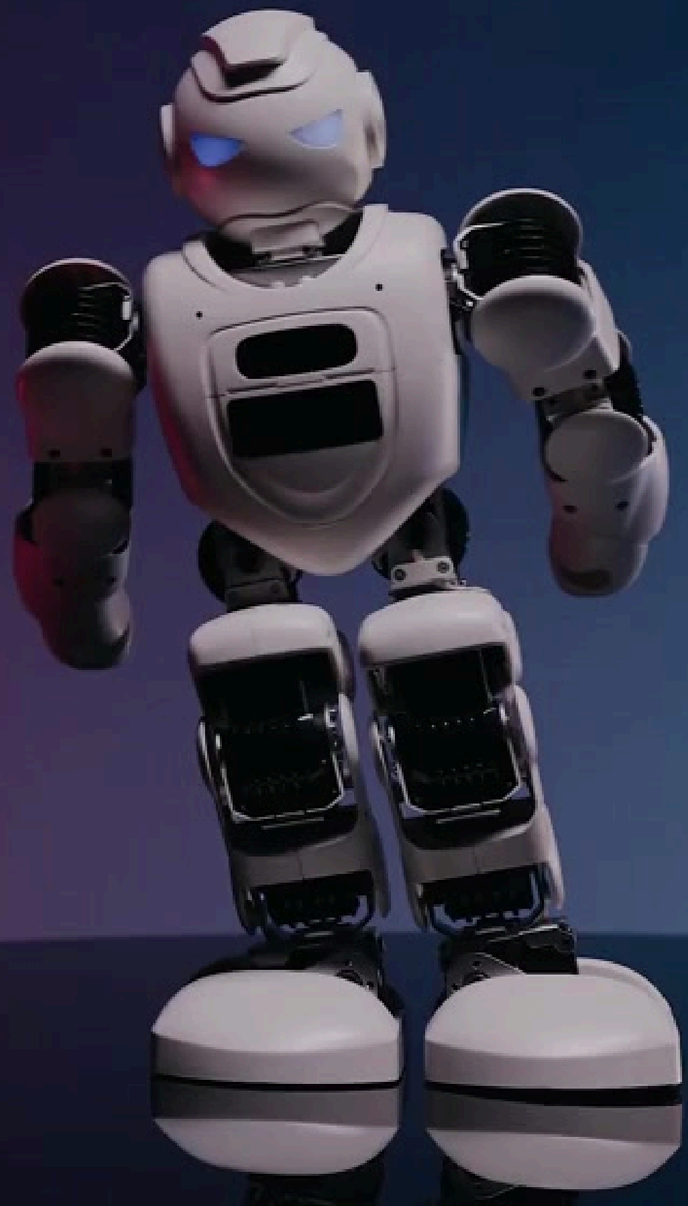
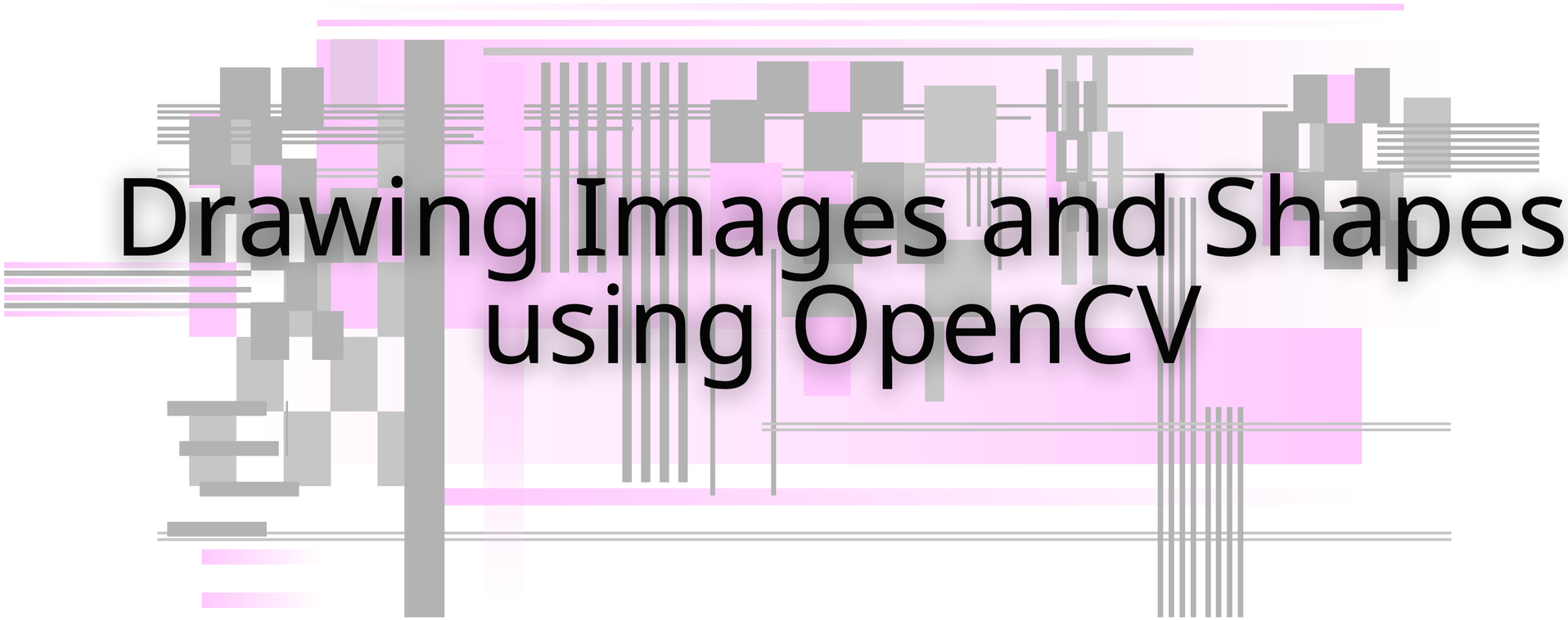
WEEK 3

DEEP LEARNING FOR COMPUTER VISION

UNLIMITED



Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**



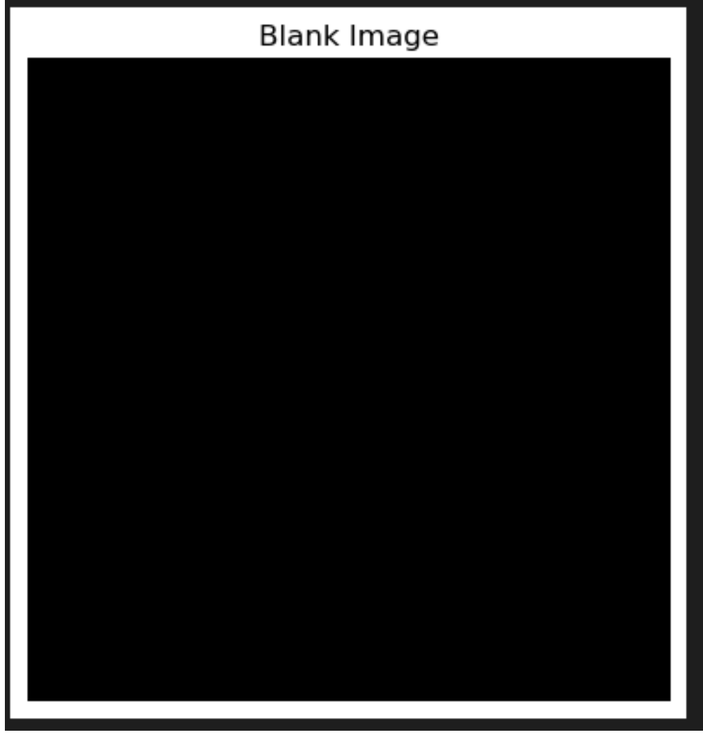
Drawing Images and Shapes using OpenCV

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

def show_image(image, title="Image"):
    # Function to display image using matplotlib
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show()
```

✓ 0.3s

```python
# Create a blank image - a 500x500 image with 3 color channels (BGR) filled with black
blank_image = np.zeros((500, 500, 3), dtype='uint8')
show_image(blank_image, title="Blank Image")
```

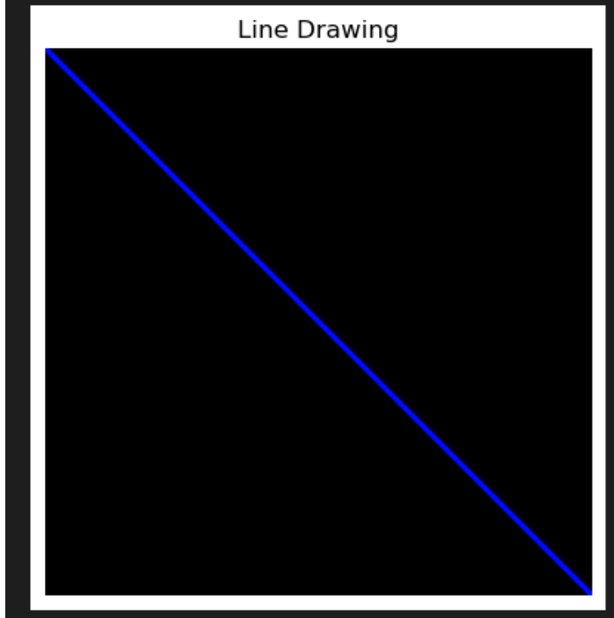
✓ 0.0s



Step 2: Drawing a Line

Let's draw a line from the top-left corner to the bottom-right corner.

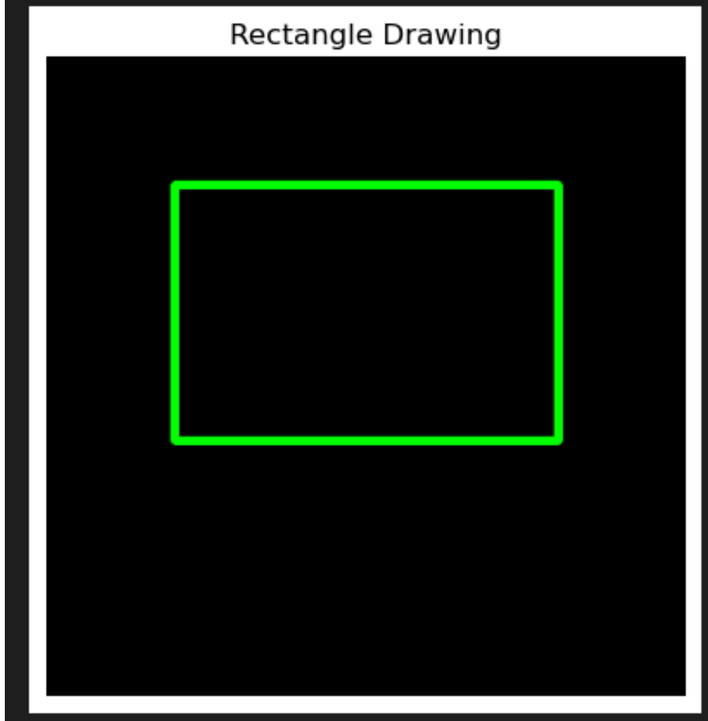
```
# Draw a blue line from top-left to bottom-right
line_image = blank_image.copy()
cv2.line(line_image, (0, 0), (500, 500), (255, 0, 0), thickness=3)
show_image(line_image, title="Line Drawing")
```



Explanation of `cv2.line` Function:

- **Function**: `cv2.line(image, start_point, end_point, color, thickness)`
- **Parameters:**
  - `image`: The image on which to draw.
  - `start_point`: Tuple representing the starting point of the line (x, y).
  - `end_point`: Tuple representing the ending point of the line (x, y).
  - `color`: BGR tuple for the color of the line (e.g., `(255, 0, 0)` for blue).
  - `thickness`: Thickness of the line in pixels.

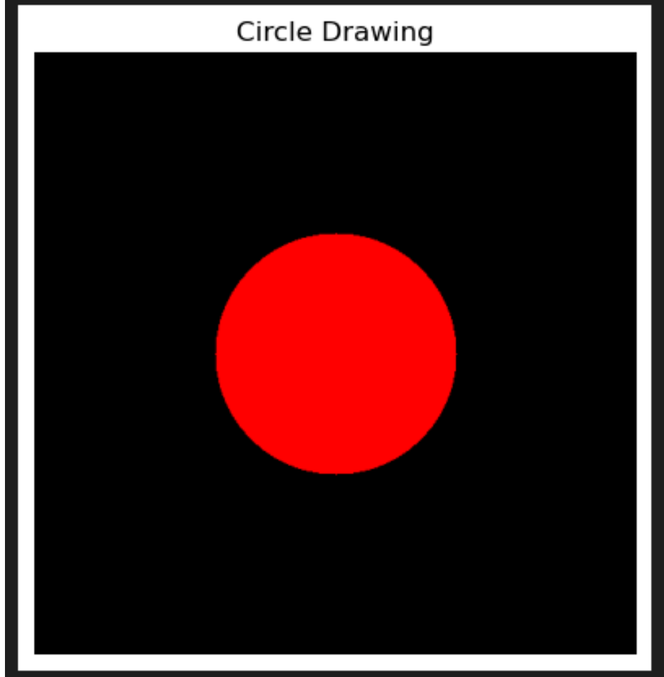
```
# Draw a green rectangle
rectangle_image = blank_image.copy()
cv2.rectangle(rectangle_image, (100, 100), (400, 300), (0, 255, 0), thickness=5)
show_image(rectangle_image, title="Rectangle Drawing")
```



Explanation of `cv2.rectangle` Function:

- **Function**: `cv2.rectangle(image, top_left, bottom_right, color, thickness)`
- **Parameters**:
  - `image`: The image on which to draw.
  - `top_left`: Tuple representing the top-left corner (x, y).
  - `bottom_right`: Tuple representing the bottom-right corner (x, y).
  - `color`: BGR tuple for the color of the rectangle (e.g., `(0, 255, 0)` for green).
  - `thickness`: Thickness of the rectangle's border. Use `-1` to fill the rectangle.

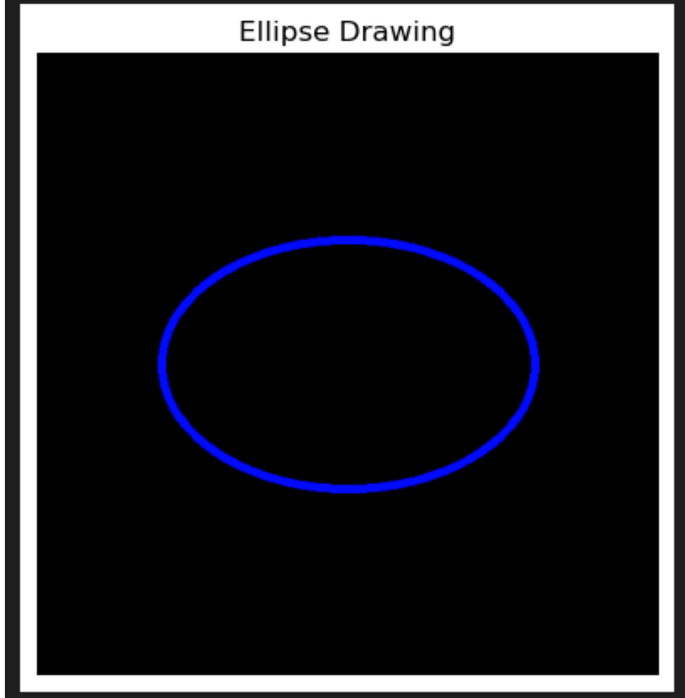
```python
# Draw a red circle
# thickness = -1 fills the circle
circle_image = blank_image.copy()
cv2.circle(circle_image, (250, 250), radius=100, color=(0, 0, 255), thickness=-1)
show_image(circle_image, title="Circle Drawing")
```



Explanation of `cv2.circle` Function:

- **Function**: `cv2.circle(image, center, radius, color, thickness)`
- **Parameters**:
  - `image`: The image on which to draw.
  - `center`: Tuple representing the center of the circle (x, y).
  - `radius`: Radius of the circle.
  - `color`: BGR tuple for the color of the circle (e.g., `(0, 0, 255)` for red).
  - `thickness`: Thickness of the circle's border. Use `-1` to fill the circle.

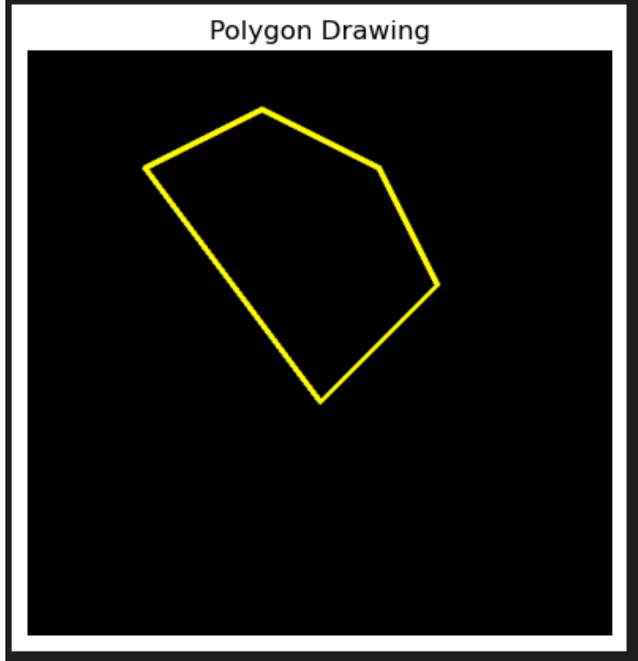
```python
# Draw an ellipse
ellipse_image = blank_image.copy()
cv2.ellipse(ellipse_image, (250, 250), (150, 100), 0, 0, 360, (255, 0, 0), thickness=5)
show_image(ellipse_image, title="Ellipse Drawing")
```



Explanation of `cv2.ellipse` Function:

- **Function**: `cv2.ellipse(image, center, axes, angle, startAngle, endAngle, color, thickness)`
- **Parameters**:
  - `image`: The image on which to draw.
  - `center`: Tuple representing the center of the ellipse (x, y).
  - `axes`: Tuple representing the lengths of the ellipse axes (major axis, minor axis).
  - `angle`: Angle of rotation of the ellipse in degrees.
  - `startAngle`: Starting angle of the elliptic arc in degrees.
  - `endAngle`: Ending angle of the elliptic arc in degrees.
  - `color`: BGR tuple for the color of the ellipse.
  - `thickness`: Thickness of the ellipse's border.

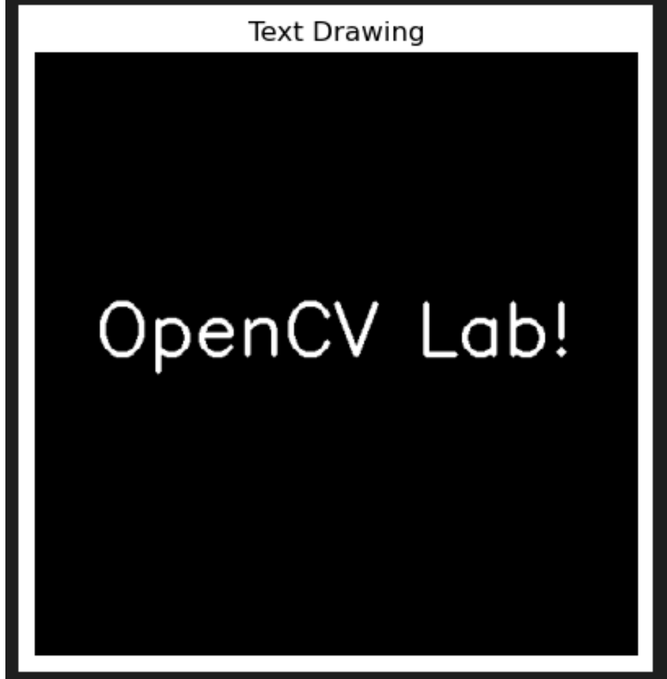
```python
# Draw a closed polygon
polygon_image = blank_image.copy()
points = np.array([[100, 100], [200, 50], [300, 100], [350, 200], [250, 300]], np.int32)
points = points.reshape((-1, 1, 2))
cv2.polylines(polygon_image, [points], isClosed=True, color=(0, 255, 255), thickness=3)
show_image(polygon_image, title="Polygon Drawing")
```



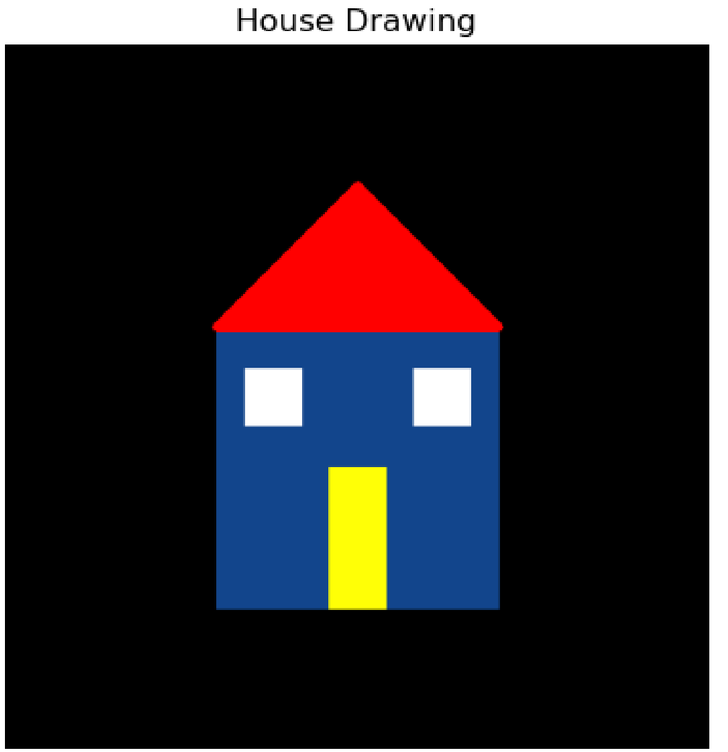
Explanation of `cv2.polylines` Function:

- **Function**: `cv2.polylines(image, points, isClosed, color, thickness)`
- **Parameters**:
  - `image`: The image on which to draw.
  - `points`: Array of points representing the vertices of the polygon.
  - `isClosed`: Boolean value specifying whether the polygon should be closed or not.
  - `color`: BGR tuple for the color of the polygon.
  - `thickness`: Thickness of the polygon's border.

```python
# Add text to the image
text_image = blank_image.copy()
cv2.putText(text_image, 'OpenCV Lab!', (50, 250), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 3)
show_image(text_image, title="Text Drawing")
```

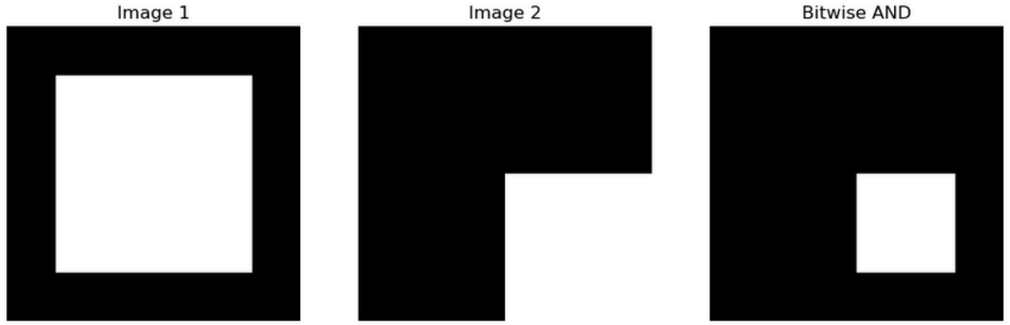


```python
# Draw a simple house
house_image = blank_image.copy()

# Draw the main rectangle for the house
cv2.rectangle(house_image, (150, 200), (350, 400), (139, 69, 19), thickness=-1)  # Brown filled rectangle

# Draw the roof using a triangle
roof_points = np.array([[150, 200], [350, 200], [250, 100]], np.int32)
roof_points = roof_points.reshape((-1, 1, 2))
cv2.polylines(house_image, [roof_points], isClosed=True, color=(0, 0, 255), thickness=5)  # Red outline for the roof
cv2.fillPoly(house_image, [roof_points], color=(0, 0, 255))  # Fill the roof with red color

# Draw the door
cv2.rectangle(house_image, (230, 300), (270, 400), (0, 255, 255), thickness=-1)  # Yellow filled rectangle for the door

# Draw windows
cv2.rectangle(house_image, (170, 230), (210, 270), (255, 255, 255), thickness=-1)  # White filled rectangle for left window
cv2.rectangle(house_image, (290, 230), (330, 270), (255, 255, 255), thickness=-1)  # White filled rectangle for right window

show_image(house_image, title="House Drawing")
```

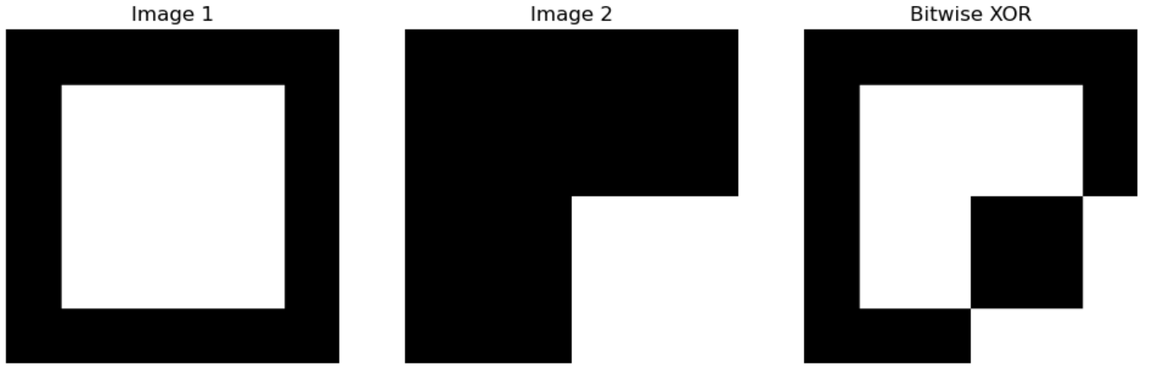


Bitwise Operations

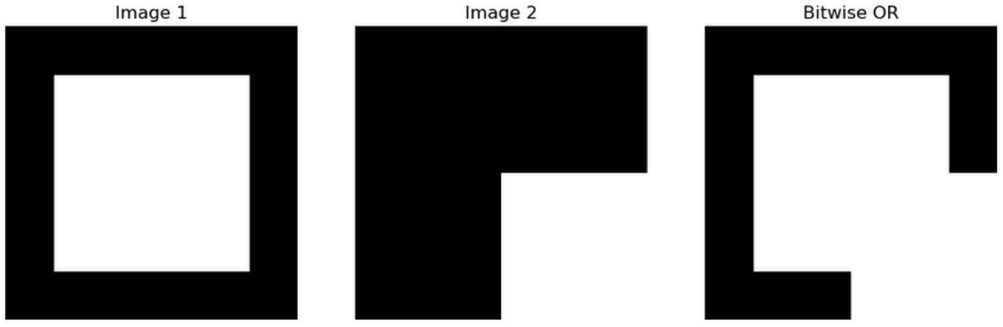
```python
# Bitwise AND
and_result = cv2.bitwise_and(img1, img2)
```



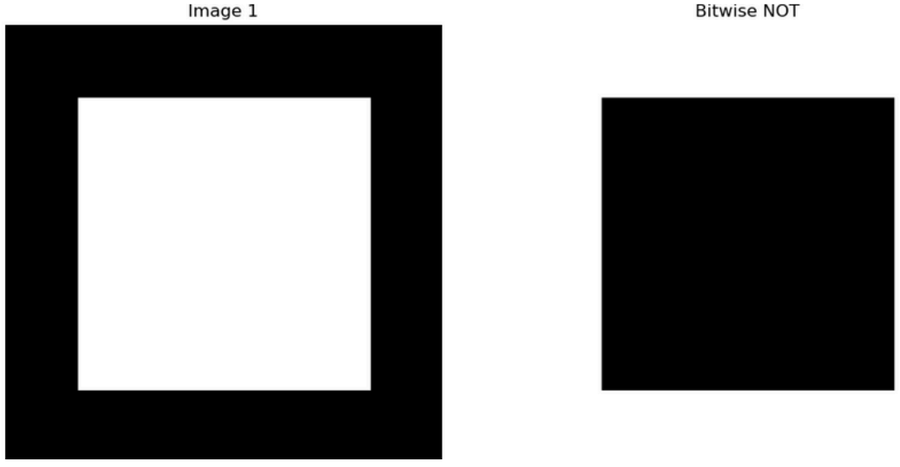

```python
# Bitwise XOR
xor_result = cv2.bitwise_xor(img1, img2)
```




```python
# Bitwise OR
or_result = cv2.bitwise_or(img1, img2)
```




```python
# Bitwise NOT
not_result = cv2.bitwise_not(img1)
```

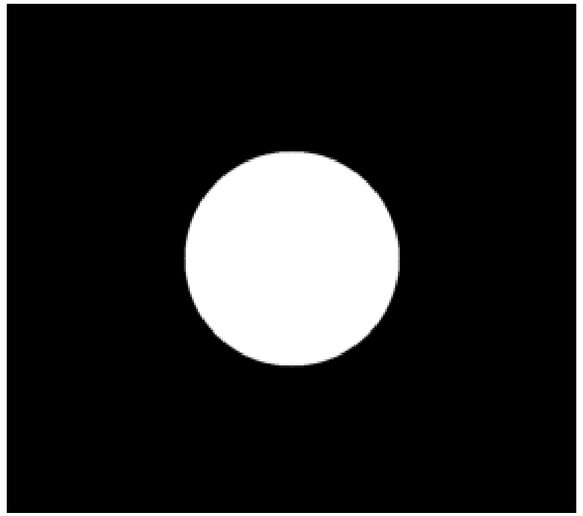


Practical Example - Masking an Image

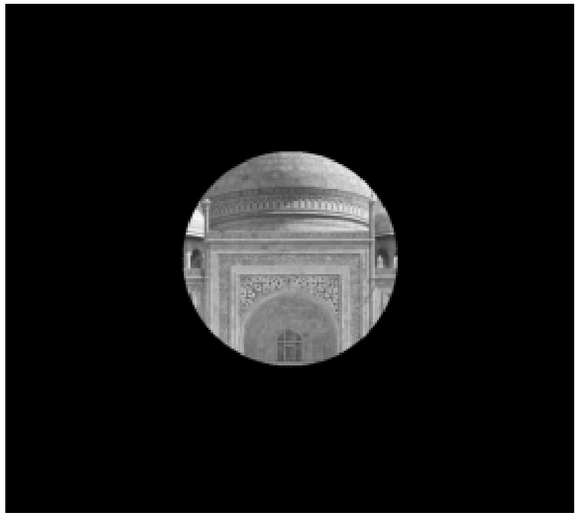
```python
# Load a sample image
image = cv2.imread(cv2.samples.findFile('./images/Taj_Mahal.jpg'), cv2.IMREAD_GRAYSCALE)

# Create a circular mask
mask = np.zeros_like(image, dtype=np.uint8)
cv2.circle(mask, (mask.shape[1] // 2, mask.shape[0] // 2), 150, 255, -1)

# Apply the mask using Bitwise AND
masked_image = cv2.bitwise_and(image, image, mask=mask)
```

Original Image

Mask

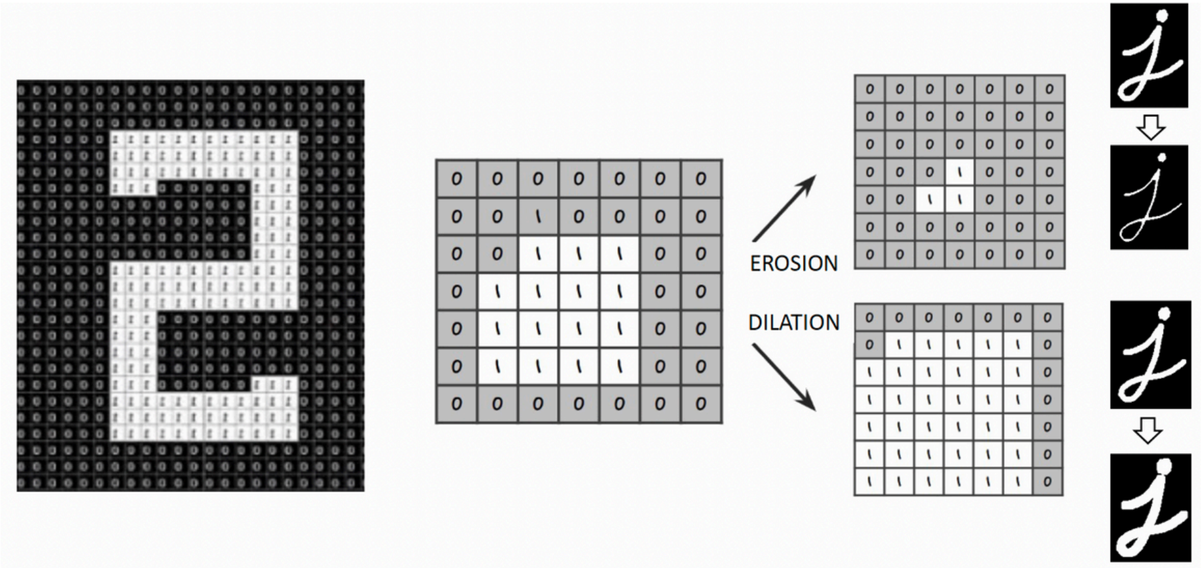


Masked Image



Morphological Operations

Morphological Operations in OpenCV

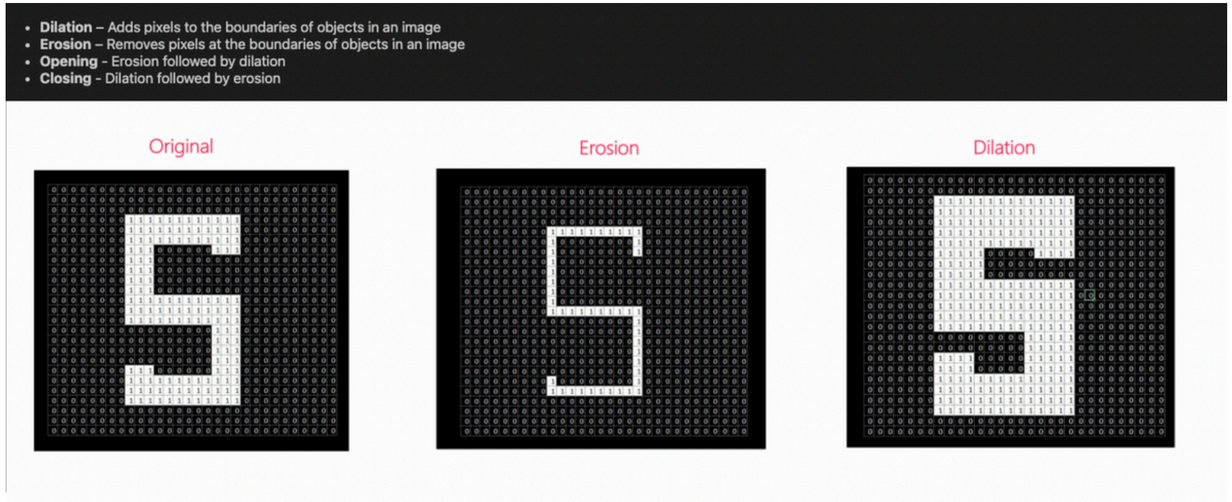


1. Erosion

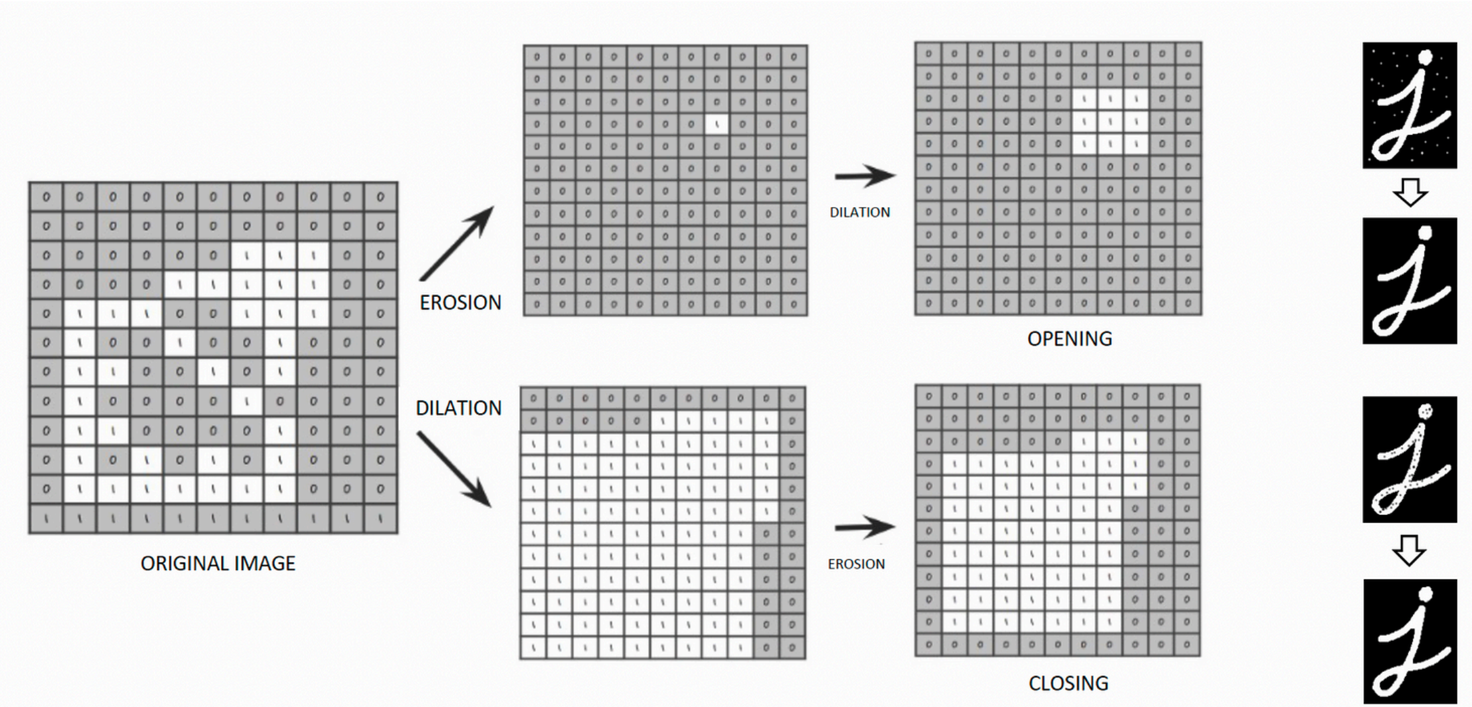
Definition. : Removes pixels at the boundaries of objects in an image.
Applications: Used to remove small noise, separate connected objects, and refine boundaries.

2. Dilation

Definition : Adds pixels to the boundaries of objects in an image.
Applications: Useful for increasing object size, reducing noise, and filling small gaps.



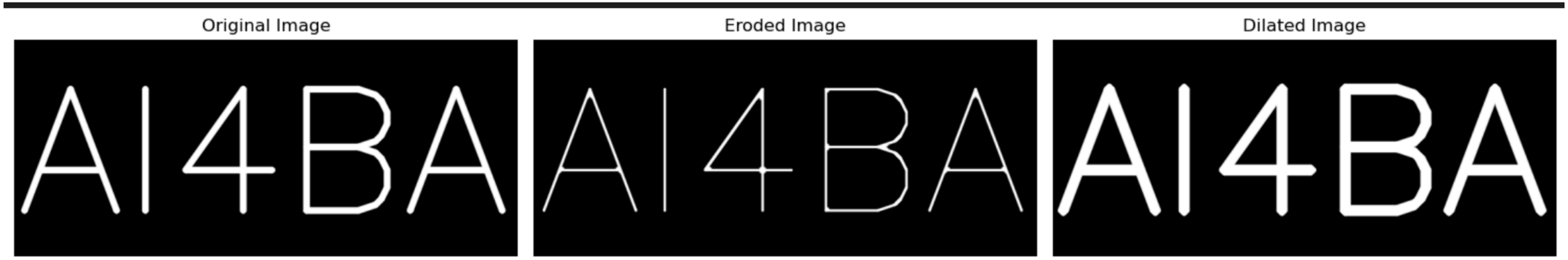
Morphological Operations in OpenCV



3. Opening
   - Process: Erosion followed by dilation.
   - Effect: Removes small objects (noise) from the foreground while keeping the shape and size of larger objects intact.
   - Applications: Useful for cleaning up noise in binary images.

4. Closing
   - Process: Dilation followed by erosion.
   - Effect: Fills small holes and gaps within the objects while maintaining the outer boundary.
   - Applications: Often used to close small holes or gaps in the foreground objects.

```python
# Define kernels
kernel_3x3 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3 , 3))
# 1. Erosion
eroded = cv2.erode(original, kernel_3x3, iterations=3)
dilated = cv2.dilate(original, kernel_3x3, iterations=3)
```

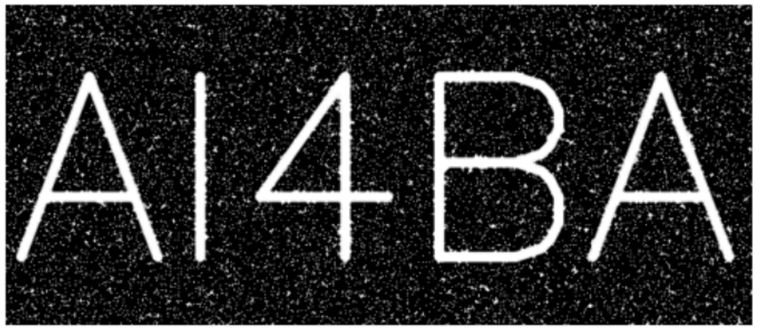


```python
# 3. Opening (Erosion followed by Dilation)
opened = cv2.morphologyEx(result_image, cv2.MORPH_OPEN, kernel_3x3)

# 4. Closing (Dilation followed by Erosion)
closed = cv2.morphologyEx(result_image, cv2.MORPH_CLOSE, kernel_3x3)
```

Noisy Image

Opened Image

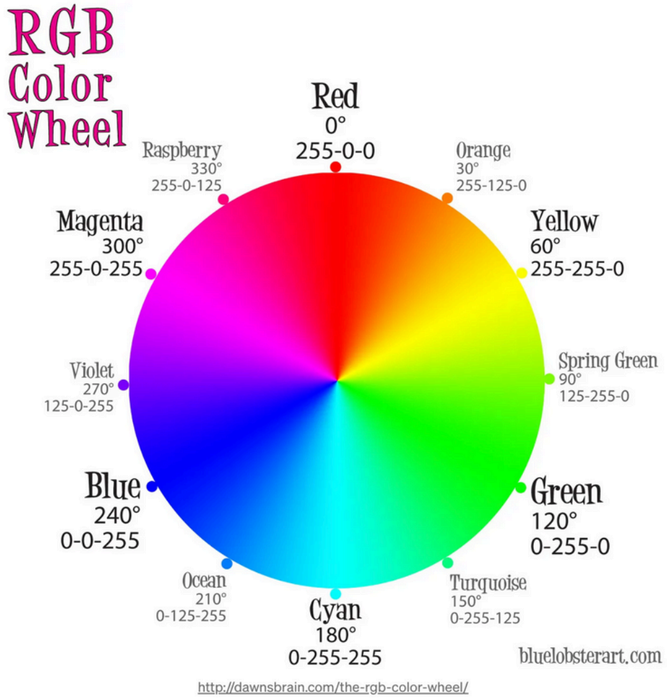
Closed Image

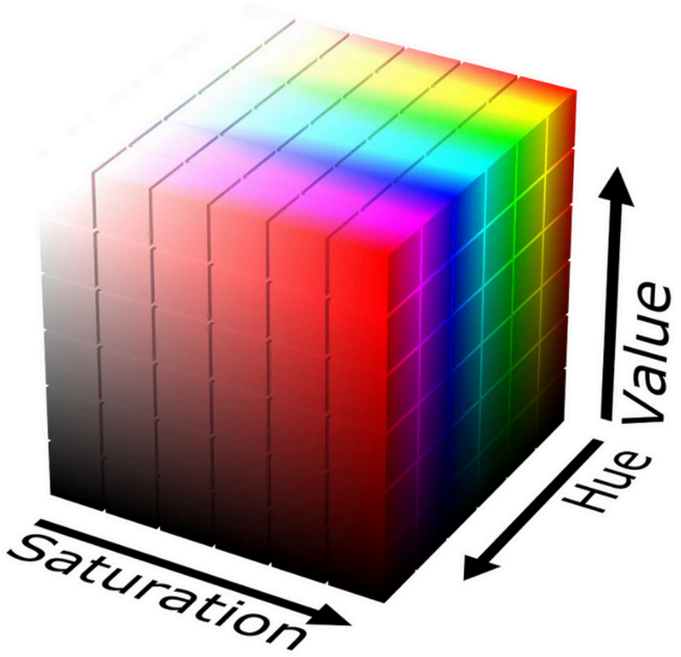


Color Spaces and Applications

RGB — HSV

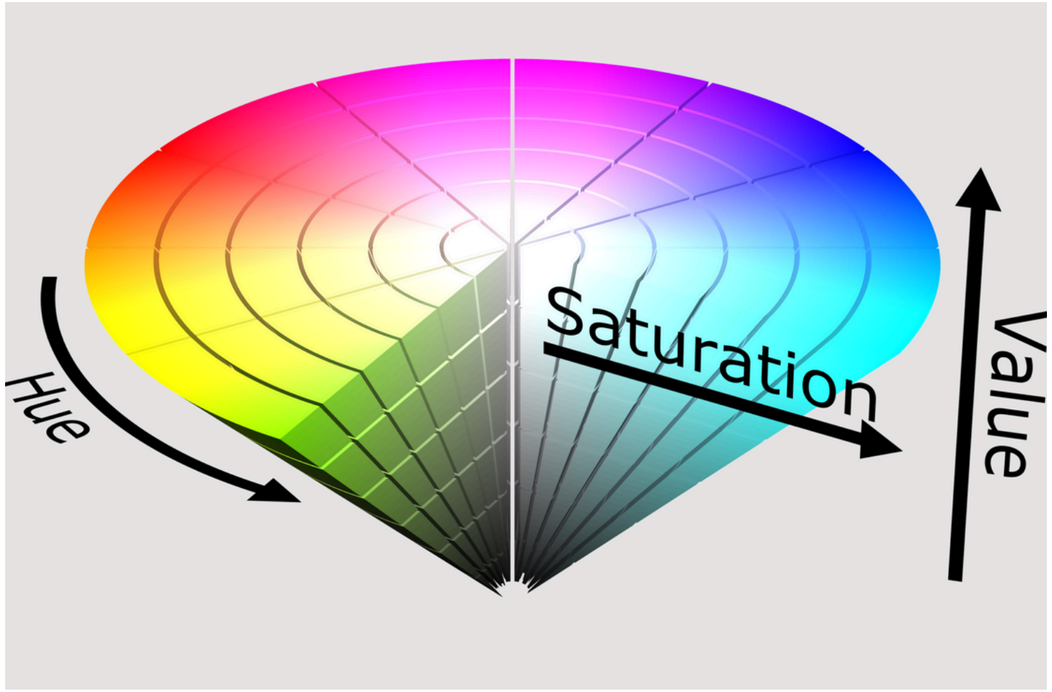
RGB เป็นสีที่เกิดจากการผสมกันของสี(แสง) 3 channels (red, green, blue) โดยเราจะเรียกค่าสีที่เปลี่ยนไปตามวงล้อด้านล่างว่า “Hue”





การทำความเข้าใจ Hue ช่วยให้เราสามารถแยกแยะและเลือกใช้สีได้อย่างมีประสิทธิภาพในการออกแบบและงานศิลปะ
โดย Hue จะระบุถึงตำแหน่งของสีบนวงล้อสี ซึ่งเป็นเครื่องมือที่ใช้ในการจัดระเบียบและแสดงความสัมพันธ์ระหว่างสีต่าง ๆ

RGB — HSV



ใน OpenCV ปริภูมิสี HSV (Hue, Saturation, Value) มีช่วงค่า

Hue (H): 0 ถึง 179, 0 ถึง 360
Saturation (S): 0 ถึง 255
Value (V): 0 ถึง 255

ช่วงค่าของ Hue ถูกกำหนดให้มีค่าตั้งแต่ 0 ถึง 179 แทนที่จะเป็น 0 ถึง 360
เพื่อประหยัดหน่วยความจำและเพิ่มประสิทธิภาพในการประมวลผล
ดังนั้น ค่าของ Hue ใน OpenCV จะมีความละเอียดครึ่งหนึ่งของค่าปกติ

- HSV (Hue, Saturation, Value) เป็นรูปแบบการแทนสีที่ใช้ในคอมพิวเตอร์กราฟิกและการประมวลผลภาพ เพื่อให้การเลือกและปรับแต่งสีเป็นไปอย่างสะดวกและสอดคล้องกับการรับรู้สีของมนุษย์ ปริภูมินี้ประกอบด้วยสามองค์ประกอบหลัก:
- Hue (เฉดสี): แทนมุมรอบแกนแนวตั้งของทรงกระบอกสี โดยเริ่มจากสีแดงที่ 0° ผ่านสีเขียวที่ 120° และสีน้ำเงินที่ 240° จนกลับมาที่สีแดงที่ 360°
- Saturation (ความอิ่มตัว): แทนระยะห่างจากแกนกลางของทรงกระบอกสี โดยค่าต่ำหมายถึงสีที่ซีดหรือเป็นสีเทา และค่าสูงหมายถึงสีที่สดใส
- Value (ความสว่าง): แทนตำแหน่งตามแกนแนวตั้งของทรงกระบอกสี โดยค่าต่ำหมายถึงสีที่มืดหรือดำ และค่าสูงหมายถึงสีที่สว่างหรือขาว

cv2.cvtColor is a function in OpenCV used to convert images from one color space to another.
It supports various color conversions such as RGB to grayscale, RGB to HSV, and many more.

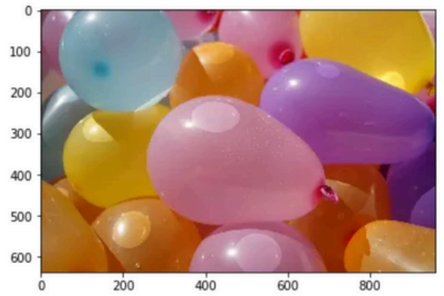
python คัดลอกโค้ด

```
cv2.cvtColor(src, code, dstCn=0)
```

Parameters:

- `src` : The source image to be converted. It should be an image array.
- `code` : The color conversion code. Commonly used codes include:
  - `cv2.COLOR_BGR2GRAY` : Converts BGR to Grayscale.
  - `cv2.COLOR_BGR2RGB` : Converts BGR to RGB.
  - `cv2.COLOR_BGR2HSV` : Converts BGR to HSV.
  - `cv2.COLOR_RGB2BGR` : Converts RGB to BGR.
  - `cv2.COLOR_RGB2GRAY` : Converts RGB to Grayscale.
  - `cv2.COLOR_BGR2Lab` : Converts BGR to Lab color space.
- `dstCn` (optional): The number of channels in the destination image. Default is 0, meaning the number of channels will be derived from the conversion code.

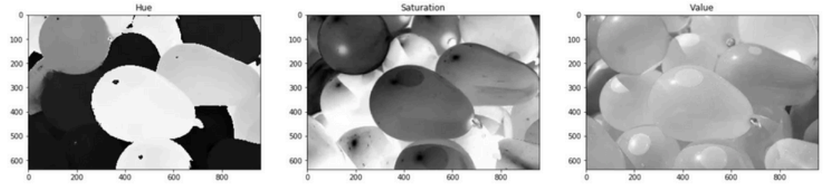
It is very easy to segment a given color if we select the range of Hue that we are interseted, for example, the pink:



Ballons of diferent colors (hues)

First we need to convert to HSV color space, this can be done with openCV:

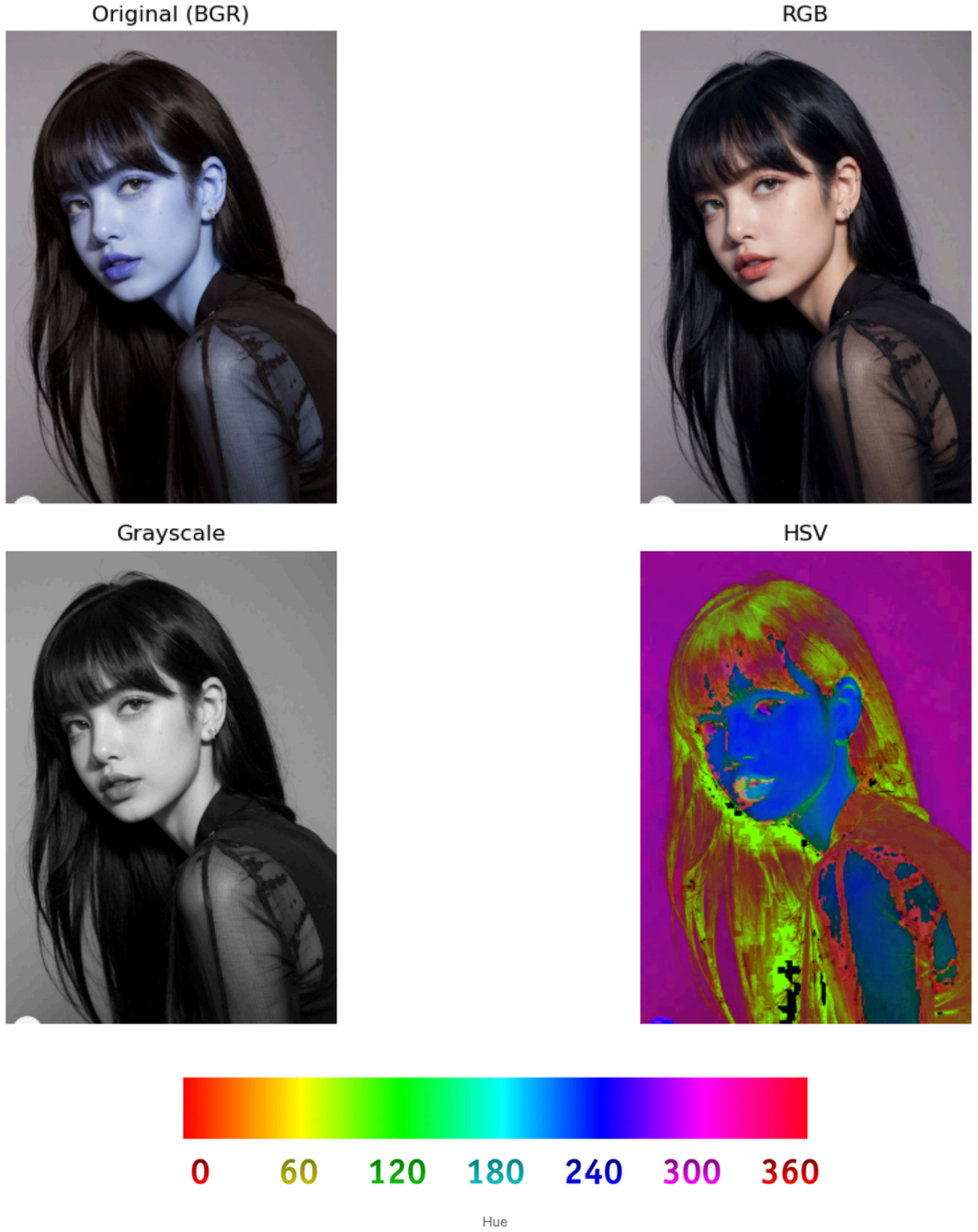
```
cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
```



HSV channels

Hue

```python
import cv2
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('./images/lisa.jpg')

# Convert BGR to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Convert BGR to Grayscale
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Convert BGR to HSV
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Display the images
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.title("Original (BGR)")
plt.imshow(image)  # Convert BGR to RGB for correct display in matplotlib
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title("RGB")
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title("Grayscale")
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title("HSV")
plt.imshow(image_hsv)
plt.axis('off')

plt.tight_layout()
plt.show()
```



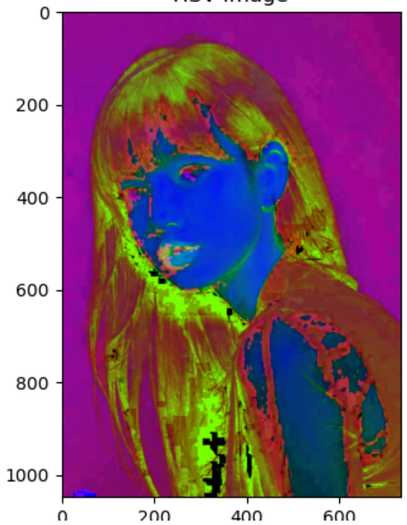
```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image_path = './images/lisa.jpg'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#Convert to HSV for better color segmentation
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

✓ 0.3s

```python
# Define the skin color range in HSV
lower_skin = np.array([0, 30, 60], dtype=np.uint8)
upper_skin = np.array([20, 150, 255], dtype=np.uint8)

# Create a binary mask where skin color matches
mask = cv2.inRange(hsv, lower_skin, upper_skin)

#Bitwise AND to extract the face region
result = cv2.bitwise_and(image_rgb, image_rgb, mask=mask
```

Original Image

HSV Image

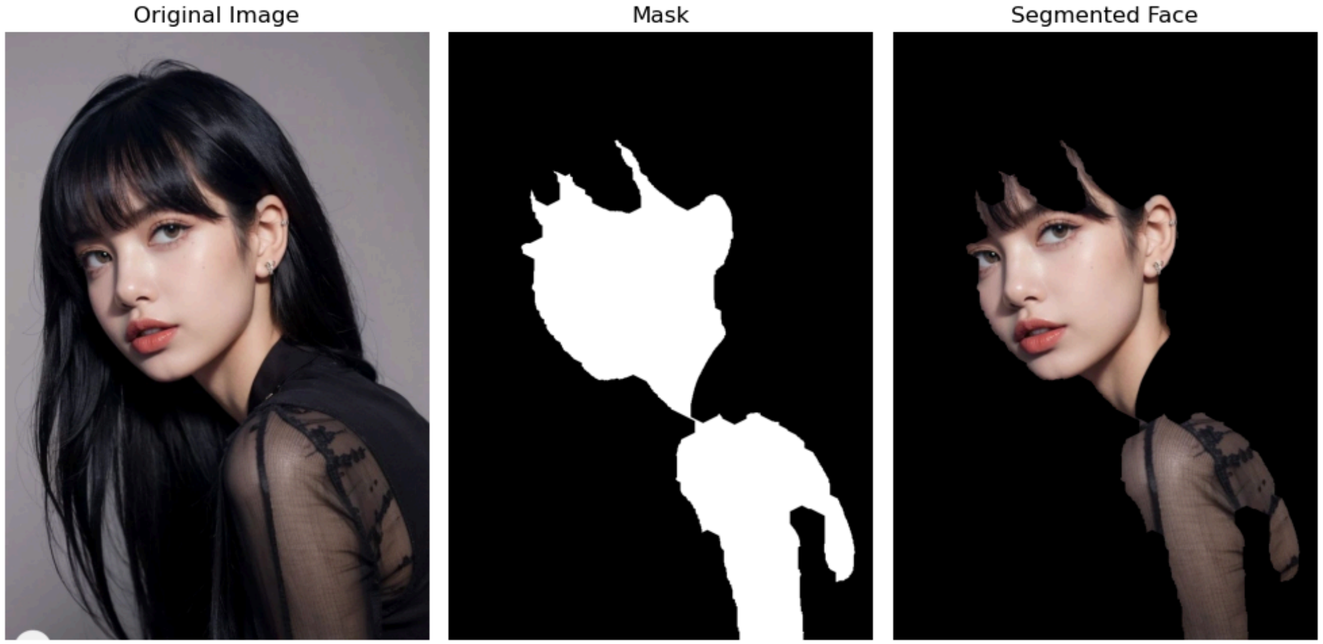


Original Image

Mask

Segmented Face

```python
# Apply morphological operations to clean the mask
kernel        = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5 , 5))
mask_cleaned = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterations=10)

# Bitwise AND to extract the face region
result       = cv2.bitwise_and(image_rgb, image_rgb, mask=mask_cleaned)
```



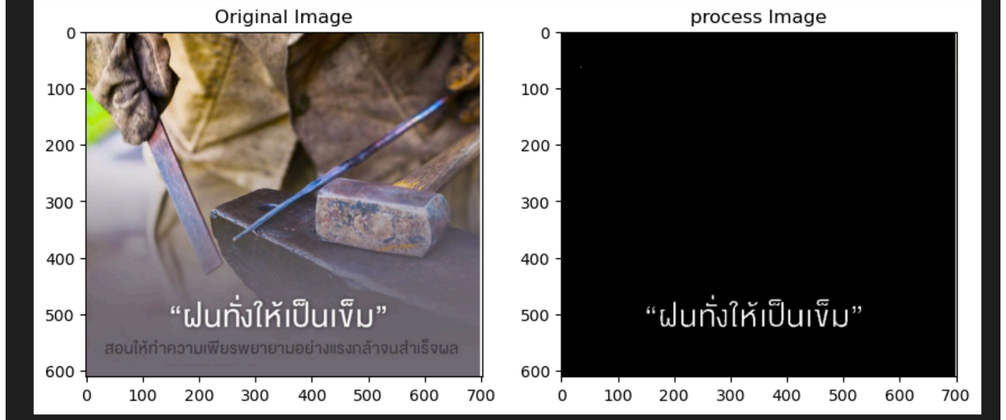
before morphological

after morphological

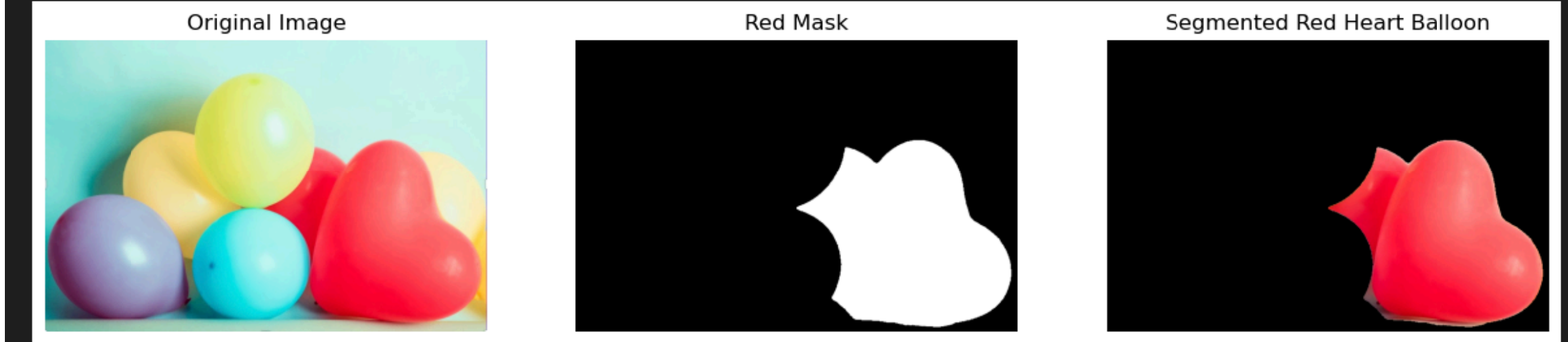
Homework -> ให้ดึงข้อความ ฝนทั่งให้เป็นเข็ม ออกมาตามรูป

✓ 0.2s

```
<matplotlib.image.AxesImage at 0x16864b6d0>
```



Homework ---> ให้ Segment ballon หัวใจ



Homework ---> ให้ Segment cracked egg

```
Press ⌘ I to ask GitHub Copilot to do something. Start typing to dismiss.
```

```
<matplotlib.image.AxesImage at 0x136c64220>
```