


# Computer Vision

CVI620

Session 9  
02/2025

---

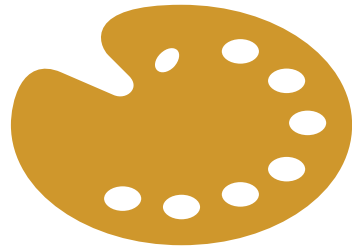


# Feature Extraction

---

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

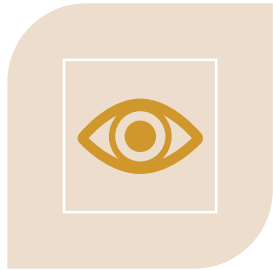
Textures: Patterns formed by pixel intensities

# Color Feature Extraction

---

# Why?

---



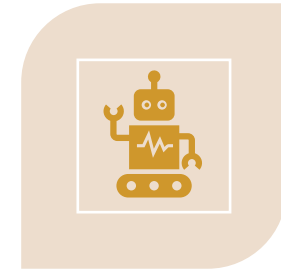
HELP IN RECOGNIZING  
OBJECTS IN IMAGES



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 Detection

## Review

```
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)
```

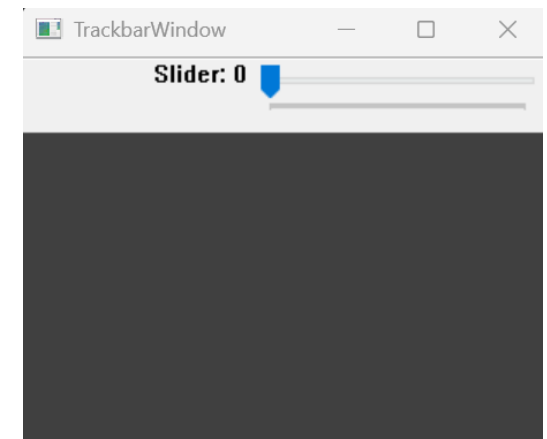
trackbar\_name: Name of the slider.

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()

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.

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

```
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

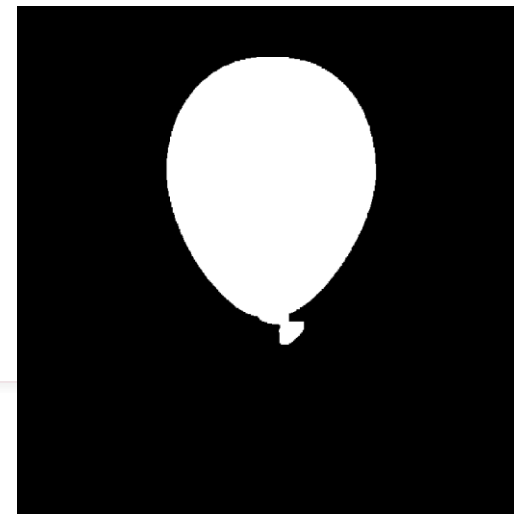
```
mask = cv2.inRange(image, lower_bound, upper_bound)
```

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).



```
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)

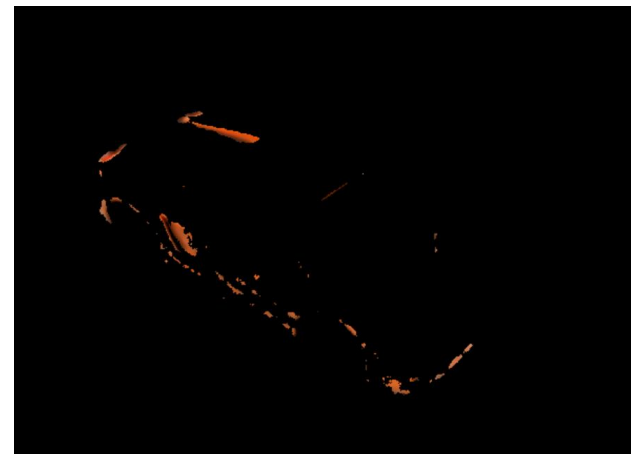
cv2.imshow('Original', image)
cv2.imshow('Mask', mask)

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

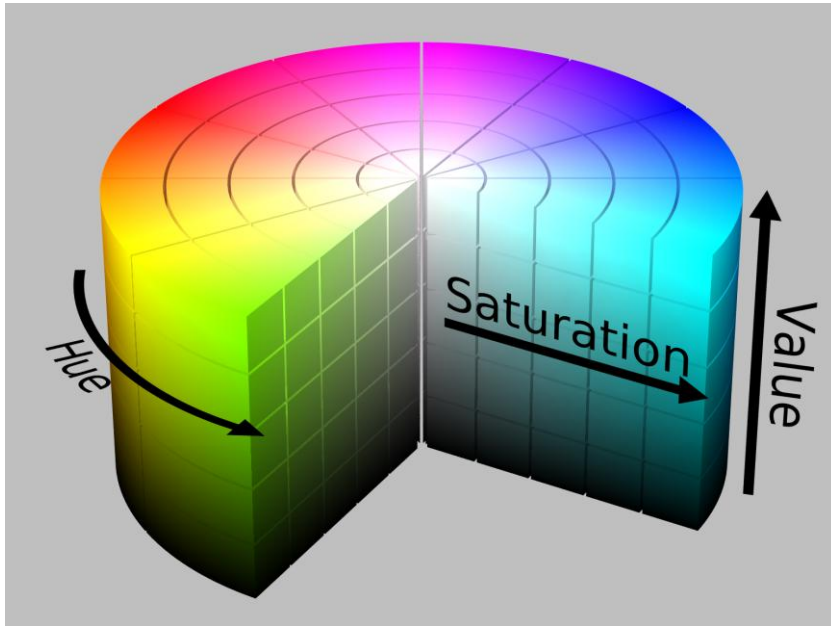
# cv2.bitwise\_and()

- 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 ( $0^{\circ}$ - $360^{\circ}$ )
  - Red =  $0^{\circ}$ , Green =  $120^{\circ}$ , Blue =  $240^{\circ}$
- 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

---



```
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)
```



# Shape Feature Extraction



```

import cv2
import numpy as np

def getContours(img):
    contours,hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    for cnt in contours:
        area = cv2.contourArea(cnt)
        print(area)
        if area>500:
            cv2.drawContours(imgContour, cnt, -1, (255, 0, 0), 3)
            peri = cv2.arcLength(cnt, True)

            approx = cv2.approxPolyDP(cnt, 0.02*peri, True)
            print(len(approx))
            objCor = len(approx)
            x, y, w, h = cv2.boundingRect(approx)

            if objCor == 3: objectType = "Tri"
            elif objCor == 4:
                aspRatio = w / float(h)
                if aspRatio > 0.98 and aspRatio < 1.03: objectType = "Square"
                else:objectType = "Rectangle"
            elif objCor > 4: objectType = "Circles"
            else: objectType = "None"

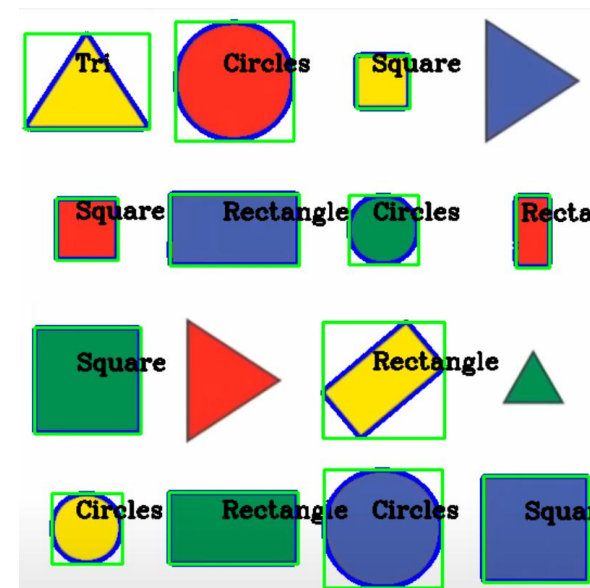
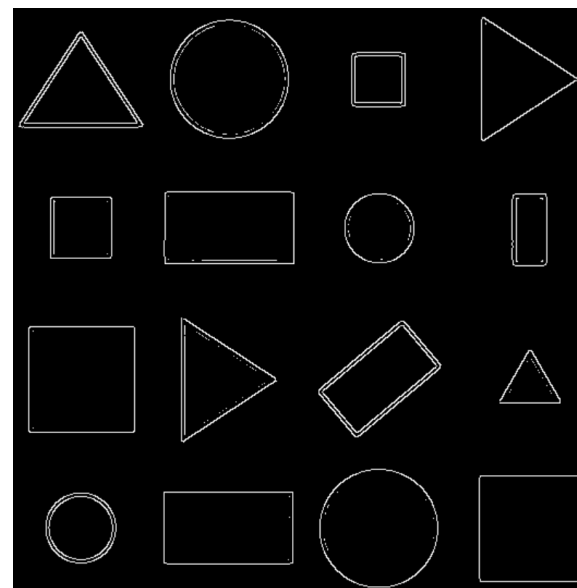
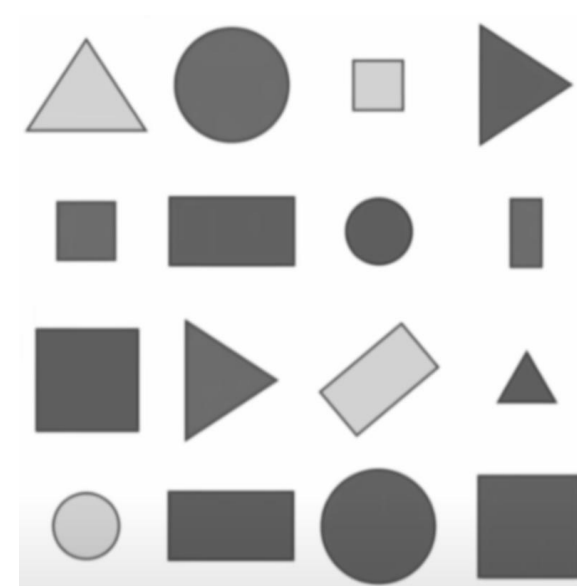
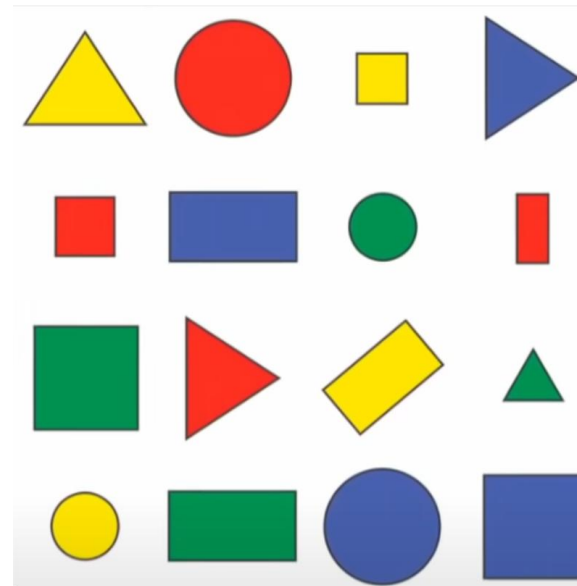
            cv2.rectangle(imgContour, (x,y), (x+w, y+h), (0, 255, 0), 2)
            cv2.putText(imgContour,objectType,
                        (x + (w//2) - 10, y + (h//2) - 10), cv2.FONT_HERSHEY_COMPLEX, 0.7,
                        (0, 0, 0), 2)

path = 'S9_shapes.png'
img = cv2.imread(path)
imgContour = img.copy()
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray, (7,7), 1)
imgCanny = cv2.Canny(imgBlur, 50, 50)
getContours(imgCanny)

imgBlank = np.zeros_like(img)
cv2.imshow("Original", img)
cv2.imshow("HSV", imgContour)
cv2.imshow("Mask", imgBlur)
cv2.imshow("Result", imgCanny)
cv2.waitKey(1)

cv2.waitKey(0)

```



# 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

# Next Sessions



Friday: Implement 1 or 2 projects



Tuesday: Start Machine Learning