Computer Vision

CVI620

Session 9

Overview

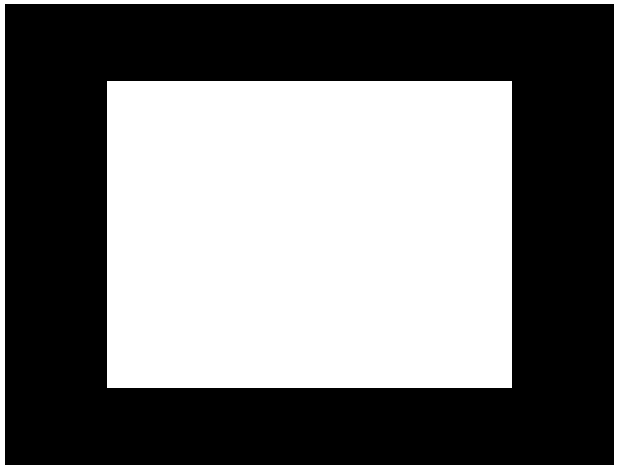
Noise
Noise Types
Denoising Techniques
Filters
Convolution

Agenda

Convolution Cont.
Mean Denoising
Blurring
Edge Detection Algorithms
Feature Extraction
Detection with Colors

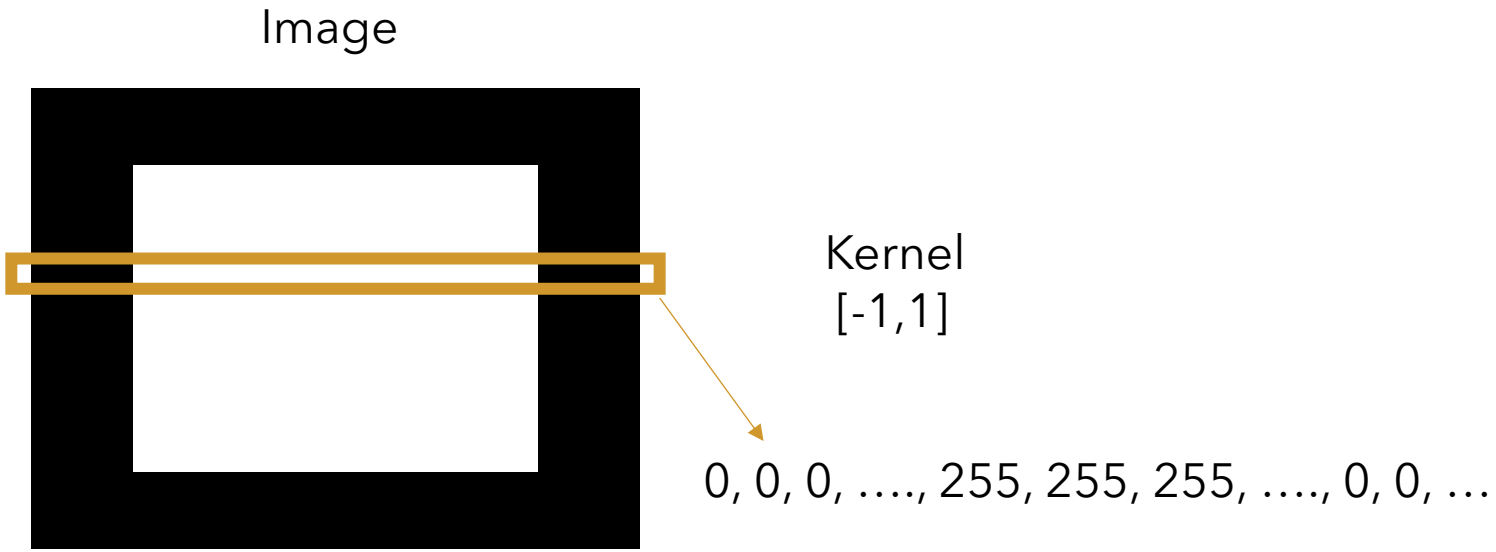
Real Example

Image



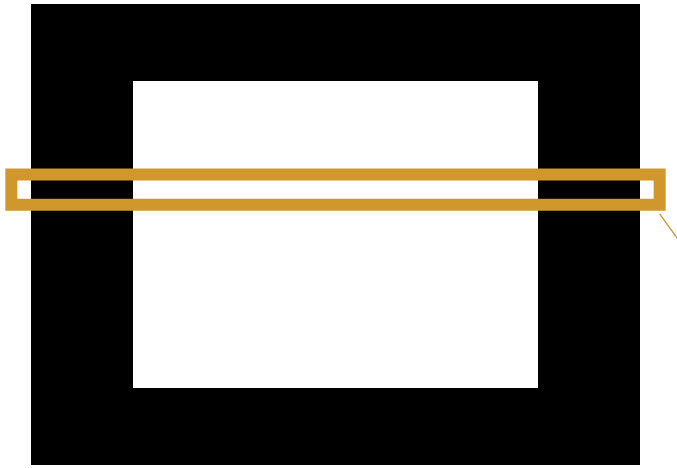
Kernel
[-1,1]

Real Example



Real Example

Image



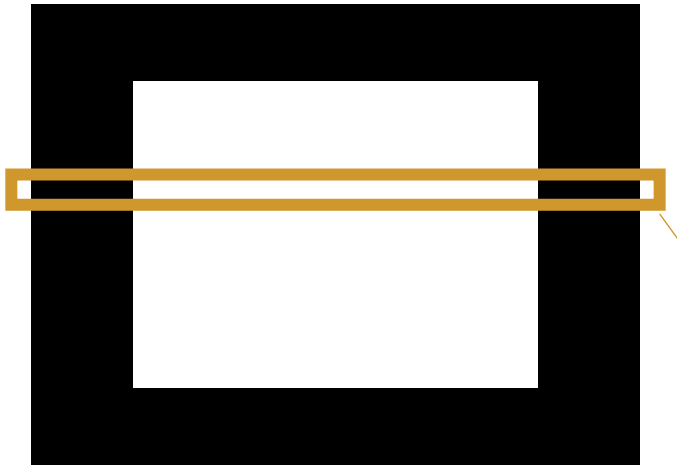
Kernel
[-1,1]

0, 0, 0, ….,0, 255, 255, 255, …., 0, 0, …

0*-1 + 0*1 = 0
0*1 + 255*1 = 255
255*-1 + 255*1 = 0
255*-1 + 255*1 = 0
255*-1 + 255*1 = 0
255*-1 + 0*-1 = -255 → clip(-255) =0

Real Example

Image



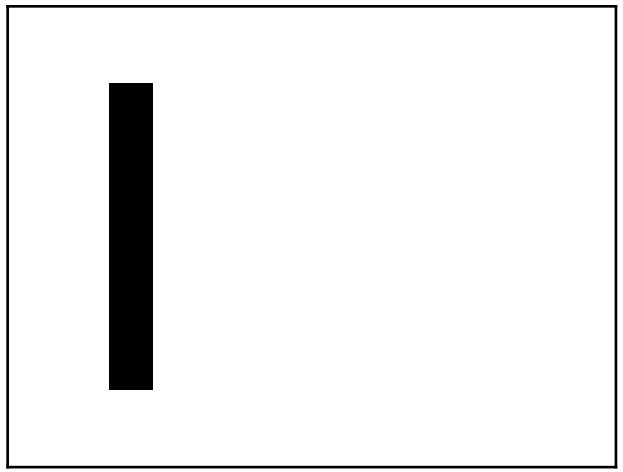
Kernel
[-1,1]

0, 0, 0, ….,0, 255, 255, 255, …., 0, 0, …

0, 0, 0, …, 0, 255, 0, 0, 0, …

Real Example

0, 0, 0, …, 0, 255, 0, 0, 0, …



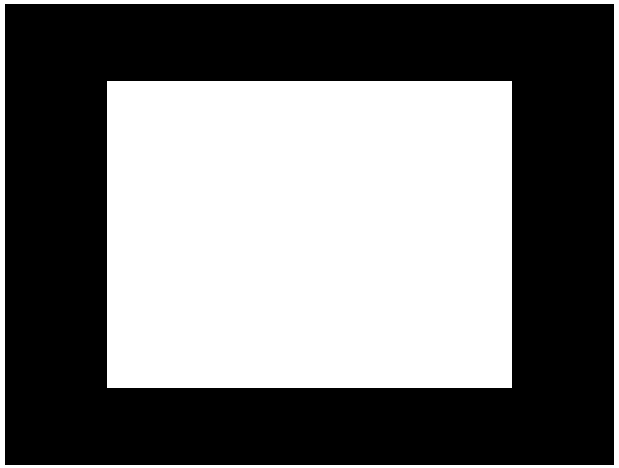
Edge!!

Let’s Code

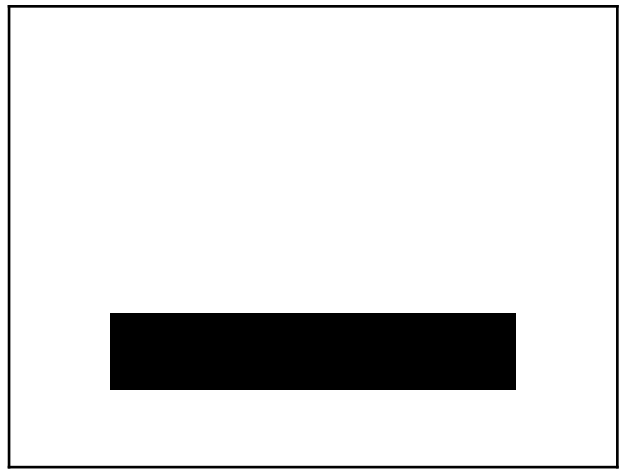
```python
import cv2
import numpy as np


img = cv2.imread("square.png")
kernel = np.array([[-1, 1]])

out_image = cv2.filter2D(img, cv2.CV_8U, kernel)

cv2.imshow("left edge", out_image)
cv2.waitKey(0)
```

Let’s Convolve Vertically

Image



Kernel
[[-1],
[1]]



Example

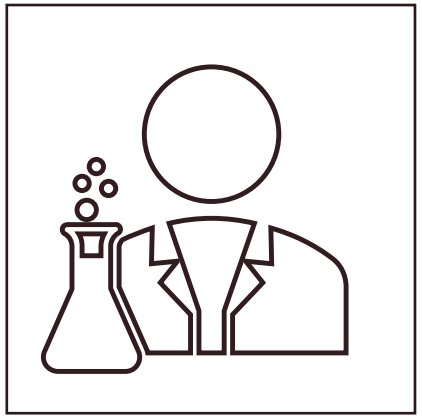
```python
import cv2
import numpy as np


img = cv2.imread("Lenna.png")
kernel1 = np.array([[-1, 1]])
kernel2 = np.array([[-1],
                    [1]])

out_image1 = cv2.filter2D(img, cv2.CV_8U, kernel1)
out_image2 = cv2.filter2D(img, cv2.CV_8U, kernel2)

cv2.imshow("left edge", out_image1)
cv2.imshow("bottom edge", out_image2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

More Complex Convolutions!

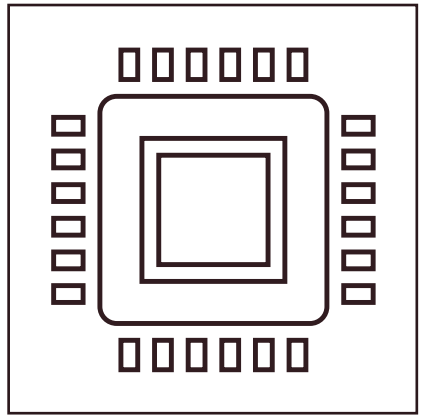
-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

How to achieve these kernels?



Experiments!



ML to learn kernels!

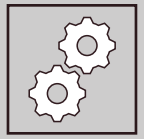
Now let’s get back to denoising!

Mean Denoising

- Replaces each pixel value with the average of its neighboring pixels.
- Smooths the image by averaging pixel intensities.
- Reduces random noise while preserving structure.

Mean Denoising

The filter slides over the image, replacing each pixel with the mean of its neighbors.



Uses a convolution operation with a kernel

Mean Denoising

- Applying a 3×3 Mean Filter
- Kernel Example:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * 1/9$$

- Each pixel value is replaced by the average of surrounding pixels.
- Result: A smoother image with reduced noise.
- Also called blurring!

Example

```python
import cv2
import numpy as np

image = cv2.imread("Lucy.jpg")

if image is None:
    print("ERROR! Image not available...!")

# Gaussian Noise
noise = np.random.normal(0, 25, image.shape).astype('float32')

noisy_image = image + noise
noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')

kernel = np.ones((3,3), dtype= np.float32) / 9
denoised_image = cv2.filter2D(noisy_image, -1, kernel)

cv2.imshow("left edge", denoised_image)
cv2.imshow('frame', noisy_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Blurring

Blurring (or smoothing) is a technique to reduce image noise and detail by averaging pixel values with their neighbors.

- Remove noise
- Preprocess for edge detection
- Reduce details for compression
- Improve feature extraction

- Object detection preprocessing
- Image segmentation
- Face and motion detection
- Artistic effects

Blurring

- Kernel Blurring (cv2.filter2D): Uses a custom averaging kernel of size 25×25 to smooth the image.

- Box Filter (cv2.boxFilter) and Averaging Blur (cv2.blur): Both apply an averaging filter over a region, but cv2.boxFilter() allows normalization control.

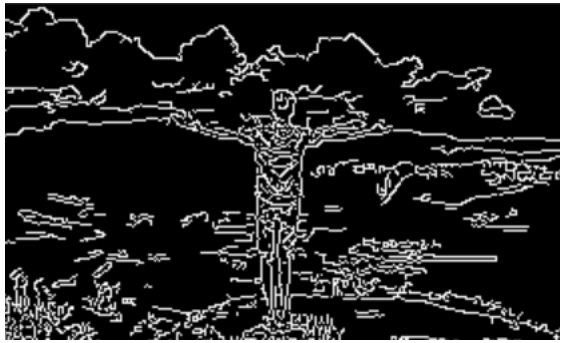
- Gaussian Blur (cv2.GaussianBlur): Uses a Gaussian function for smoothing, making it less harsh than a simple average filter.

- Median Blur (cv2.medianBlur): Replaces each pixel with the median value of its neighborhood.
- Effective for noise reduction, especially for salt-and-pepper noise.

- Bilateral Filter (cv2.bilateralFilter): Reduces noise while preserving edges, unlike other smoothing filters that blur everything.

Edge Detection Algorithms

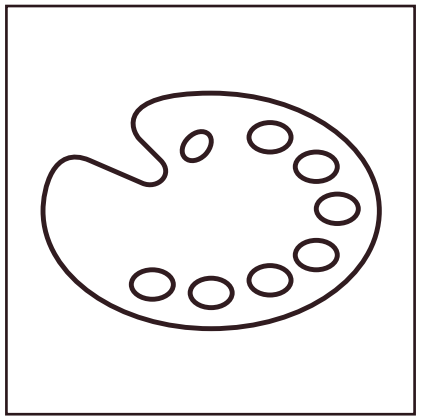
- Sobel Operator
- Prewitt Operator
- Roberts Cross Operator
- Laplacian of Gaussian (LoG)
- Canny Edge Detector
- Difference of Gaussians (DoG)
- Scharr Operator
- Marr-Hildreth Operator
- Frei-Chen Operator
- Kirsch Compass Operator



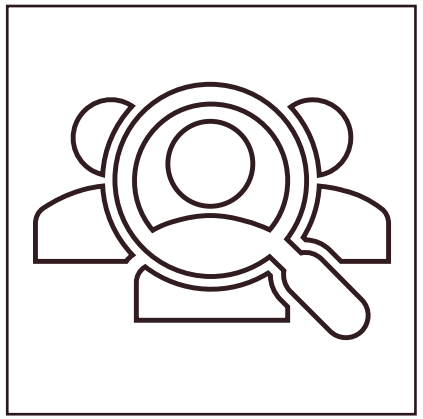
Feature Extraction

What is feature?

Definition



Features are distinct patterns in images that help in identifying objects, textures, and structures.



Used in tasks like object detection, recognition, and segmentation.

Types

- Color Features: RGB, HSV, or other color space information
- Shape: Shapes and format
- Edges: Borders between different regions
- Corners: Points with high curvature
- Blobs: Regions with uniform texture or intensity
- Keypoints: Important points used for matching
- Textures: Patterns formed by pixel intensities

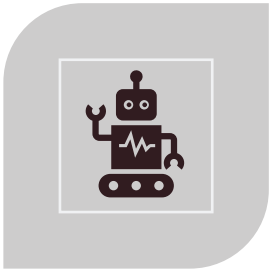
Color Feature Extraction

Why?

HELP IN RECOGNIZING OBJECTS IN IMAGES

IMPROVE ACCURACY IN CLASSIFICATION AND DETECTION TASKS

REDUCE DIMENSIONALITY FOR BETTER PROCESSING

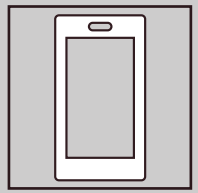


AUTOMATION

Use Case Examples

Traffic light detection

Fruit ripeness classification



Object tracking in augmented reality

Methods

CLASSICAL METHODS: SIFT, SURF, ORB, HOG

DEEP LEARNING BASED FEATURES: CNN FEATURE MAPS

Algorithms

Solution	Description
Thresholding	range to detect specific colors
Histogram Analysis	uses the color histogram of an image to detect dominant colors
Gaussian Mixture Models	probabilistic approach that models pixel distributions for different colors
ML based (K-Means clustering)	groups similar colors into clusters and highlight the dominant color
DL based	a model can be trained to classify and segment colors

Color Pixel Extraction

- Feature extraction from images
- Edge or color detection
- A kind of masking (like segmentation)

Color Detection

```
gray_img1_copy = np.copy(gray_image1)

gray_img1_copy[gray_img1_copy[:, :] < 140] = 0

cv2.imshow("Gray Image", gray_img1_copy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
img2 = cv2.imread(img_path2)
img2 = cv2.resize(img2, (1280, 720))

img_copy = np.copy(img2)

img_copy[(img_copy[:,:,0] > 50) | (img_copy[:,:,1] < 100) | (img_copy[:, :, 2] < 150) ] = 0

img_2 = np.hstack((cv2.resize(img2, (650, 500)), cv2.resize(img_copy, (650, 500)))) # for showing image beside each other
cv2.imshow("Yellow Road Image", img_2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
img3 = cv2.imread(img_path3)
img3 = cv2.resize(img3, (1280, 720))

img3_copy = np.copy(img3)

img3_copy[(img3_copy[:,:,0] > 60) | (img3_copy[:,:,1] > 60) | (img3_copy[:, :, 2] < 80) ] = 0

img_3 = np.hstack(( cv2.resize(img3, (500, 500)), cv2.resize(img3_copy, (500, 500)) ))

cv2.imshow("Color Image VS Color Extracted Image", img_3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Not Scalable or User Friendly

Steps for Color Extraction

- Convert the image to the desired color space
- Define color ranges (thresholding)
- Use masking to extract specific colors
- Apply contour detection for object identification

Functions

- cv2.namedWindow()
- cv2.resizeWindow()
- cv2.createTrackbar()
- cv2.getTrackbarPos()
- cv2.inRange()
- cv2.bitwise_and()
- HSV

cv2.namedWindow

- Creates a window for displaying images
- Allows customization of window properties

```
cv2.namedWindow('WindowName', flags)
```

cv2.WINDOW_NORMAL – Resizable window
cv2.WINDOW_AUTOSIZE – Fixed size based on the image
cv2.WINDOW_FULLSCREEN – Fullscreen mode

cv2.resizeWindow()

- Resizes an existing OpenCV window
- Works only if the window was created with cv2.WINDOW_NORMAL

```
cv2.resizeWindow('WindowName', width, height)
```

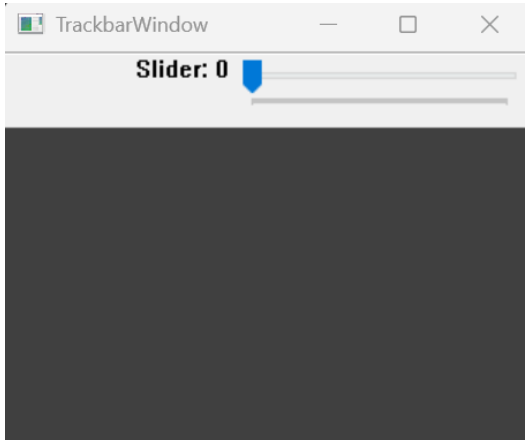
```
import cv2
cv2.namedWindow('MyWindow', cv2.WINDOW_NORMAL)
cv2.resizeWindow('MyWindow', 100, 600)
image = cv2.imread('S9_lambo.png')
cv2.imshow('MyWindow', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.createTrackbar()

- Creates a trackbar (slider) in an OpenCV window
- Used for interactive parameter tuning
- Useful for real-time parameter adjustments

```
cv2.createTrackbar(trackbar_name, window_name, min_value, max_value, callback_function)
```

trackbar_name: Name of the slider.
window_name: The window where the trackbar appears.
min_value: Minimum value of the trackbar.
max_value: Maximum value of the trackbar.
callback_function: Function called when the trackbar value changes.



```
cv2.getTrackbarPos(trackbar_name, window_name)
```

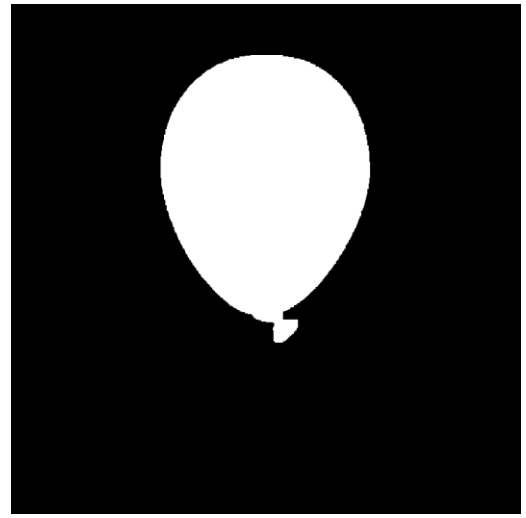
cv2.getTrackbarPos()

- Retrieves the current position (value) of a trackbar

trackbar_name: Name of the trackbar.
window_name: Name of the window where the trackbar is attached.

```python
import cv2
import numpy as np

def on_trackbar(val):
    pass

cv2.namedWindow('TrackbarWindow')
cv2.createTrackbar('Slider', 'TrackbarWindow', 0, 255, on_trackbar)

while True:
    value = cv2.getTrackbarPos('Slider', 'TrackbarWindow')
    print(f"Trackbar Value: {value}")
    if cv2.waitKey(1) & 0xFF == 27: #ESC
        break

cv2.destroyAllWindows()
```

cv2.inRange()



- Performs color thresholding to create a binary mask
- Identifies pixels within a specified color range

```
mask = cv2.inRange(image, lower_bound, upper_bound)
```

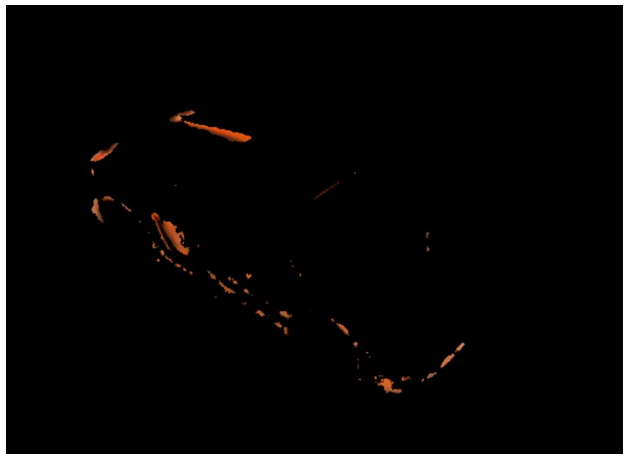
image: Source image (in HSV, RGB, or grayscale).
lower_bound: Lower limit of the color range.
upper_bound: Upper limit of the color range.
mask: Output binary image (white for pixels within range, black otherwise).

```python
import cv2
import numpy as np

image = cv2.imread('balloon.png')
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

lower_red = np.array([0, 120, 70])
upper_red = np.array([10, 255, 255])

mask = cv2.inRange(hsv, lower_red, upper_red)

cv2.imshow('Original', image)
cv2.imshow('Mask', mask)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

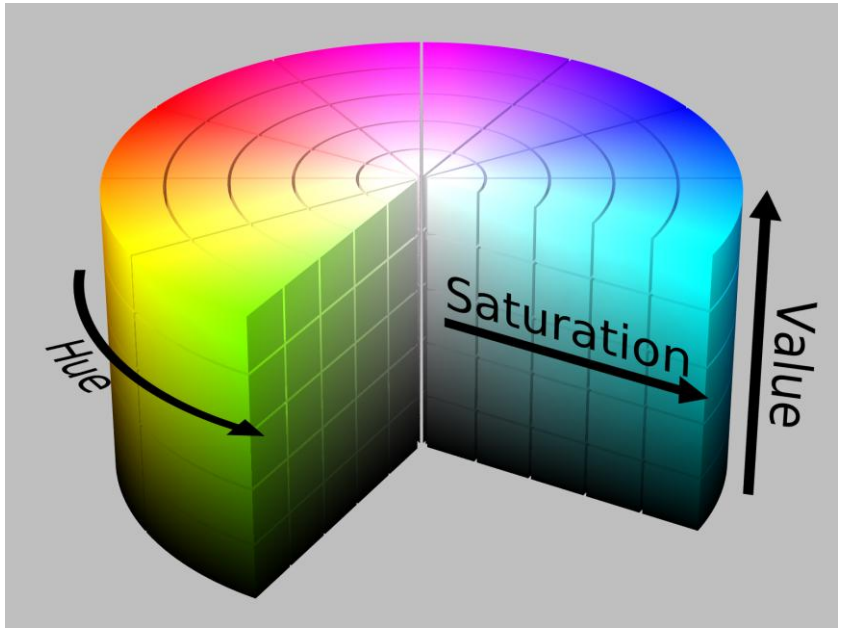
cv2.bitwise_and()

- Performs a bitwise AND operation on two images or an image and a mask
- Used for extracting regions of interest (ROI)

```
result = cv2.bitwise_and(image1, image2, mask=mask)
```



HSV



- Used for better color segmentation
- Hue (H): Represents color (0°-360°)
  - Red = 0°, Green = 120°, Blue = 240°
- Saturation (S): Intensity of the color (0-100%)
  - 0% = grayscale, 100% = pure color
- Value (V): Brightness level (0-100%)
  - 0% = black, 100% = full brightness
- in OpenCV, the Hue (H) value ranges from 0 to 179 instead of 0 to 360 degrees. This is because OpenCV stores images in 8-bit format (0-255 per channel), and to fit the Hue component into 1 byte efficiently, it is scaled down to 0-179.


Color Detection Project

```python
import cv2
import numpy as np

def empty(a):
    pass

path = 'S9_lambo.png'
cv2.namedWindow("TrackBars")
cv2.resizeWindow("TrackBars",640,240)
cv2.createTrackbar("Hue Min","TrackBars",0,179,empty)
cv2.createTrackbar("Hue Max","TrackBars",19,179,empty)
cv2.createTrackbar("Sat Min","TrackBars",110,255,empty)
cv2.createTrackbar("Sat Max","TrackBars",240,255,empty)
cv2.createTrackbar("Val Min","TrackBars",153,255,empty)
cv2.createTrackbar("Val Max","TrackBars",255,255,empty)

while True:
    img = cv2.imread(path)
    imgHSV = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
    h_min = cv2.getTrackbarPos("Hue Min","TrackBars")
    h_max = cv2.getTrackbarPos("Hue Max", "TrackBars")
    s_min = cv2.getTrackbarPos("Sat Min", "TrackBars")
    s_max = cv2.getTrackbarPos("Sat Max", "TrackBars")
    v_min = cv2.getTrackbarPos("Val Min", "TrackBars")
    v_max = cv2.getTrackbarPos("Val Max", "TrackBars")
    print(h_min,h_max,s_min,s_max,v_min,v_max)
    lower = np.array([h_min,s_min,v_min])
    upper = np.array([h_max,s_max,v_max])
    mask = cv2.inRange(imgHSV,lower,upper)
    imgResult = cv2.bitwise_and(img,img,mask=mask)

    cv2.imshow("Original",img)
    cv2.imshow("HSV",imgHSV)
    cv2.imshow("Mask", mask)
    cv2.imshow("Result", imgResult)
    cv2.waitKey(1)
```