray')  
plt.title('B-Channel')  
plt.show()
```

B-Channel



### 이미지 데이터

- 배열을 활용해 이미지 생성

```
two_channel = np.stack([R, G, np.zeros_like(B)], axis=-1)
```

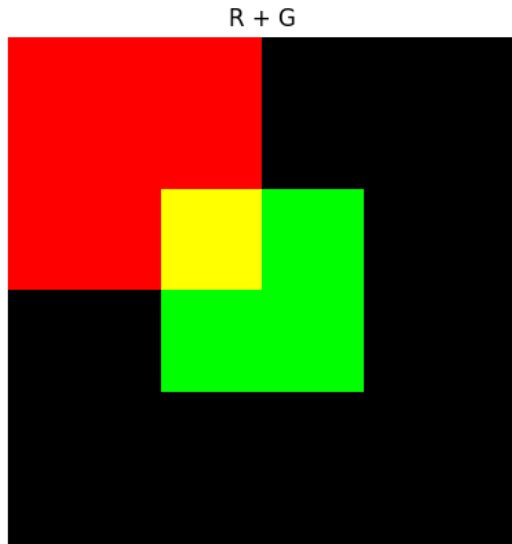
# 4. 2채널 (R+G → 노랑 영역)

```
plt.figure(figsize=(20, 4))
```

```
plt.imshow(two_channel)
```

```
plt.title('R + G')
```

```
plt.show()
```

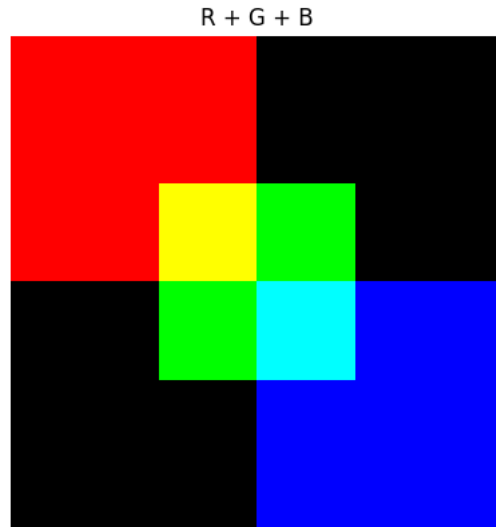


## 이미지 데이터

- 배열을 활용해 이미지 생성

```
three_channel = np.stack([R, G, B], axis=-1)
```

```
# 5. 3채널 (RGB 컬러 전체)  
plt.figure(figsize=(20, 4))  
plt.imshow(three_channel)  
plt.title('R + G + B')  
plt.show()
```



### 이미지 데이터

- 컴퓨터로 표현 하는 이미지 데이터

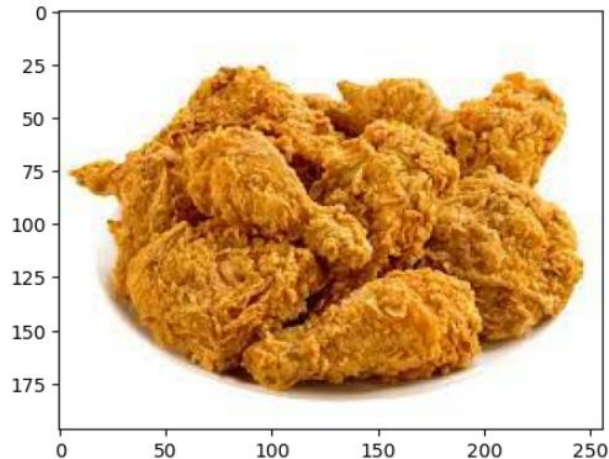
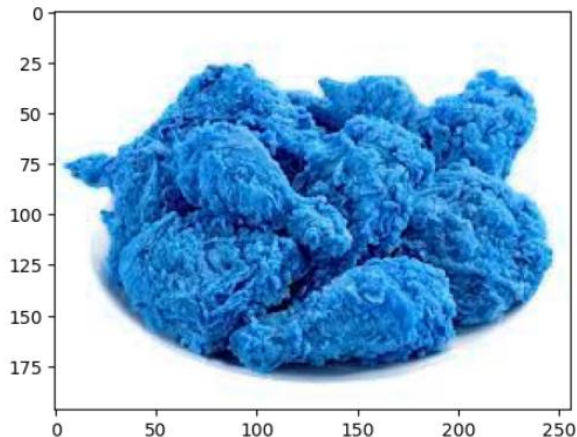
```
image = cv2.imread("download2.jpg")
```

```
RGB_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
plt.figure(figsize=(12, 4))
```

```
plt.imshow(image)
```

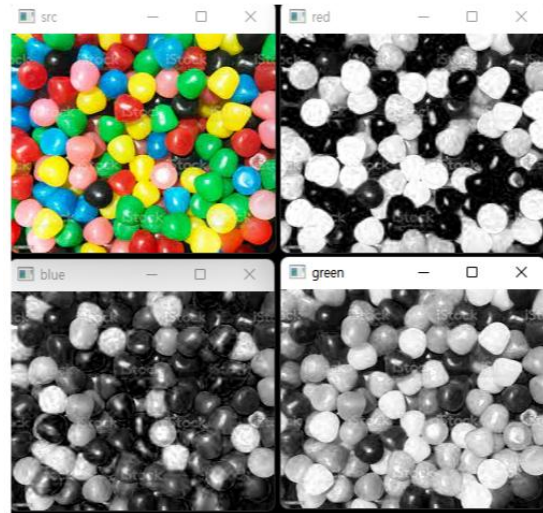
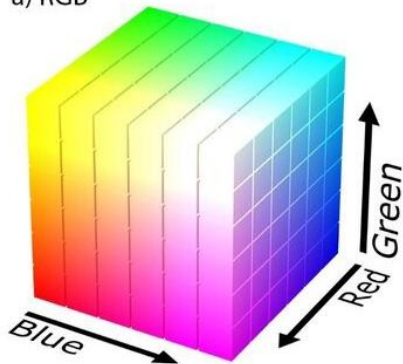
```
plt.show()
```



### 이미지 데이터

- RGB는 Red (빨강), Green (녹색), Blue (파랑)의 약자로, 이 세 가지 기본 색을 조합하여 다양한 색을 표현함
- 대부분의 디지털 화면과 카메라는 RGB 색상 모델을 기반으로 작동
- 인간의 색상 인식과는 다소 거리가 있음: 색상, 명도, 채도를 분리하지 못함 → 계산이 빠름
- 0부터 255의 색상의 전체 강도를 가지고, 총  $255^3$ 의 색을 표현할 수 있음(R, G, B)

a) RGB



# 01 이미지 데이터

## 이미지 데이터의 구조

# R, G, B 채널로 분리

```
RGB_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
R, G, B = cv2.split(RGB_image)
```

# 각 채널을 이미지로 표시

```
plt.figure(figsize=(12, 4))
```

# 빨간 채널

```
plt.subplot(1, 3, 1)
```

```
plt.imshow(R, cmap='Reds')
```

```
plt.title('Red Channel')
```

# 초록 채널

```
plt.subplot(1, 3, 2)
```

```
plt.imshow(G, cmap='Greens')
```

```
plt.title('Green Channel')
```

# 파란 채널

```
plt.subplot(1, 3, 3)
```

```
plt.imshow(B, cmap='Blues')
```

```
plt.title('Blue Channel')
```

```
plt.show()
```

1X3 공간

plt.subplot(1,3,1)	plt.subplot(1,3,2)	plt.subplot(1,3,3)
--------------------	--------------------	--------------------

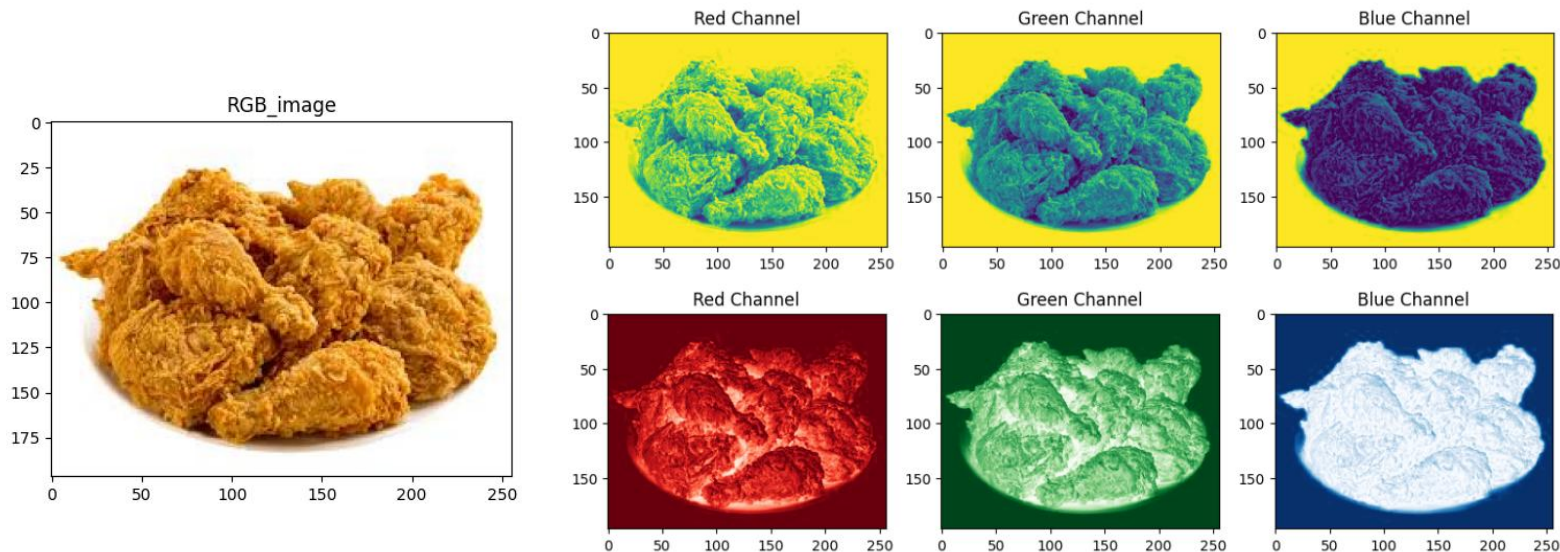
2X3 공간

plt.subplot(2,3,1)	plt.subplot(2,3,2)	plt.subplot(2,3,3)
plt.subplot(2,3,4)	plt.subplot(2,3,5)	plt.subplot(2,3,6)



### 이미지 데이터

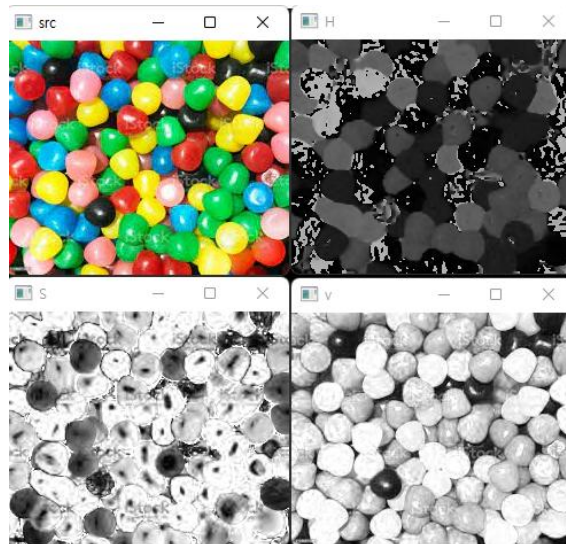
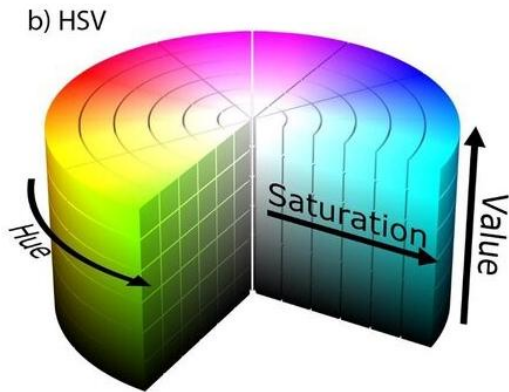
- RGB는 Red (빨강), Green (녹색), Blue (파랑)의 약자로, 이 세 가지 기본 색을 조합하여 다양한 색을 표현함
- 대부분의 디지털 화면과 카메라는 RGB 색상 모델을 기반으로 작동
- 인간의 색상 인식과는 다소 거리가 있음: 색상, 명도, 채도를 분리하지 못함 → 계산이 빠름



**cmap='viridis'  
or 설정X**

### 이미지 데이터

- HSV는 Hue (색상), Saturation (채도), Value (명도)의 약자 → RGB값들의 강도를 통해 계산
- 색상(Hue)은 색의 종류 (빨강, 녹색, 파랑 등)를, 채도(Saturation)는 색의 강렬함, 명도(Value)는 색의 밝기를 나타냄
- 인간의 시각적 인식과 유사하여, 더 직관적인 색상 조정이 가능함 → 계산이 복잡함



# 01 이미지 데이터

## 이미지 데이터의 구조

# 이미지 변환 BGR to RGB

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

# 이미지 변환 RGB to HSV

```
image_hsv = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)
```

# H, S, V 채널로 분리

```
H, S, V = cv2.split(image_hsv)
```

# 각 채널을 그레이스케일 이미지로 표시

```
plt.figure(figsize=(12, 4))
```

```
plt.subplot(1, 3, 1)
```

```
plt.imshow(H, cmap='hsv')
```

```
plt.title('Hue Channel')
```



Hue (색상)

```
plt.subplot(1, 3, 2)
```

```
plt.imshow(S, cmap='gray')
```

```
plt.title('Saturation Channel')
```



Saturation (채도)

```
plt.subplot(1, 3, 3)
```

```
plt.imshow(V, cmap='gray')
```

```
plt.title('Value Channel')
```

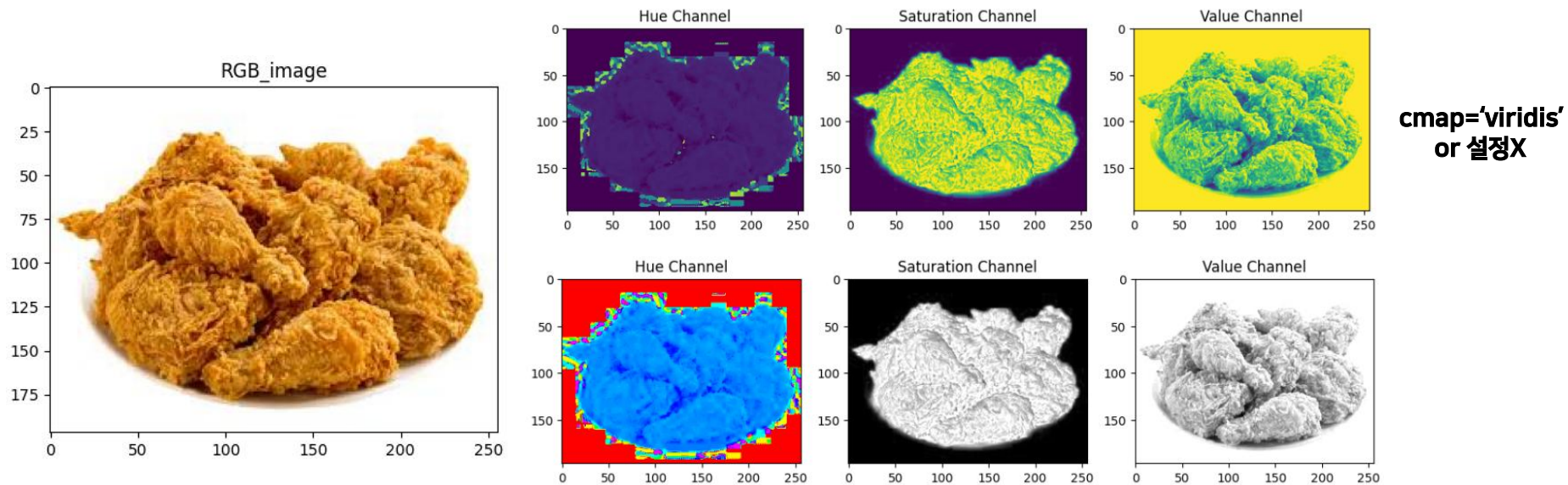


Value (명도)

```
plt.show()
```

### 이미지 데이터

- HSV는 Hue (색상), Saturation (채도), Value (명도)의 약자
- 색상(Hue)은 색의 종류 (빨강, 녹색, 파랑 등)를, 채도(Saturation)는 색의 강렬함, 명도(Value)는 색의 밝기를 나타냄
- 인간의 시각적 인식과 유사하여, 더 직관적인 색상 조정이 가능함 → 계산이 복잡함



### 이미지 데이터

- 핵심은 이미지의 배열 값을 판단하는 것 → 배열의 핵심 특징을 찾고, 이를 연산함



변수1	변수2	변수3	변수4	변수5	변수6	변수7	변수8	변수9	변수10	변수11	변수12	변수13	변수14
45	15	245	244	12	241	56	24	67	55	21	78	126	245



각 데이터 조각이 의미하는 것은 무엇일까요?



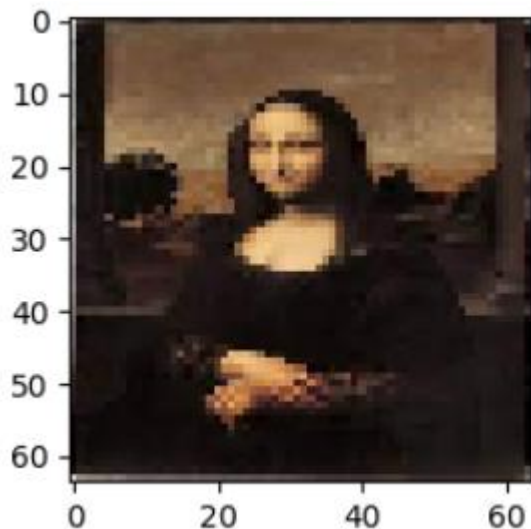
(이미지의 특징을 도출하는 것이 이미지 전처리의 핵심)

### 이미지 리사이징

- 이미지 리사이징 : 큰 이미지는 그만큼 많은 정보가 들어 있고, 계산을 저해하기 때문에 이미지 크기를 줄여 계산을 쉽게 함



원본 이미지



이미지 리사이징(64X64)

#단순 이미지 리사이징

```
resized_image = cv2.resize(image, (64, 64))
```

#이미지 보간 리사이징

```
resized_image = cv2.resize(image, (64, 64),  
interpolation=cv2.INTER_AREA)
```

### 이미지 리사이징

- 이미지 리사이징 : 큰 이미지는 그만큼 많은 정보가 들어 있고, 계산을 저해하기 때문에 이미지 크기를 줄여 계산을 쉽게 함

# 이미지를 64x64으로 크기 조정

```
resized_image = cv2.resize(image, (64, 64), interpolation=cv2.INTER_AREA)
```

```
resized_image = cv2.cvtColor(resized_image , cv2.COLOR_BGR2RGB)
```

# 결과를 시각화하기 위한 코드

```
plt.figure(figsize=(10, 6))
```

```
plt.imshow(resized_image)
```

```
plt.title('Resize')
```

```
plt.show()
```

### 이미지 노이즈 제거

- 노이즈 제거는 이미지 처리에서 널리 사용되는 기술로, 이미지에 부드러운 흐림 효과를 주어 잡음을 줄이고 세부 사항을 부드럽게 만드는 데 사용됨
- 이미지의 배경을 부드럽게 처리하거나 특정 요소를 강조하거나 흐린 값을 복원하기 위해 사용
- 디지털 사진에서 불필요한 잡음을 제거하고 전반적인 이미지 품질을 향상



### 이미지 노이즈 제거

- 가우시안 블러 : 이미지의 각 픽셀 값을 그 주변 픽셀들의 평균값으로 대체하는 방법
- 중간값 필터 : 이미지의 각 픽셀을 해당 픽셀 주변의 픽셀 값들의 중간값으로 대체하는 방법



원본 이미지



가우시안 블러



중간값 필터(Median Filter)

`blurred_image = cv2.GaussianBlur(image, (5, 5), 0)`

`median_filtered_img = cv2.medianBlur(image, 5)`

### 이미지 노이즈 제거

- 가우시안 블러 : 이미지의 각 픽셀 값을 그 주변 픽셀들의 평균값으로 대체하는 방법
- 중간값 필터 : 이미지의 각 픽셀을 해당 픽셀 주변의 픽셀 값들의 중간값으로 대체하는 방법

# 가우시안 블러 적용 (5x5 크기의 커널)

```
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
```

```
blurred_image = cv2.cvtColor(blurred_image , cv2.COLOR_BGR2RGB)
```

# 중간값

```
median_filtered_img = cv2.medianBlur(image, 5)
```

```
median_filtered_img = cv2.cvtColor(median_filtered_img , cv2.COLOR_BGR2RGB)
```

```
plt.figure(figsize=(10, 6))
```

```
plt.subplot(1,2,1)
```

```
plt.imshow(blurred_image )
```

```
plt.title('Blurred')
```

```
plt.subplot(1,2,2)
```

```
plt.imshow(median_filtered_img )
```

```
plt.title('Median')
```

```
plt.show()
```

### 이미지 Gray Scale

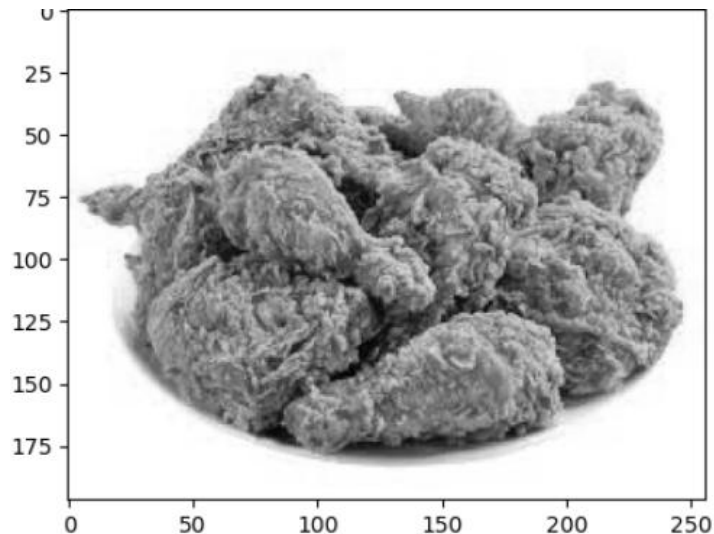
- Gray Scale : 이미지를 흑백으로 만드는 것
- 색상 정보의 손실: HSV 맵을 사용할 때는 각 값이 고유한 색상으로 표현되어 구분이 쉽지만, Gray Scale에서는 모든 값을 밝기로만 표현하기 때문에 세부적인 차이를 구분하기 어려움
- 특히 255와 같이 높은 값은 Gray Scale에서 매우 밝게 표현되며, 0과 1과 같은 낮은 값은 매우 어둡게 표현됨 → 중간 범위의 값들이 시각적으로 덜 두드러질 수 있음
- 데이터 해석의 복잡성: HSV 색상 맵에서는 색상 자체가 데이터의 특성을 나타낼 수 있지만, Gray Scale에서는 이러한 해석이 덜 직관적일 수 있음

### 이미지 Gray Scale

- **복잡성 감소:** 컬러 이미지는 일반적으로 세 개의 채널(RGB)을 가지며, 이로 인해 처리해야 할 데이터의 양이 많음  
그레이스케일로 변환하면 단일 채널로 처리할 수 있어 데이터의 복잡성과 처리 시간이 줄어들어 → 계산의 효율성
- **강조된 대비:** 그레이스케일 이미지에서는 색상 정보가 제거되고 밝기(명도) 정보만 남기 때문에 객체 감지, 텍스트 인식, 경계 감지 등에서 중요한 요소가 됨 → 색상 정보가 불필요하거나 방해가 될 수 있는 분석에 활용
- **일관성 있는 데이터 처리:** 다양한 조명 조건 및 카메라 설정에서 촬영된 이미지의 경우, 그레이스케일을 사용하면 이러한 변동성을 줄이고 더 일관된 데이터 처리가 가능함

### 이미지 Gray Scale

- Gray Scale을 했을 경우 그림을 이해하지 못할까?



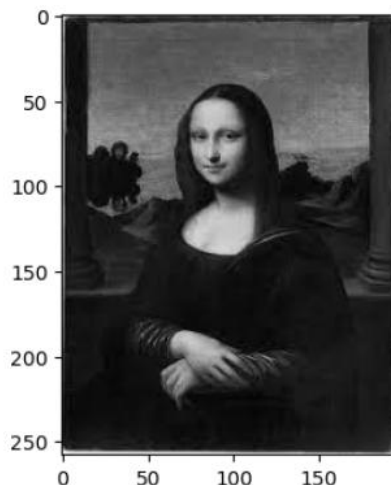
```
image = cv2.imread("download2.jpg")  
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
GRAY_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# 데이터를 이미지로 표시  
plt.figure(figsize=(12, 4))  
plt.imshow(GRAY_image, cmap='gray')  
plt.title("Gray Scale")  
plt.show()
```

실제로 저장되는 값은 gray

### 이미지 정규화

- 정규화 : 계산식을 간단하게 만들어 복잡성을 줄여주는 것



```
[[252 246 232 ... 237 249 253]
 [247 223 173 ... 86 139 241]
 [240 245 142 ... 51 86 249]
 ...
 [253 194 109 ... 98 66 216]
 [254 254 251 ... 94 158 220]
 [249 248 245 ... 222 242 254]]
```

### 이미지 정규화

- 정규화 : 계산식을 간단하게 만들어 복잡성을 줄여주는 것



```
[[0.98823529 0.96470588 0.90980392 .  
 [0.96862745 0.8745098 0.67843137 .  
 [0.94117647 0.96078431 0.55686275 .  
 ...  
 [0.99215686 0.76078431 0.42745098 .  
 [0.99607843 0.99607843 0.98431373 .  
 [0.97647059 0.97254902 0.96078431 .
```

### 이미지 정규화

- 정규화 : 계산식을 간단하게 만들어 복잡성을 줄여주는 것

```
image = cv2.imread('download.jpg')  
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

```
normalized_gray_image = gray_image/255
```

```
plt.figure(figsize=(12, 4))  
plt.imshow(normalized_gray_image, cmap='gray')  
plt.title('Noramlized')  
plt.show()
```

```
print(normalized_gray_image)
```

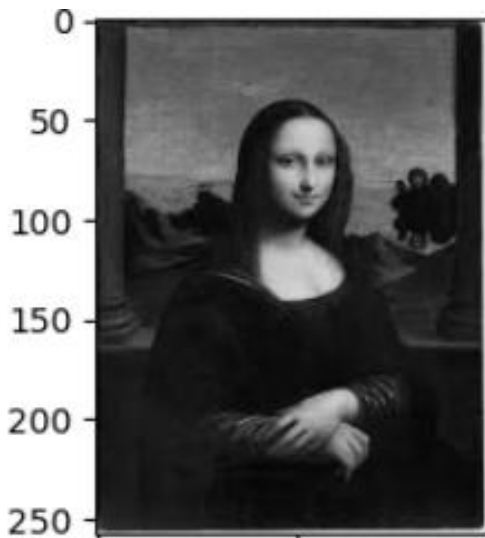


### 이미지 증강

- 이미지 증강 : 학습 데이터의 각도를 다르게 해 학습 데이터의 크기를 늘리는 방법



원본 이미지



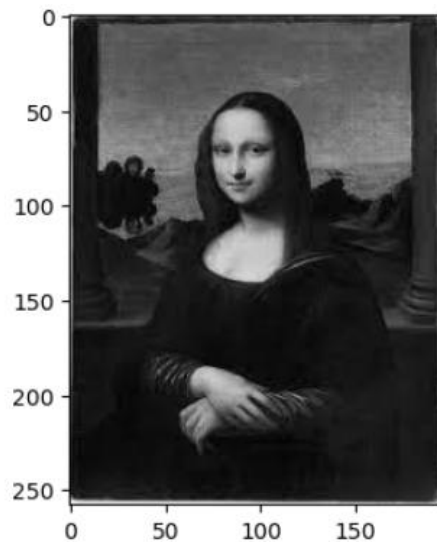
이미지 증강

`flipped_image = cv2.flip(normalized_image, 1)`

값이 0 : 새로 반전  
값이 1 : 좌우 반전  
값이 -1 : 180도 회전

### 이미지 전처리 프로세스

- 이미지 읽기 → Gray Scale → 이미지 축소 → 노이즈 처리 → 정규화



### 이미지 저장

- 이미지 읽기 → Gray Scale → 이미지 축소 → 노이즈 처리 → 정규화



```
cv2.imwrite('random_image.png', random_image)
```

저장할 이름

저장할 이미지

```
cv2.imwrite('random_image.png', cv2.cvtColor(random_image, cv2.COLOR_RGB2BGR))
```

### 이미지 구간 자르기



### 이미지 구간 자르기

```
image_resized = cv2.resize(image, (64, 64), interpolation=cv2.INTER_AREA)
```

```
x, y, width, height = 10, 10, 40, 20
```

```
cropped_image = image_resized[y:y+height, x:x+width]
```

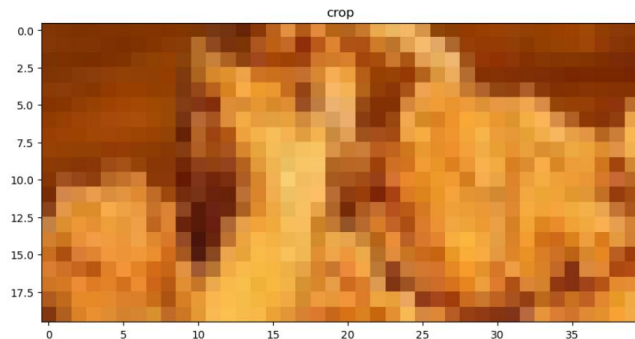
```
cropped_image = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB)
```

```
plt.figure(figsize=(10, 6))
```

```
plt.imshow(cropped_image)
```

```
plt.title('crop')
```

```
plt.show()
```



### 이미지 마스킹

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

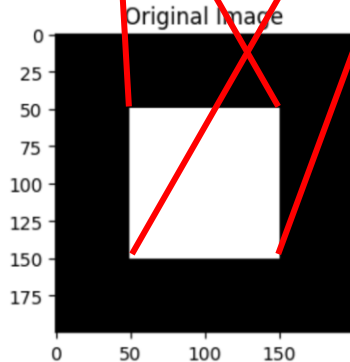
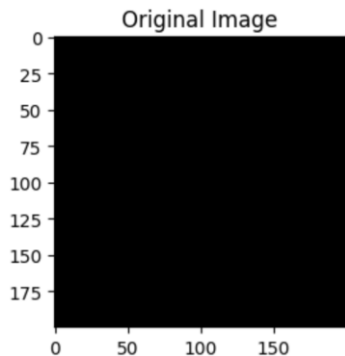
# Step 1: 도형 넣기

```
img_color = np.zeros((200, 200, 3), dtype="uint8")
```

```
img_color = cv2.rectangle(img_color, (50, 50), (150, 150), (0, 255, 0), -1)
```

-1 : 색깔 채우기  
1 : 선만 그리기

사각형의 색깔

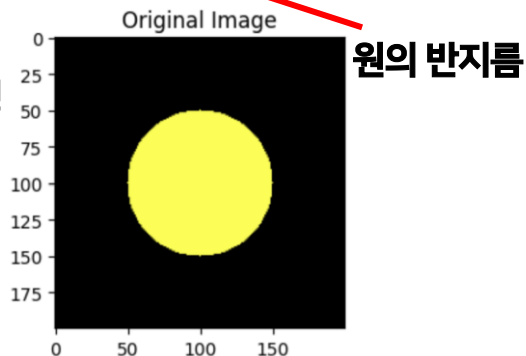
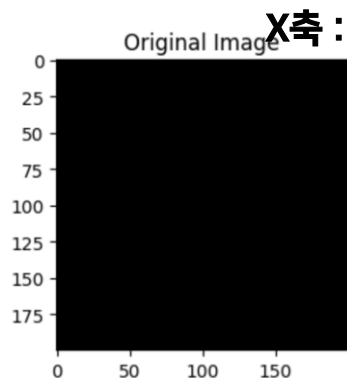


### 이미지 마스킹

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

# Step 1: 도형 넣기

```
mask = np.zeros((200, 200, 3), dtype="uint8")
mask = cv2.circle(mask, (100, 100), 50, (0,255,255), -1)
```



### 이미지 반전-색 대비

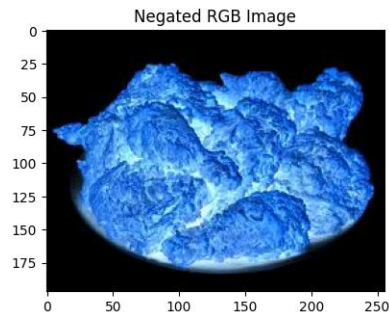
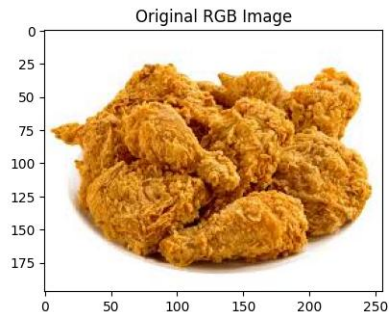
```
image = cv2.imread('download.jpg')
```

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
image_neg = 255 - image_rgb
```

```
plt.figure(figsize=(10, 5))  
plt.subplot(1,2,1)  
plt.imshow(image_rgb)  
plt.title('Original RGB Image')
```

```
plt.subplot(1,2,2)  
plt.imshow(image_neg)  
plt.title('Negated RGB Image')  
plt.show()
```





### 이미지 반전-조명변화

```
image = cv2.imread('download.jpg')
```

```
lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB) #2D 배열로 모양
```

```
plt.figure(figsize=(10, 5))  
plt.subplot(1, 2, 1)  
plt.imshow(image)  
plt.title('Original RGB Image')
```

```
plt.subplot(1, 2, 2)  
plt.imshow(lab_image)  
plt.title('Lab Image')
```

```
plt.show()
```

