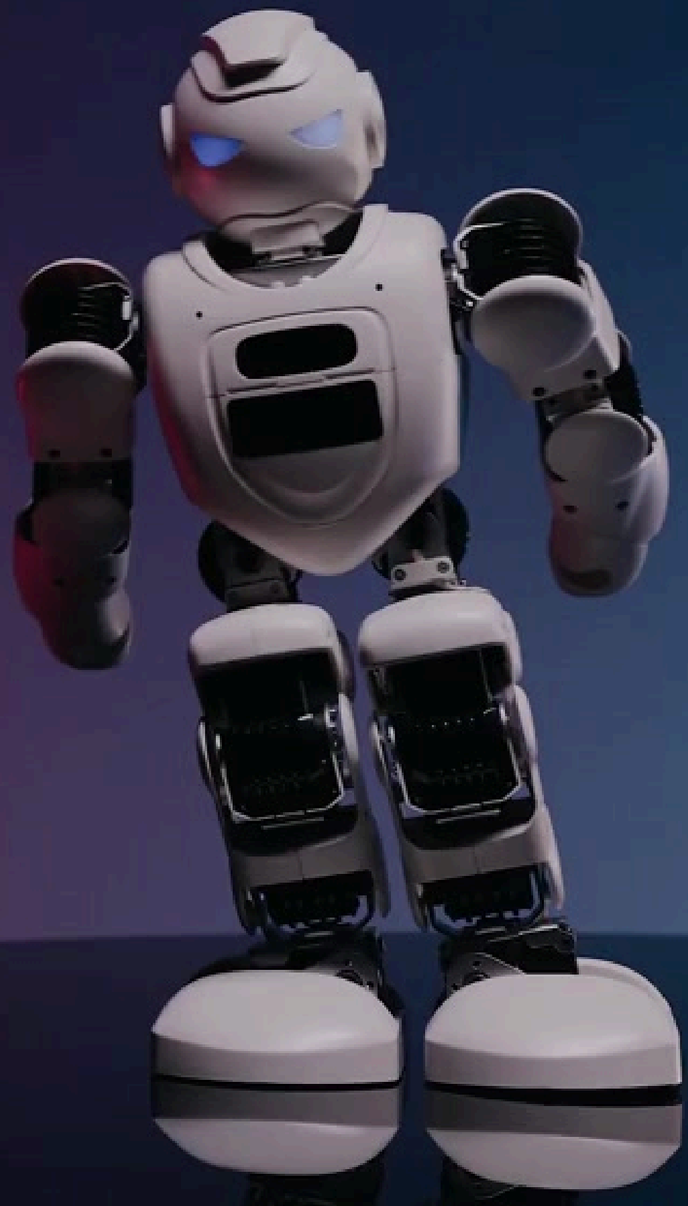
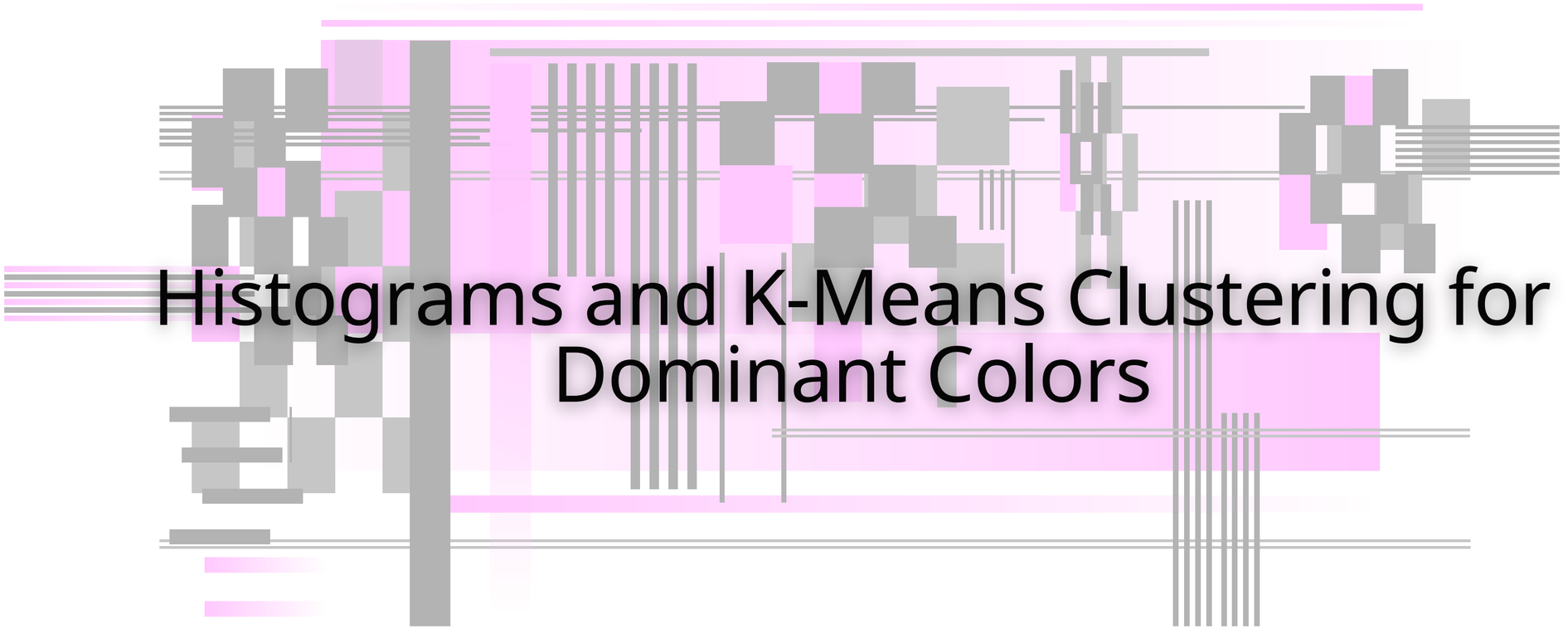
WEEK 4

DEEP LEARNING FOR COMPUTER VISION

UNLIMITED



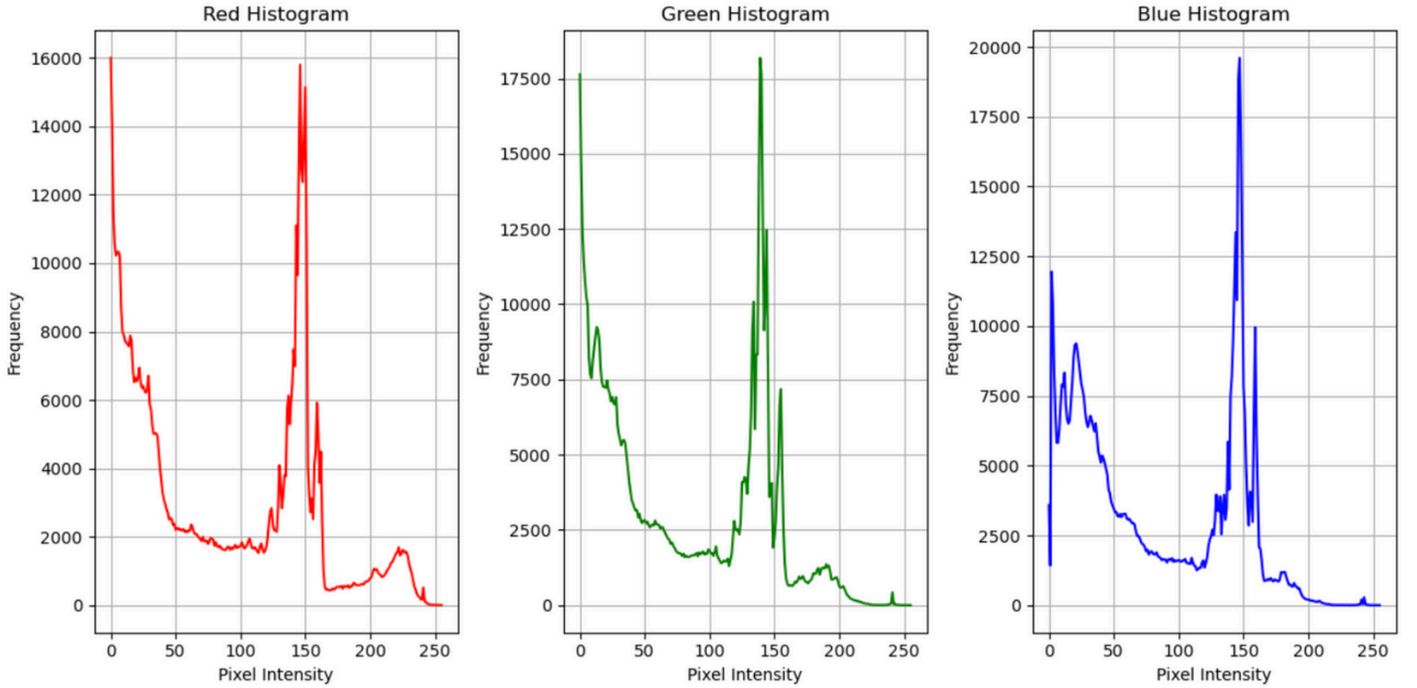
Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**

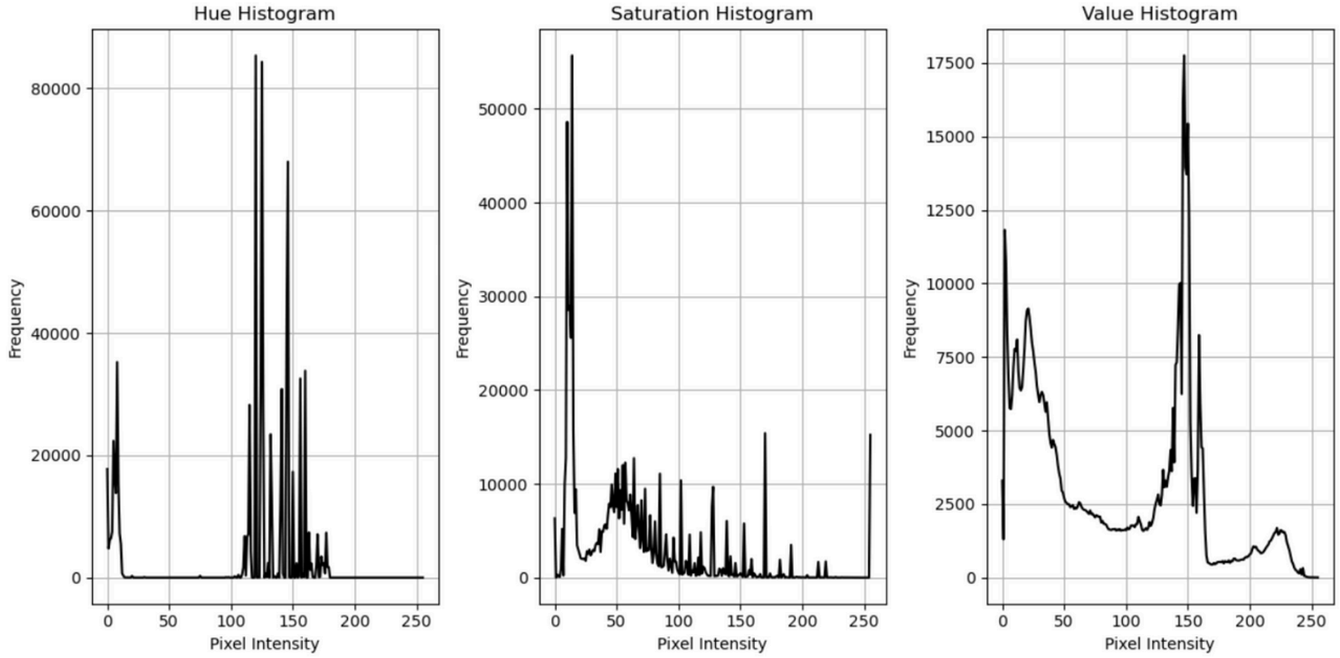

Histograms and K-Means Clustering for
Dominant Colors

Original Image (RGB)

RGB Histrogram

HSV Histrogram





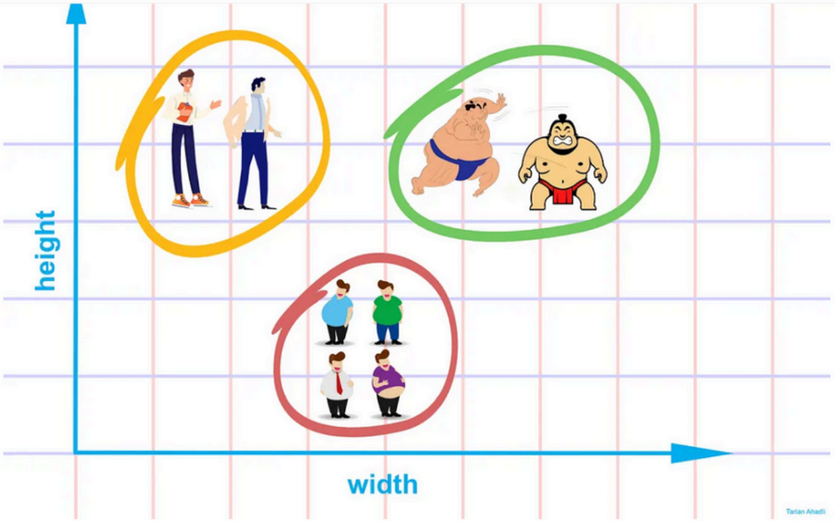
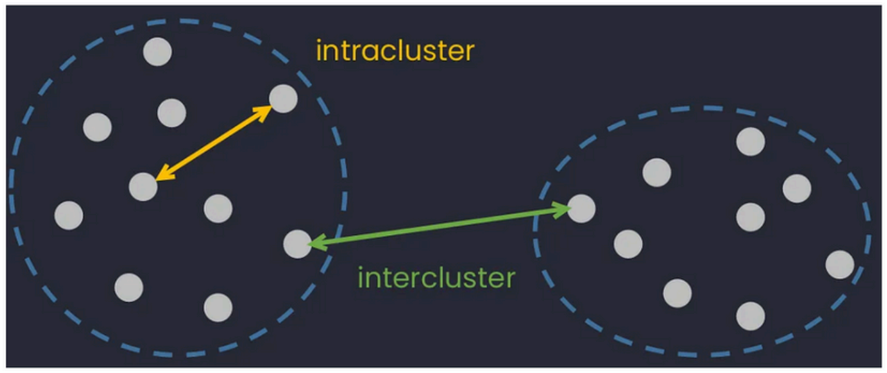


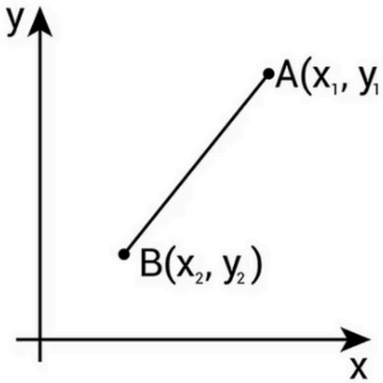
Image Source: Google Images

Main points:

- **Intercluster distance should be high:** The distance between observations in two clusters should be High.
- **Intracluster Distance should be Very Less:** The distance of observation within the cluster should be very less.



Distance Formula



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
image = cv2.imread(image_path)
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
image_hsv.shape
```

✓ 0.0s

```
(1048, 732, 3)
```

```
# Reshape the Image to a 2D Array of Pixels in HSV
pixels_hsv = image_hsv.reshape((-1, 3))
pixels_hsv.shape
```

✓ 0.0s

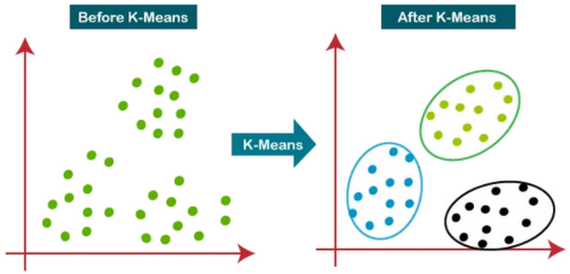
```
(767136, 3)
```

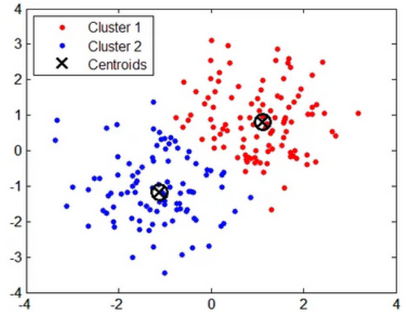
Find Centers as domiant color

```
# Reshape the Image to a 2D Array of Pixels in HSV
pixels_hsv = image_hsv.reshape((-1, 3))
pixels_hsv.shape
```

✓ 0.0s

```
(767136, 3)
```






```
# Extract the Cluster Centers (Dominant Colors) in HSV
dominant_colors_hsv = kmeans.cluster_centers_.astype(int)

# Convert Dominant Colors Back to RGB for Visualization
dominant_colors_rgb = cv2.cvtColor(np.uint8([dominant_colors_hsv]), cv2.COLOR_HSV2RGB)[0]
```

```
# Define the Number of Clusters (Colors)
num_colors = 5

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=num_colors, random_state=42)
kmeans.fit(pixels_hsv)
```

X (input)

```
pixels_hsv.shape
```

✓ 0.0s

```
(767136, 3)
```

Y (Forecasting)

```
# Assign Labels to Each Pixel
labels = kmeans.labels_
labels.shape
```

✓ 0.0s

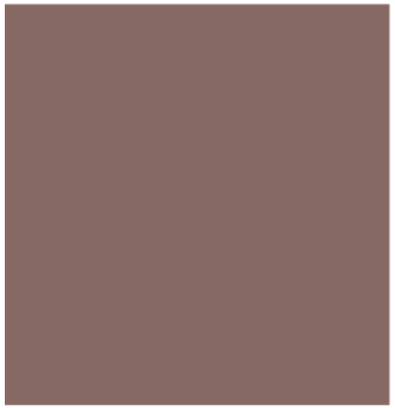
```
(767136,)
```

```
# Define the Number of Clusters (Colors)
num_colors = 5

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=num_colors, random_state=42)
kmeans.fit(pixels_hsv)

# Extract the Cluster Centers (Dominant Colors) in HSV
dominant_colors_hsv = kmeans.cluster_centers_.astype(int)

# Convert Dominant Colors Back to RGB for Visualization
dominant_colors_rgb = cv2.cvtColor(np.uint8([dominant_colors_hsv]), cv2.COLOR_HSV2RGB)[0]

# Assign Labels to Each Pixel
labels = kmeans.labels_

# Reshape the Labels Back to the Shape of the Original Image
labels_reshaped = labels.reshape(image_hsv.shape[:2])

# Display the Dominant Colors
plt.figure(figsize=(8, 4))
for i, color in enumerate(dominant_colors_rgb):
    plt.subplot(1, num_colors, i + 1)
    plt.imshow([[color / 255]])
    plt.title(f"Color {i + 1}")
    plt.axis("off")
plt.suptitle("Dominant Colors (RGB)")
plt.show()
```

Original Image (RGB)

Dominant Colors (RGB)

Color 1

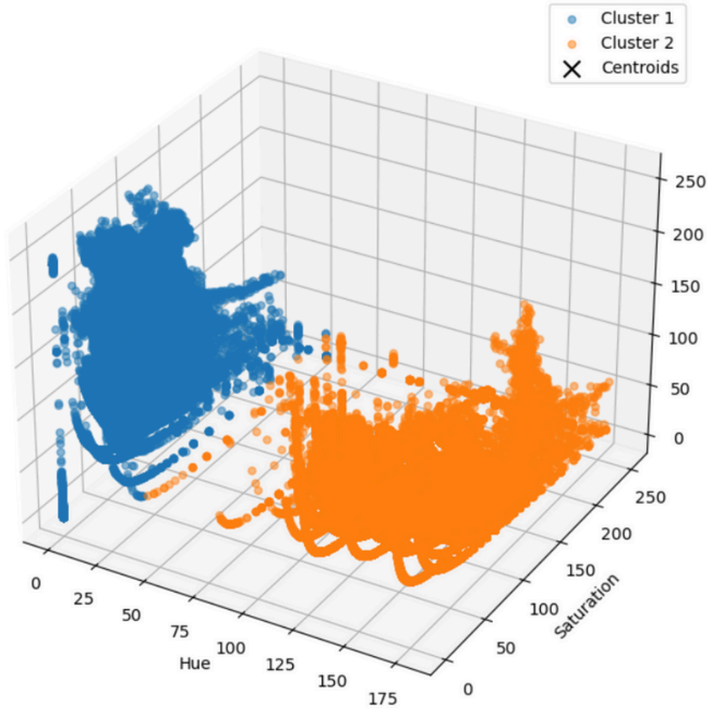


Color 2

Segmented Region for Color 1

Segmented Region for Color 2

3D Visualization of Clusters in HSV Space



Original Image (RGB)

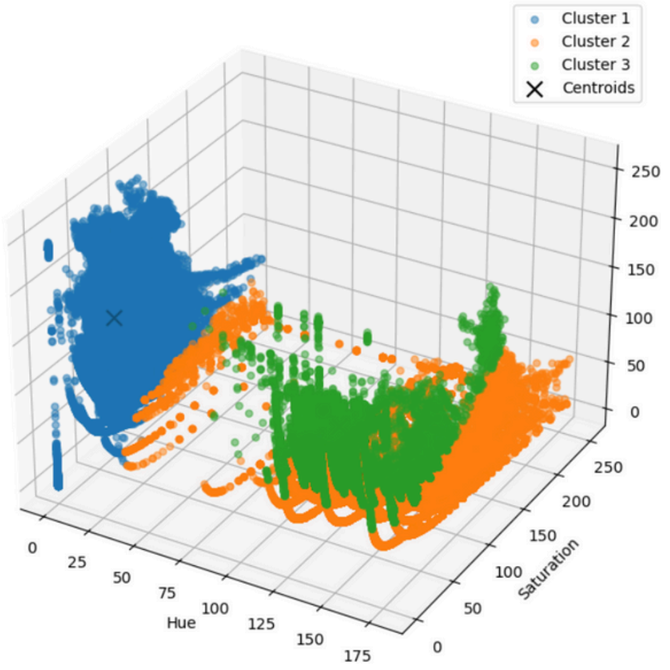
Dominant Colors (RGB)

Color 1

Color 2

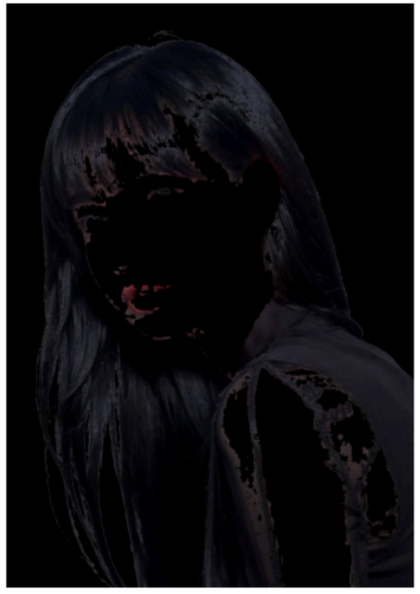
Color 3

3D Visualization of Clusters in HSV Space



Segmented Region for Color 1

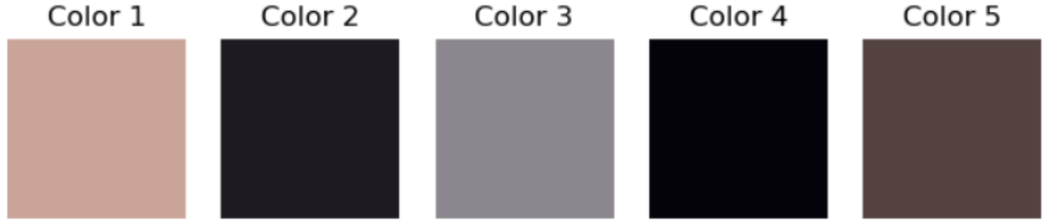
Segmented Region for Color 2

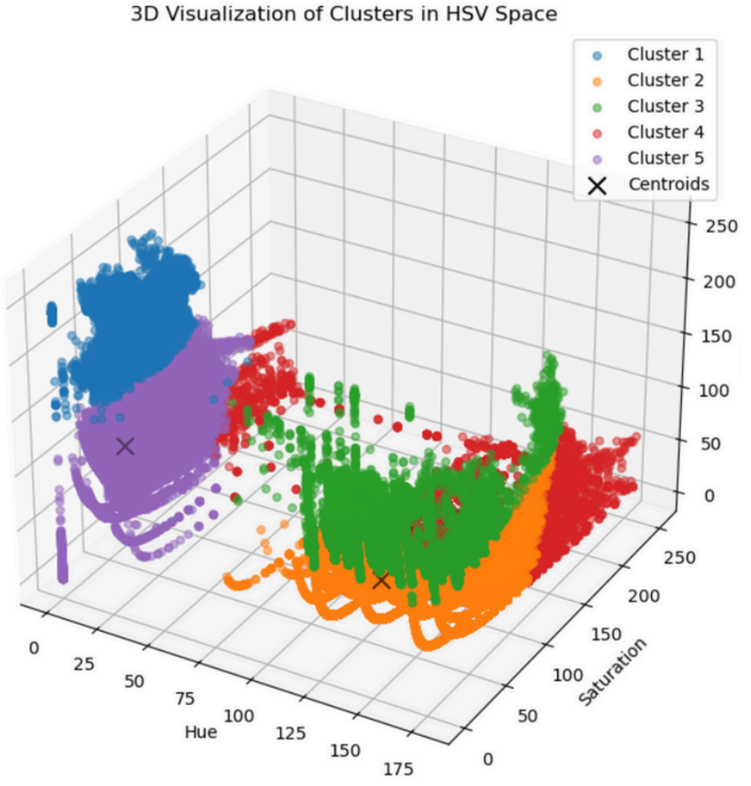


Segmented Region for Color 3

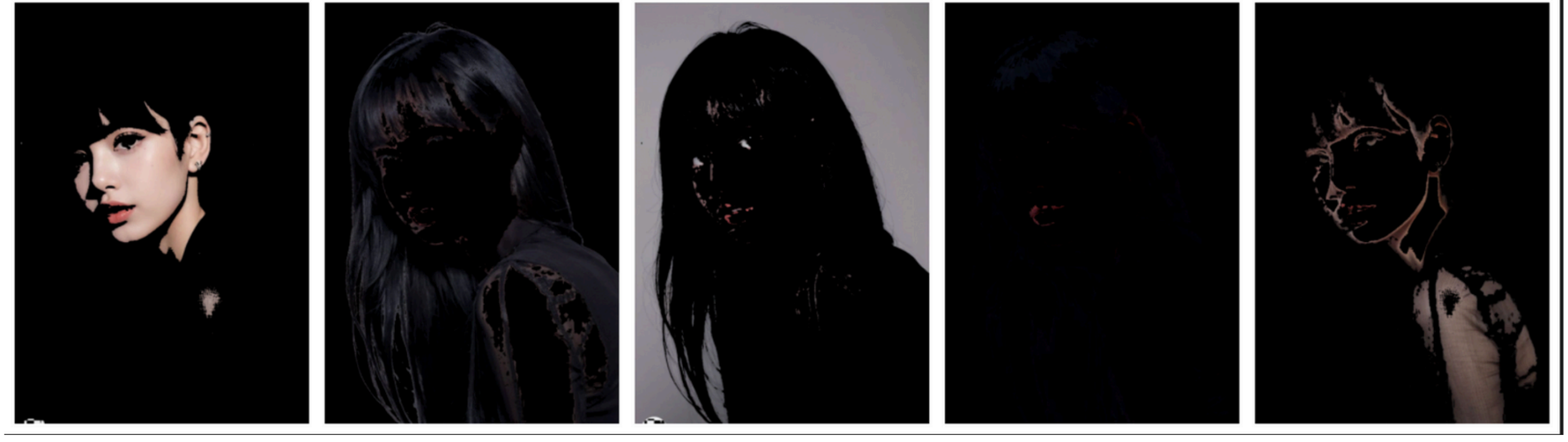
Original Image (RGB)

Color 1 | Color 2 | Color 3 | Color 4 | Color 5





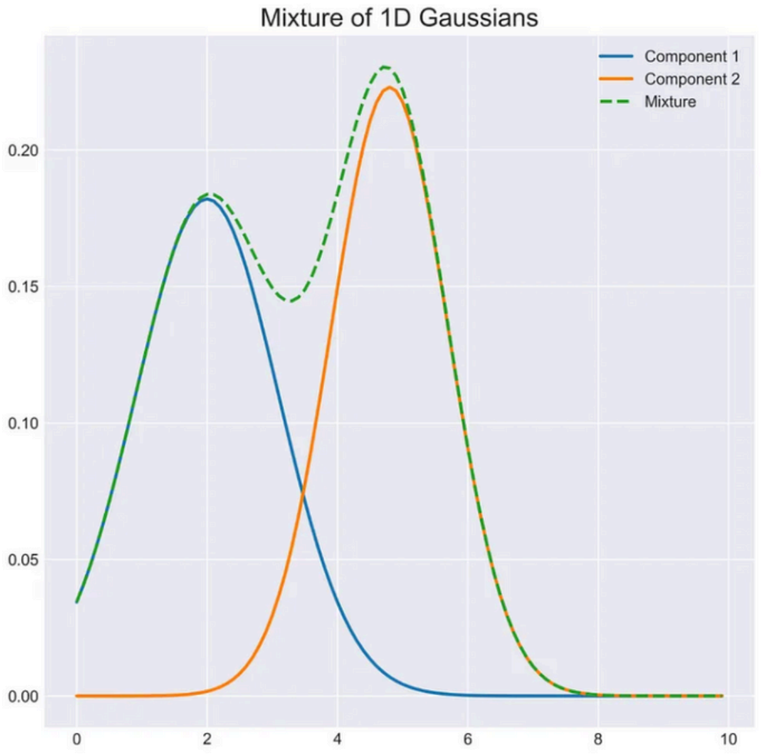
Segmented Region for Color 1 | Segmented Region for Color 2 | Segmented Region for Color 3 | Segmented Region for Color 4 | Segmented Region for Color 5

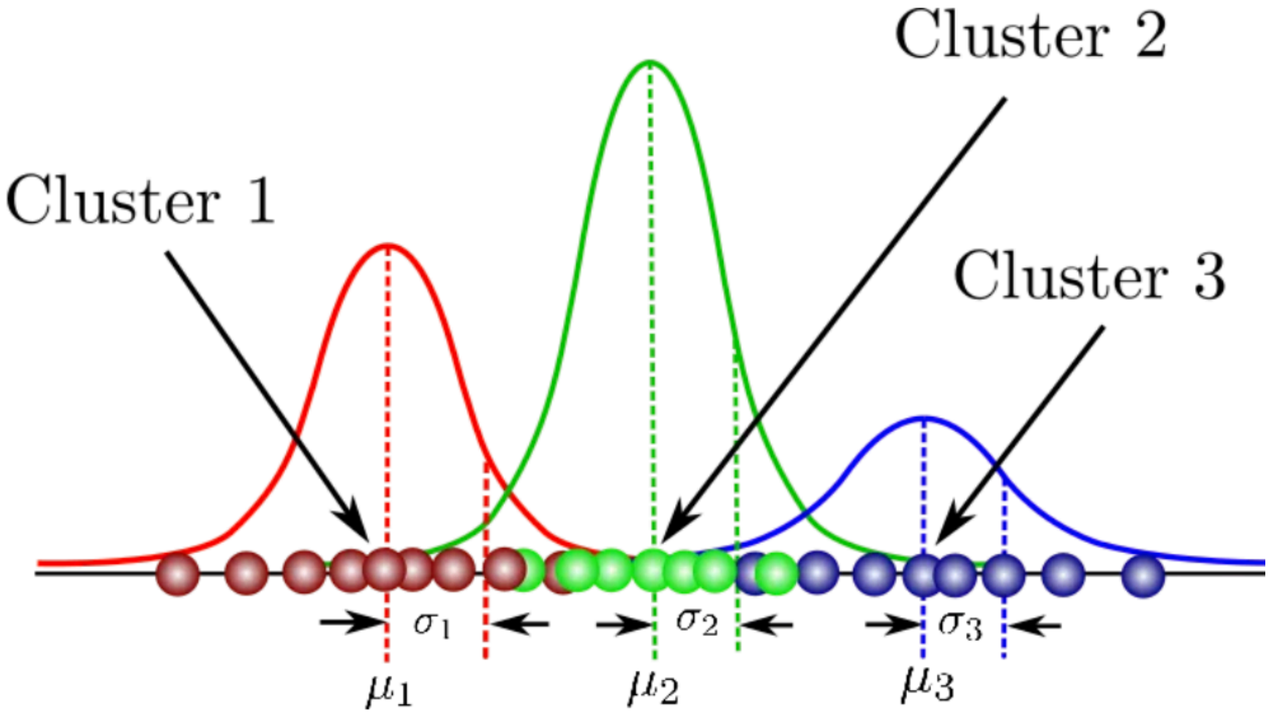


Histograms and Gaussian Mixture Models Clustering for Dominant Colors

What's a Gaussian Mixture Model Clustering?

Throughout the world, most datasets are available to be described by a Gaussian Distribution (Univariate or Multivariate). Based on this assumption, it is safe to say that clusters in datasets are formed by this distribution.





$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k)$$

How does it work?

As per the name states, GMM would incorporate all of the Gaussian Distributions into one model. Assumed that there a number of clusters, its mean and its sum will all be calculated, which then will be used in searching the probability density for each cluster.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k)$$

Probability Density Formula

- p(x): Probability density function of GMM at data point x.
- π_k: Mixing coefficient of the k-th Gaussian component.
- $\mathcal{N}$(x; μ_k, Σ_k): Multivariate Gaussian distribution with mean μ_k and covariance Σ_k.

After finding the probability, the next step would be finding its posterior probability. In simpler term, posterior probability could be considered as the confidence level in our hypothesis after gathering new evidence.

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(x; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x; \mu_j, \Sigma_j)}$$

Posterior Probability Formula

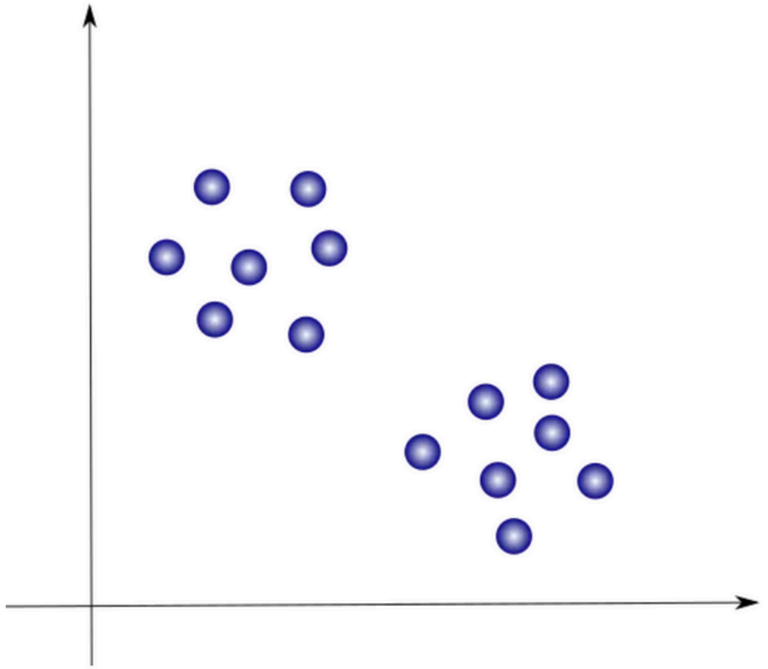
The next step after obtaining its' posterior probability, would be to update the model's parameters which are mixing coeffficients and mean vectors.

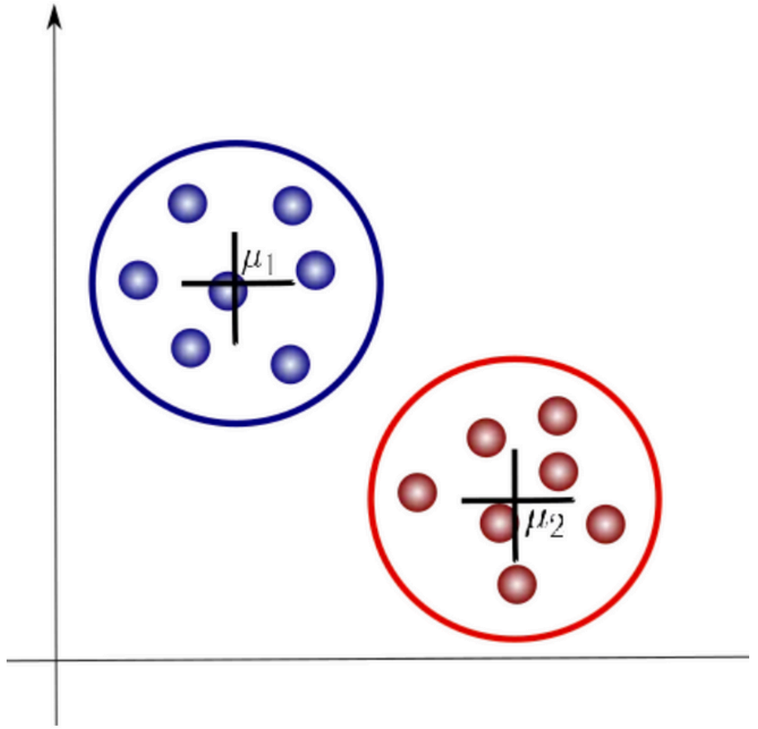
$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_k)}{N}$$

Mixing Coefficients

$$\mu_k = \frac{1}{\sum_{n=1}^{N} \gamma(z_k)} \sum_{n=1}^{N} \gamma(z_k) x_n$$

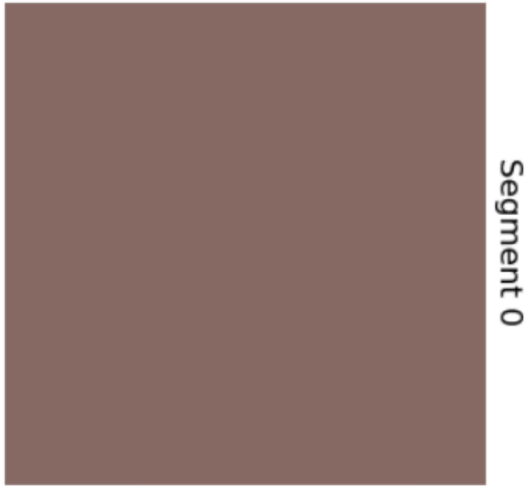
Mean Vectors

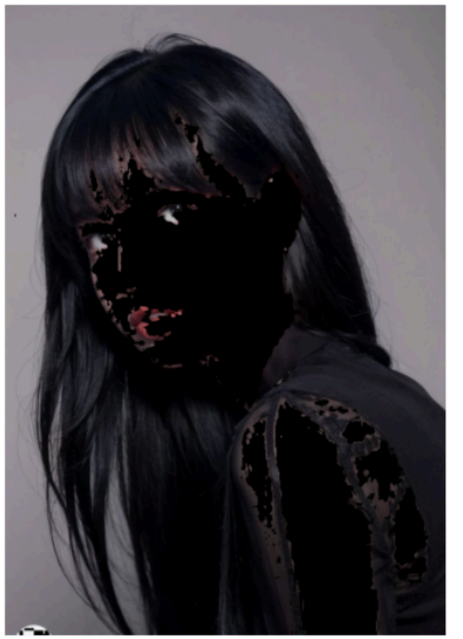




$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k)$$

```python
from sklearn.mixture import GaussianMixture
import cv2
import matplotlib.pyplot as plt

image_path = './images/lisa.jpg'  # Update this path if needed
image_rgb = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)
image_hsv = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2HSV)
# Reshape the image
pixels_rgb = image_rgb.reshape(-1, 3)
pixels_hsv = image_hsv.reshape(-1, 3)

# Apply Gaussian Mixture Model
gmm = GaussianMixture(n_components=2, covariance_type='tied', random_state=42)
gmm_labels = gmm.fit_predict(pixels_hsv )

# Reshape to the image's original dimensions
gmm_segmentation = gmm_labels.reshape(image_rgb.shape[:2])

# Display Gaussian Mixture Model Segmentation
plt.figure(figsize=(6, 6))
plt.imshow(gmm_segmentation, cmap='nipy_spectral')
plt.title("Gaussian Mixture Model Segmentation")
plt.axis("off")
plt.show()
```



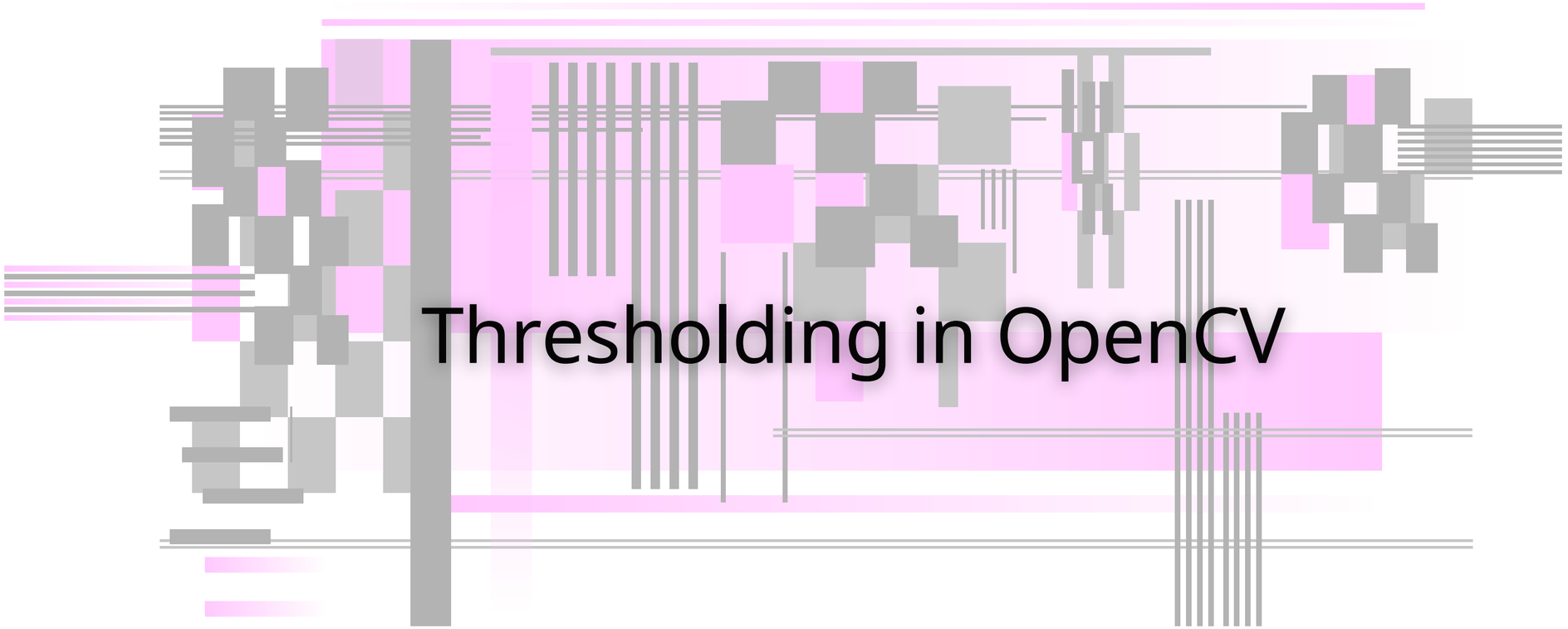


ตารางเปรียบเทียบระหว่าง K-Means และ Gaussian Mixture Model (GMM) สำหรับการใช้ใน Clustering ของ Dominant Colors

หัวข้อเปรียบเทียบ	K-Means	Gaussian Mixture Model (GMM)
วิธีการทำงาน	ใช้อัลกอริทึมแบ่งกลุ่มโดยการหาค่าเฉลี่ยระยะทางจากจุดข้อมูลไปยังจุดศูนย์กลาง (Centroids)	ใช้แบบจำลองทางสถิติโดยอาศัย Gaussian Distribution สำหรับการแบ่งกลุ่ม
ประเภทของกลุ่มที่สร้างได้	สร้างกลุ่มที่มีลักษณะเป็นวงกลมหรือรูปทรงที่ระยะห่างจากศูนย์กลางใกล้เคียงกัน	สามารถสร้างกลุ่มที่มีลักษณะไม่จำเป็นต้องเป็นวงกลม เช่น วงรี หรือการกระจายตัวที่ซับซ้อนกว่า
การกำหนดจำนวนกลุ่ม (K)	ผู้ใช้งานต้องกำหนดจำนวนกลุ่มล่วงหน้า (Hyperparameter)	ผู้ใช้งานต้องกำหนดจำนวนส่วนผสมของ Gaussian ล่วงหน้า
ความยืดหยุ่นของรูปร่างกลุ่ม	จำกัดที่รูปทรงวงกลมเท่านั้น เนื่องจากใช้ Euclidean Distance เป็นเกณฑ์	ยืดหยุ่นกว่า สามารถจัดการกับกลุ่มที่มีการกระจายแบบ Gaussian ที่แตกต่างกันได้
ความเร็วในการทำงาน	เร็วกว่า เพราะมีการคำนวณแบบ Iterative และเรียบง่าย	ช้ากว่า เพราะมีการใช้ Expectation-Maximization (EM) Algorithm
ความสามารถในการแบ่งกลุ่มซับซ้อน	จำกัดเมื่อกลุ่มมีความซับซ้อน เช่น ข้อมูลที่มีการซ้อนทับของกลุ่ม	เหมาะสำหรับข้อมูลที่ซับซ้อนและมีการซ้อนทับระหว่างกลุ่ม
ผลลัพธ์	ให้ Centroids ของแต่ละกลุ่ม	ให้ Probabilistic Membership หรือความน่าจะเป็นที่จุดข้อมูลจะอยู่ในแต่ละกลุ่ม
การทำงานร่วมกับ Dominant Colors	ใช้ Centroids ในการกำหนดค่าสีหลัก	ใช้ Gaussian Components เพื่อประมาณค่าสีหลัก
จุดเด่น	1. เร็วกว่า 2. ใช้งานง่าย 3. ไม่ซับซ้อนในเชิงคณิตศาสตร์	1. ยืดหยุ่นกว่า 2. เหมาะกับกลุ่มข้อมูลที่ซับซ้อน 3. ใช้ Probabilistic Membership
จุดด้อย	1. ไม่เหมาะกับข้อมูลซับซ้อนหรือการกระจายแบบไม่สมมาตร 2. ไวต่อค่าเริ่มต้นของ Centroids	1. ช้ากว่า 2. ซับซ้อนกว่าในเชิงคณิตศาสตร์และการคำนวณ

การเลือกใช้งาน:

- **K-Means:** เหมาะสำหรับงานที่ต้องการความเร็ว เช่น การลดสีของภาพ (Image Quantization) ในกรณีที่กลุ่มข้อมูลไม่ซับซ้อนมาก
- **GMM:** เหมาะสำหรับงานที่ต้องการความแม่นยำสูงกว่าในกรณีที่กลุ่มข้อมูลมีการซ้อนทับกันหรือมีความซับซ้อน เช่น การวิเคราะห์ข้อมูลภาพที่มีหลายแหล่งกำเนิดของสี


Thresholding in OpenCV

ฟังก์ชัน `cv2.threshold` ใน OpenCV ใช้สำหรับการ **Thresholding** ซึ่งเป็นกระบวนการแปลงภาพให้อยู่ในรูปแบบที่ง่ายขึ้น เช่น การแปลงเป็นภาพ **Binary** (ขาว-ดำ) หรือการลดระดับความเข้ม (Intensity Levels) ของพิกเซลในภาพ โดยใช้ค่ากำหนด (Threshold) เพื่อแบ่งภาพออกเป็นส่วนต่าง ๆ

การใช้งาน Thresholding

Thresholding ใช้ในงานประมวลผลภาพที่ต้องการ:

1. **แยกวัตถุออกจากพื้นหลัง** (Object Segmentation)
2. **เตรียมข้อมูลสำหรับการตรวจจับขอบเขต** (Contour Detection)
3. **ลดความซับซ้อนของภาพ** เพื่อให้ประมวลผลต่อได้ง่ายขึ้น เช่น สำหรับการรู้จำตัวอักษร (OCR)

โครงสร้างฟังก์ชัน

python — คัดลอกโค้ด

```python
retval, thresholded_image = cv2.threshold(src, thresh, maxval, type)
```

พารามิเตอร์

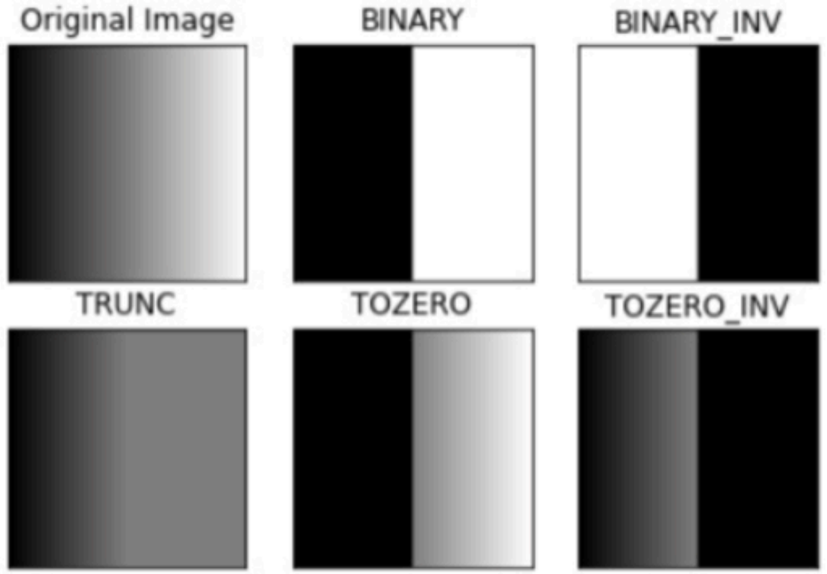
1. `src`:
   - ภาพต้นฉบับที่ต้องการทำ Thresholding (ต้องเป็นภาพ **Grayscale**)
   - รูปแบบข้อมูล: ภาพ 8-bit หรือ 32-bit floating point
2. `thresh`:
   - ค่ากำหนด (Threshold Value) ที่จะใช้แบ่งพิกเซลในภาพ
3. `maxval`:
   - ค่าสูงสุดที่ใช้แทนพิกเซลที่ผ่านเงื่อนไข (ใช้กับ Threshold แบบ Binary)
4. `type`:
   - ประเภทของ Thresholding ซึ่ง OpenCV มีหลายแบบให้เลือก:

1. `retval`:
   - ค่า Threshold ที่ใช้ (สำคัญในกรณีที่ใช้ `THRESH_OTSU` หรือ `THRESH_TRIANGLE`)
2. `thresholded_image`:
   - ภาพที่ผ่านการ Thresholding แล้ว

```python
# ใช้ Thresholding ประเภทต่างๆ
_, binary = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
_, binary_inv = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
_, trunc = cv2.threshold(image, 127, 255, cv2.THRESH_TRUNC)
_, tozero = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO)
_, tozero_inv = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO_INV)
```

การใช้งาน	ประเภท Thresholding ที่เหมาะสม
การแยกข้อความออกจากภาพพื้นหลัง	`THRESH_BINARY` หรือ `THRESH_BINARY_INV`
การวิเคราะห์ภาพที่มีความสว่างเกินไป	`THRESH_TRUNC`
การตรวจจับส่วนที่มีความเข้มสูง (bright spots)	`THRESH_TOZERO`
การลบจุดที่สว่างเกินไป	`THRESH_TOZERO_INV`

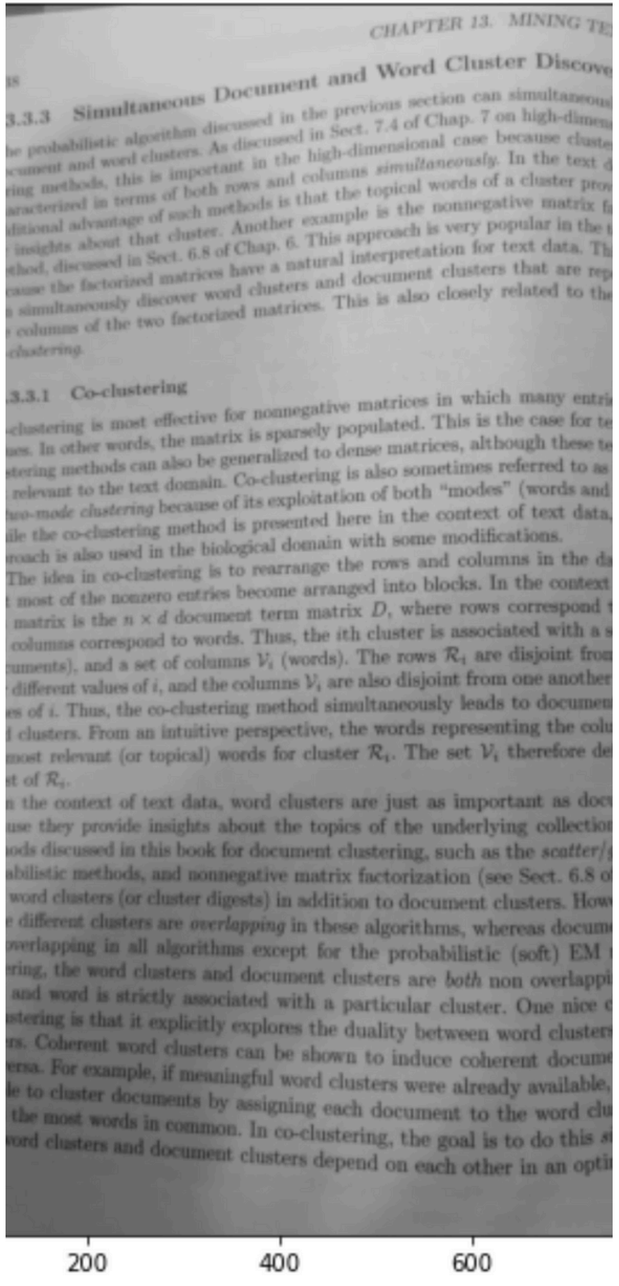
ประเภท Thresholding	คำอธิบาย	เงื่อนไข	ค่าพิกเซลที่ได้	การใช้งาน
cv2.THRESH_BINARY	พิกเซลที่ค่ามากกว่า thresh จะถูกกำหนดเป็น maxval และค่าที่เหลือจะเป็น 0	src(x, y) > thresh	maxval ถ้าจริง, ไม่เช่นนั้นเป็น 0	ใช้สำหรับแยกพื้นหลังออกจากวัตถุ เช่น การแยกข้อความในภาพ
cv2.THRESH_BINARY_INV	กลับค่าของ cv2.THRESH_BINARY พิกเซลที่ค่ามากกว่า thresh จะถูกกำหนดเป็น 0 และค่าที่เหลือเป็น maxval	src(x, y) > thresh	0 ถ้าจริง, ไม่เช่นนั้นเป็น maxval	ใช้เมื่อต้องการวัตถุเป็นพื้นหลังและพื้นหลังเป็นวัตถุ เช่น ข้อความสีขาวบนพื้นดำ
cv2.THRESH_TRUNC	พิกเซลที่มีค่ามากกว่า thresh จะถูกกำหนดค่าเป็น thresh ส่วนค่าที่เหลือจะคงเดิม	src(x, y) > thresh	thresh ถ้าจริง, ไม่เช่นนั้นเป็นค่าดั้งเดิม	ใช้สำหรับลดค่าเกินในภาพ (clipping) เพื่อเน้นขอบเขตค่าหนึ่ง
cv2.THRESH_TOZERO	พิกเซลที่มีค่ามากกว่า thresh จะคงเดิม ส่วนค่าที่เหลือจะถูกกำหนดเป็น 0	src(x, y) > thresh	ค่าดั้งเดิมถ้าจริง, ไม่เช่นนั้นเป็น 0	ใช้สำหรับเน้นส่วนที่สว่างและลบส่วนที่มืดออก เช่น การหาจุดที่มีความเข้มสูง
cv2.THRESH_TOZERO_INV	กลับค่าของ cv2.THRESH_TOZERO พิกเซลที่มีค่ามากกว่า thresh จะถูกกำหนดเป็น 0 ส่วนค่าที่เหลือจะคงเดิม	src(x, y) > thresh	0 ถ้าจริง, ไม่เช่นนั้นเป็นค่าดั้งเดิม	ใช้สำหรับลบส่วนที่สว่างและเก็บส่วนที่มืดไว้ เช่น เงาในภาพ



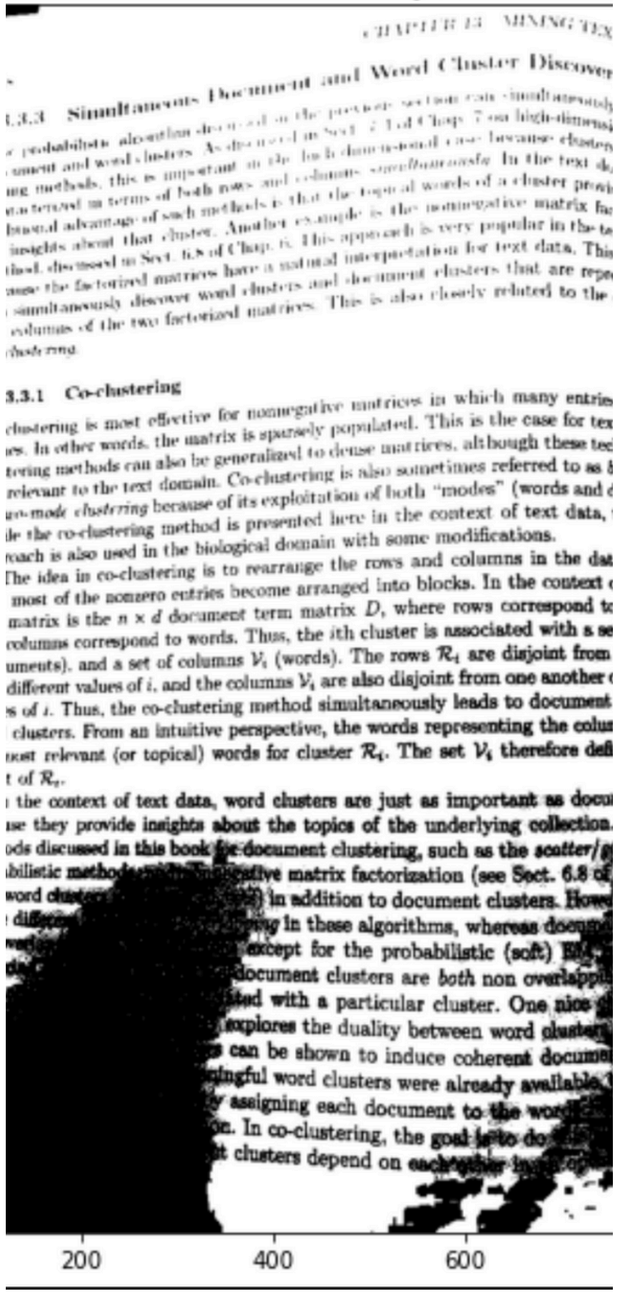

```python
# ใช้ Thresholding ประเภทต่างๆ
_, binary = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
_, binary_inv = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
_, trunc = cv2.threshold(image, 127, 255, cv2.THRESH_TRUNC)
_, tozero = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO)
_, tozero_inv = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO_INV)
```

การใช้งาน	ประเภท Thresholding ที่เหมาะสม
การแยกข้อความออกจากภาพพื้นหลัง	`THRESH_BINARY` หรือ `THRESH_BINARY_INV`
การวิเคราะห์ภาพที่มีความสว่างเกินไป	`THRESH_TRUNC`
การตรวจจับส่วนที่มีความเข้มสูง (bright spots)	`THRESH_TOZERO`
การลบจุดที่สว่างเกินไป	`THRESH_TOZERO_INV`

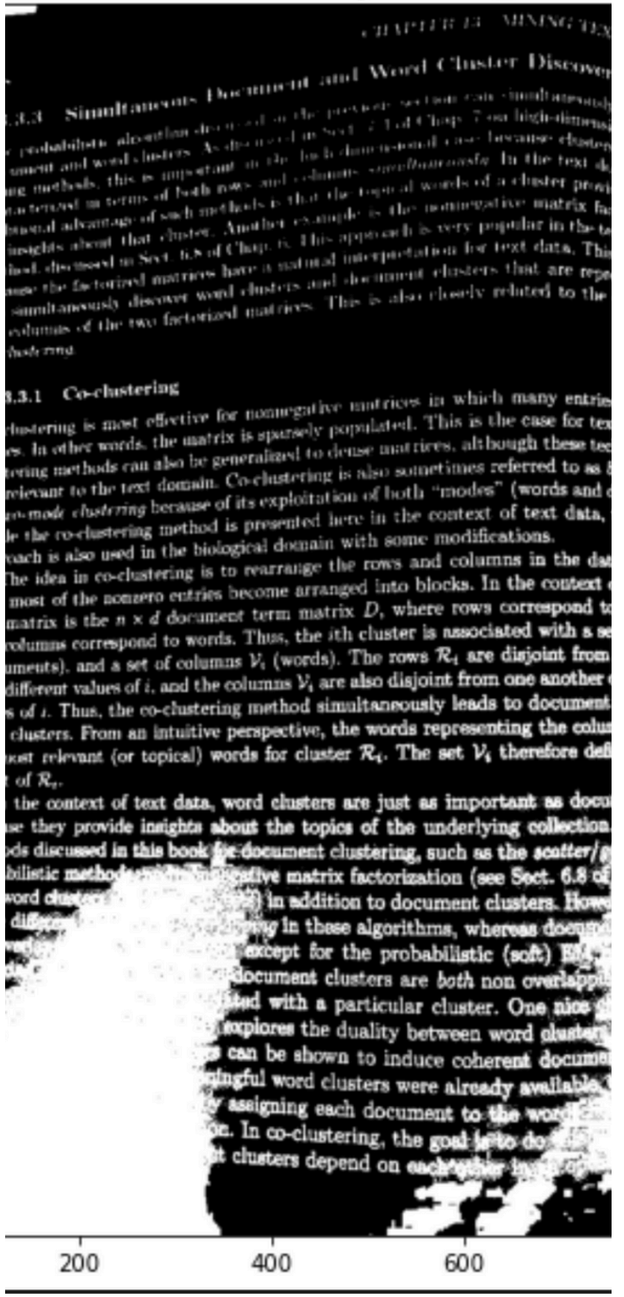
Original



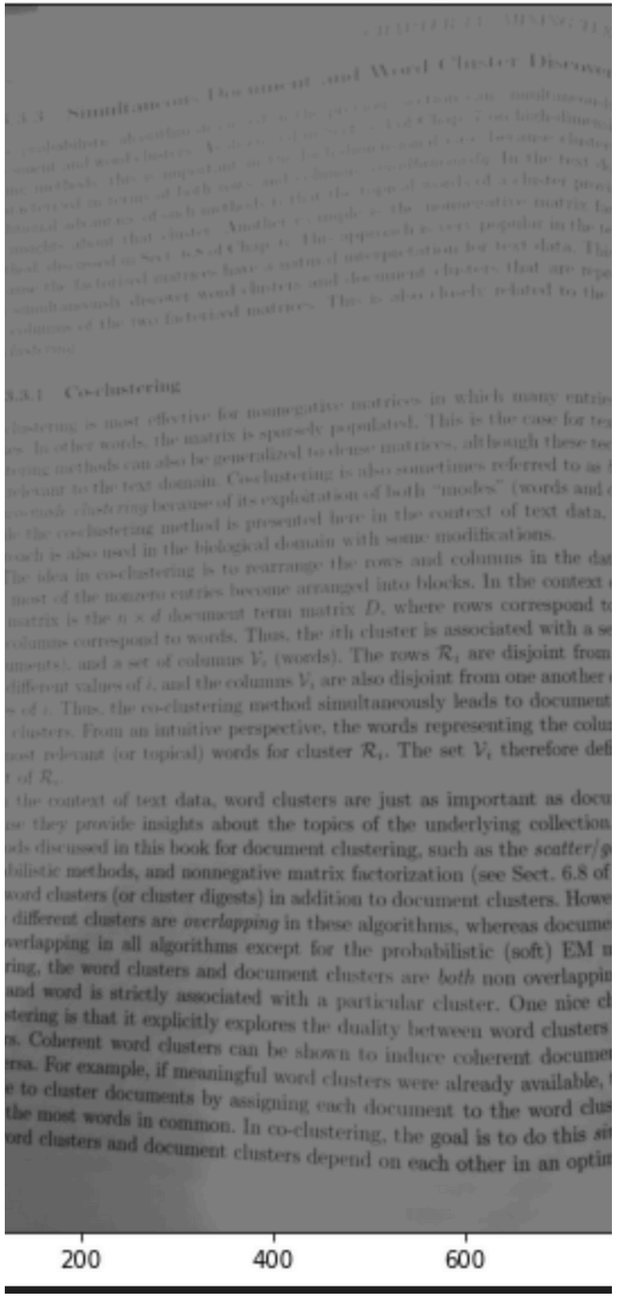
1 Threshold Binary @ 127



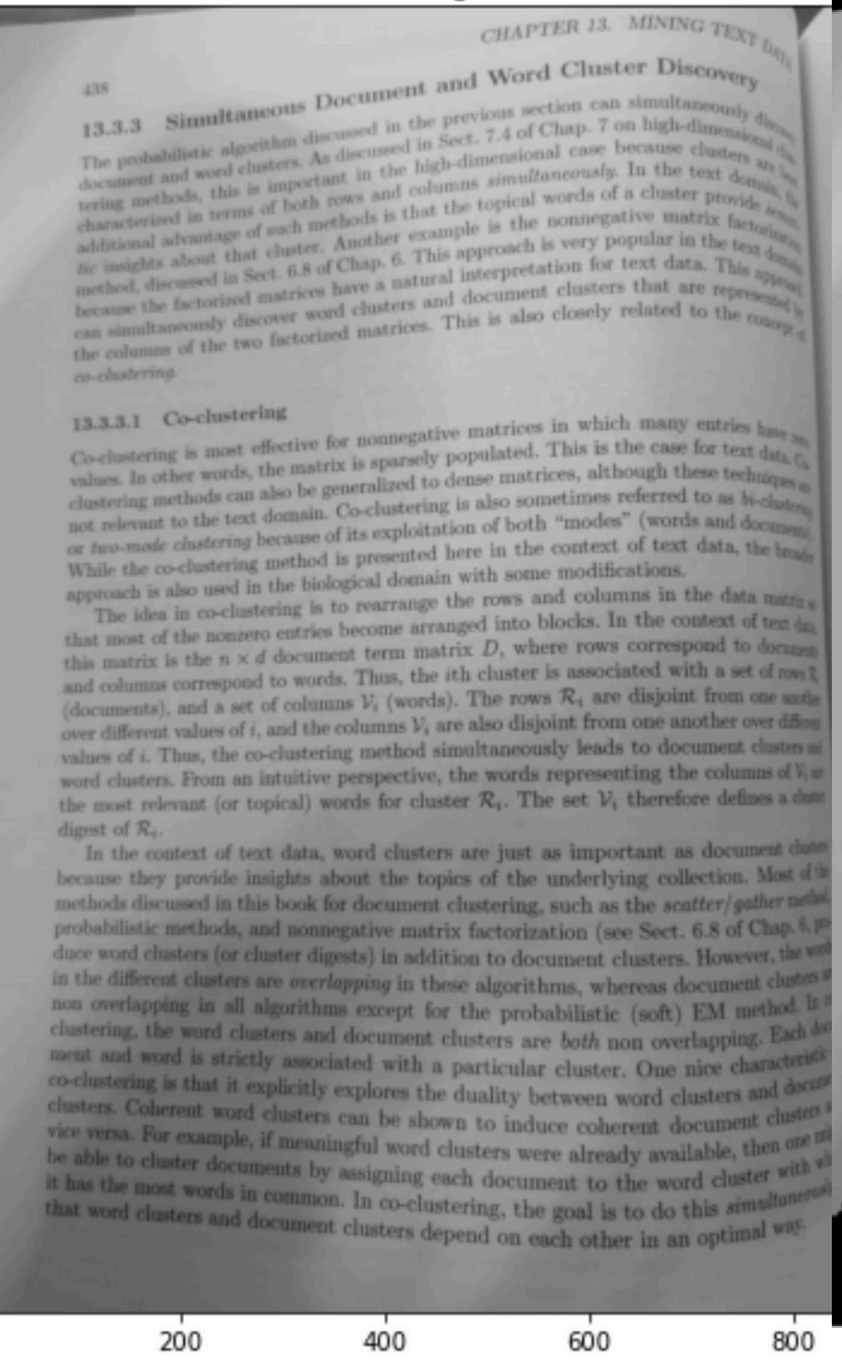
2 Threshold Binary Inverse @ 127



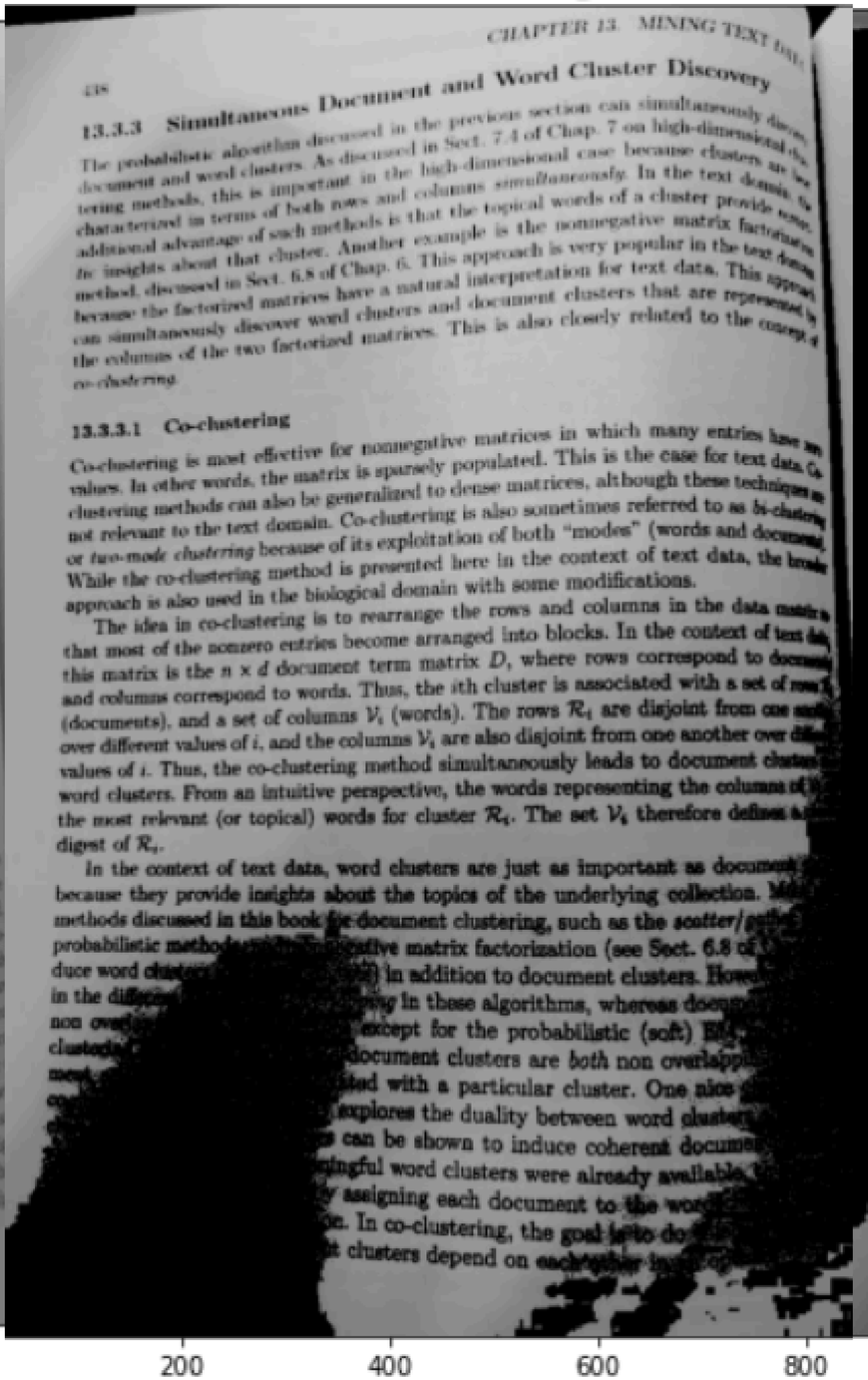
3 THRESH TRUNC @ 127



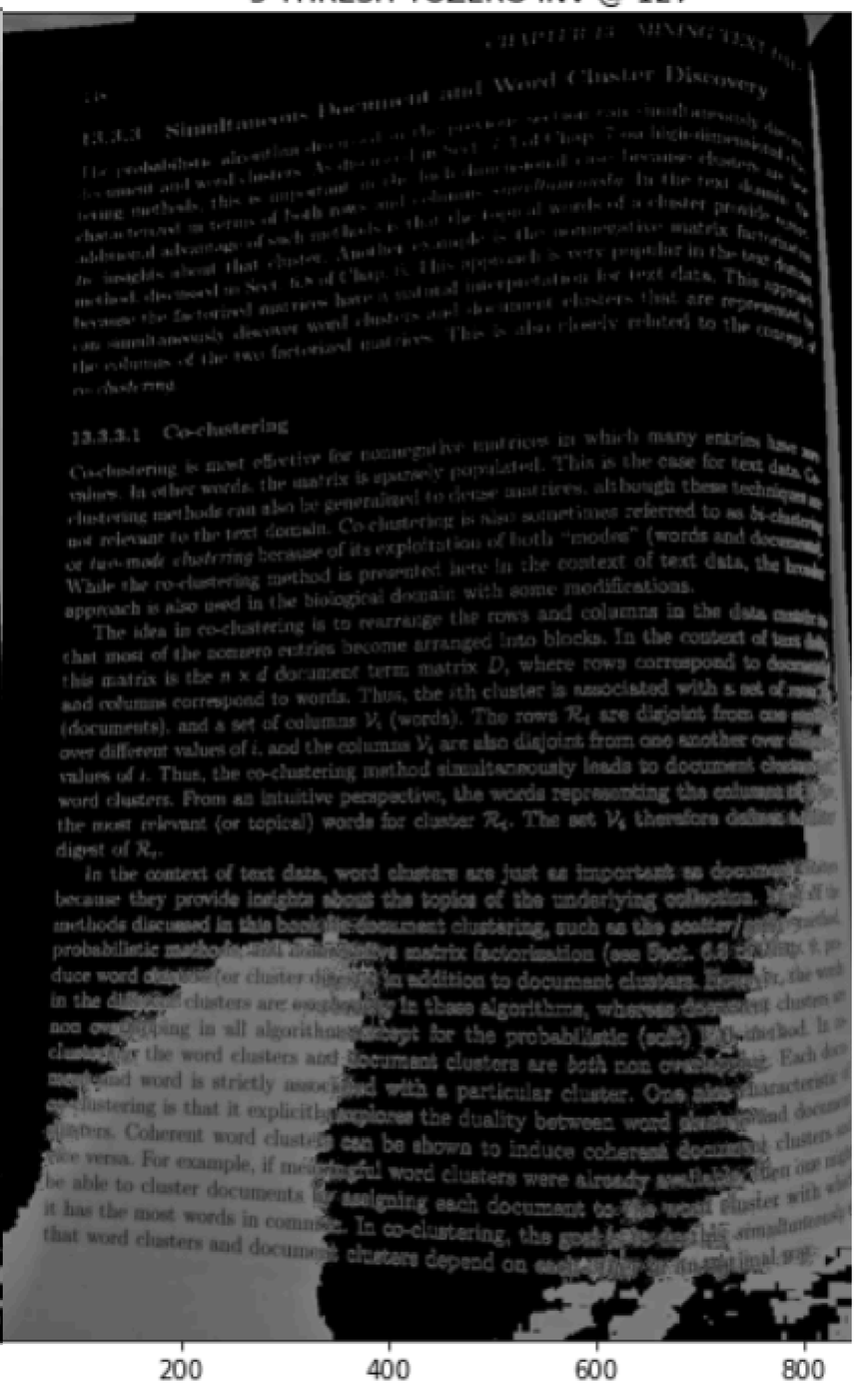
Original

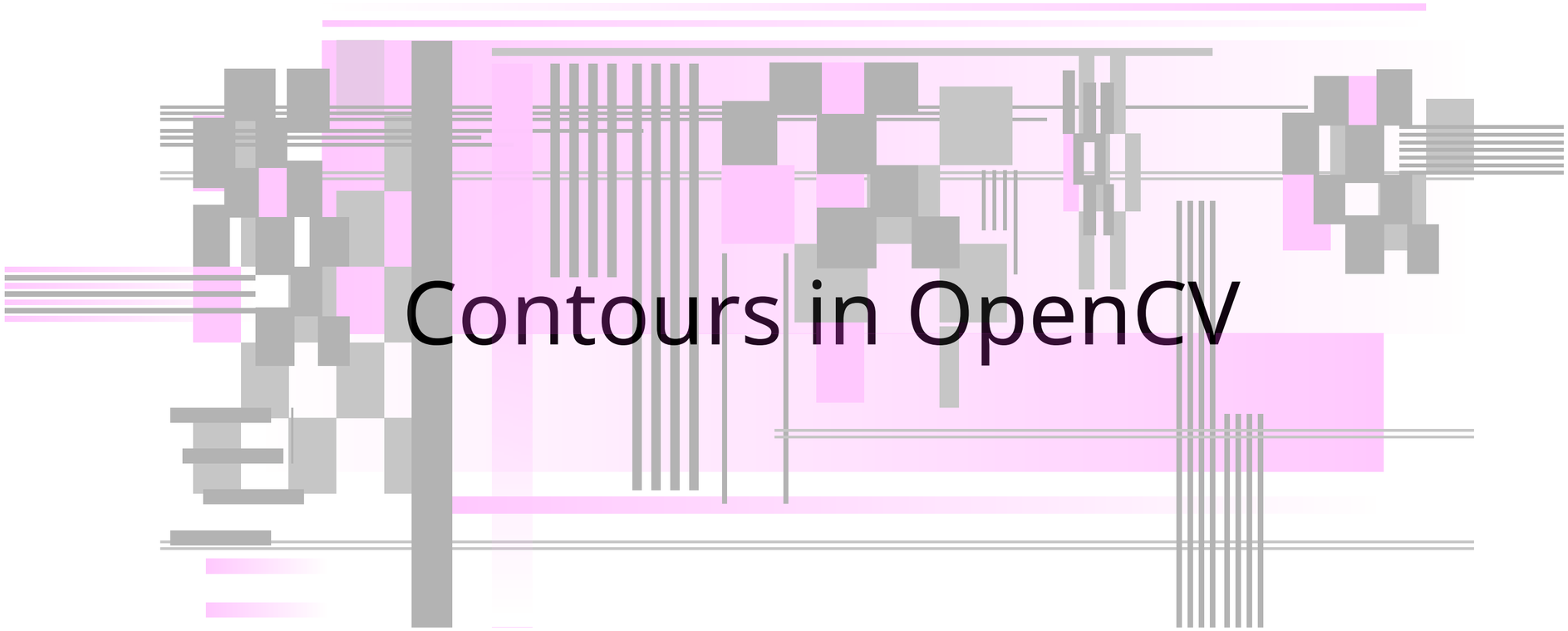


4 THRESH TOZERO @ 127



5 THRESH TOZERO INV @ 127




Contours in OpenCV

```
cv2.findContours(image, mode, method)
```

พารามิเตอร์

1. `image` **(ภาพ):**
   - ภาพที่ใช้ต้องอยู่ในรูปแบบไบนารี เช่น ภาพที่ผ่านการ Thresholding (`cv2.threshold`) หรือ Edge Detection (`cv2.Canny`) แล้ว
   - ตัวอย่างการแปลงภาพให้อยู่ในรูปแบบไบนารี:

     python — คัดลอกโค้ด

     ```
     _, binary_image = cv2.threshold(grayscale_image, 127, 255, cv2.THRESH_BINAR
     ```

2. `mode` **(โหมดการค้นหาเส้นขอบ):**
   - ระบุวิธีการค้นหาเส้นขอบ:
     - `cv2.RETR_EXTERNAL` :
       - ดึงเฉพาะเส้นขอบด้านนอกสุด
       - ใช้สำหรับการวิเคราะห์เฉพาะขอบนอกของวัตถุ
     - `cv2.RETR_LIST` :
       - ดึงเส้นขอบทั้งหมดในภาพ โดยไม่มีการจัดลำดับชั้น (Hierarchy)
     - `cv2.RETR_TREE` :
       - ดึงเส้นขอบทั้งหมดและสร้างลำดับชั้น (Hierarchy) ของเส้นขอบ เช่น เส้นขอบที่ซ้อนกัน
3. `method` **(วิธีการบันทึกจุดของเส้นขอบ):**
   - ระบุวิธีการเก็บข้อมูลจุดในเส้นขอบ:
     - `cv2.CHAIN_APPROX_NONE` :
       - เก็บจุดทั้งหมดในเส้นขอบ รวมถึงจุดที่อยู่บนเส้นตรงเดียวกัน (เปลืองพื้นที่หน่วยความจำ)
     - `cv2.CHAIN_APPROX_SIMPLE` :
       - เก็บเฉพาะจุดสำคัญของเส้นขอบ ช่วยลดการใช้หน่วยความจำได้มาก

Steps to Find Contours with OpenCV

1. **Load the Image**
   - Read the image using OpenCV.
   - Example:

```python
image = cv2.imread('image.png', cv2.IMREAD_COLOR)
```

2. **Convert the Image to Grayscale**
   - Convert the loaded image to grayscale to simplify processing.
   - Example:

```python
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

3. **Apply Preprocessing (Thresholding or Edge Detection)**
   - Create a binary image using thresholding or edge detection to highlight the contours.
   - Example using thresholding:

```python
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

   - Example using edge detection:

```python
binary = cv2.Canny(gray, 100, 200)
```

4. **Find Contours**
   - Use `cv2.findContours()` to detect contours in the binary image.
   - Example:

```python
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APF
```

5. **Draw Contours**
   - Use `cv2.drawContours()` to visualize the detected contours on the original image.
   - Example:

```python
cv2.drawContours(image, contours, -1, (0, 255, 0), 2)
```

6. **Display or Save the Result**
   - Show the resulting image with contours or save it to a file.
   - Example:

```python
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

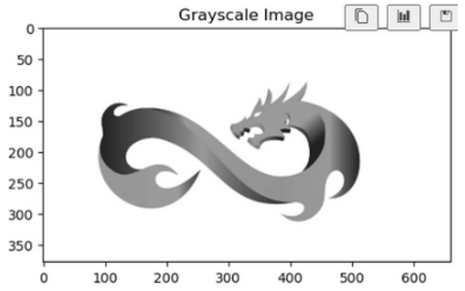
7. **(Optional) Analyze Contour Data**
   - Extract properties of the contours such as area, perimeter, or bounding boxes.
   - Example:

```python
for contour in contours:
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)
    print(f"Area: {area}, Perimeter: {perimeter}")
```

Summary

1. Load the image.
2. Convert it to grayscale.
3. Preprocess the image (Thresholding or Edge Detection).
4. Use `cv2.findContours()` to find contours.
5. Visualize contours with `cv2.drawContours()`.
6. Display or save the result.
7. Optionally analyze the detected contours.

```python
# Load the image
image = cv2.imread('./images/dragon.png', cv2.IMREAD_COLOR)

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```




```python
# Apply thresholding
_, binary = cv2.threshold(gray, 200 ,255, cv2.THRESH_BINARY)
```

```python
# Find contours
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# print the number of contours found
print('Number of contours found =', len(contours))
```

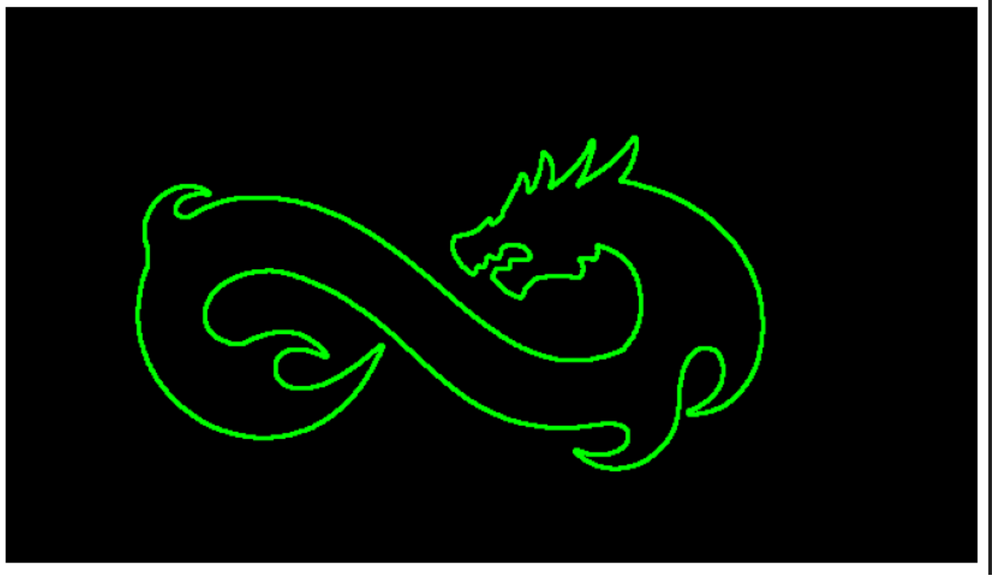
✓ 0.0s

```
Number of contours found = 4
```

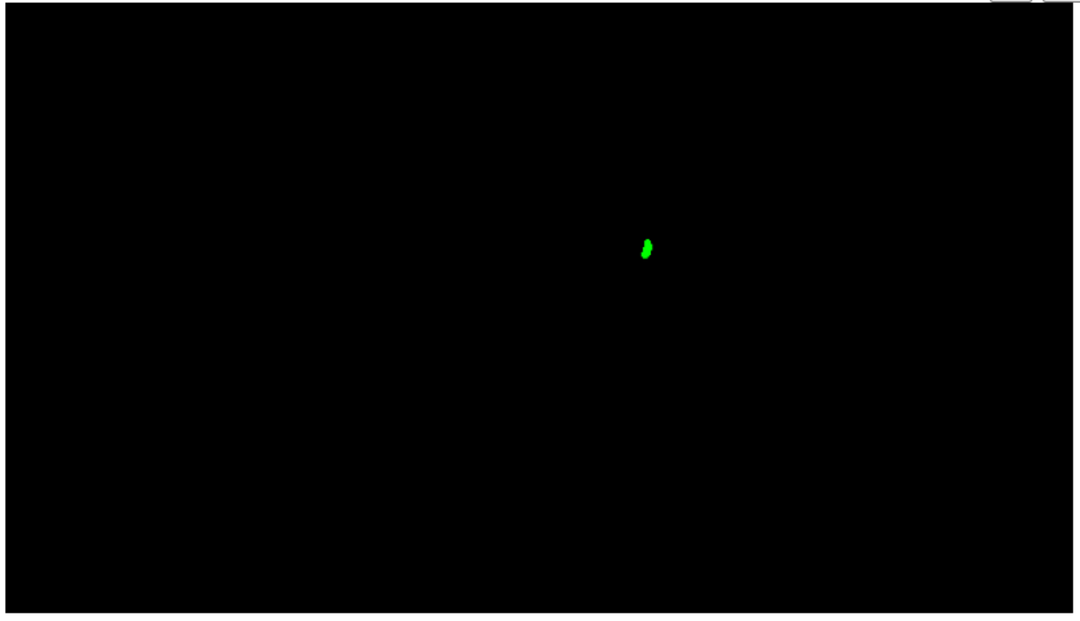
```python
# Iterate through each contour
for i, contour in enumerate(contours):
    # Create a blank image for each contour
    image_contours = np.zeros_like(image)
    # Draw the current contour
    cv2.drawContours(image_contours, [contour], -1, (0, 255, 0), 2)

    # Display the image with the current contour
    plt.figure(figsize=(12, 6))
    plt.imshow(cv2.cvtColor(image_contours, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for Matplotlib
    plt.title(f'Contour {i + 1}')
    plt.axis('off')
    plt.show()
```

Contour 1

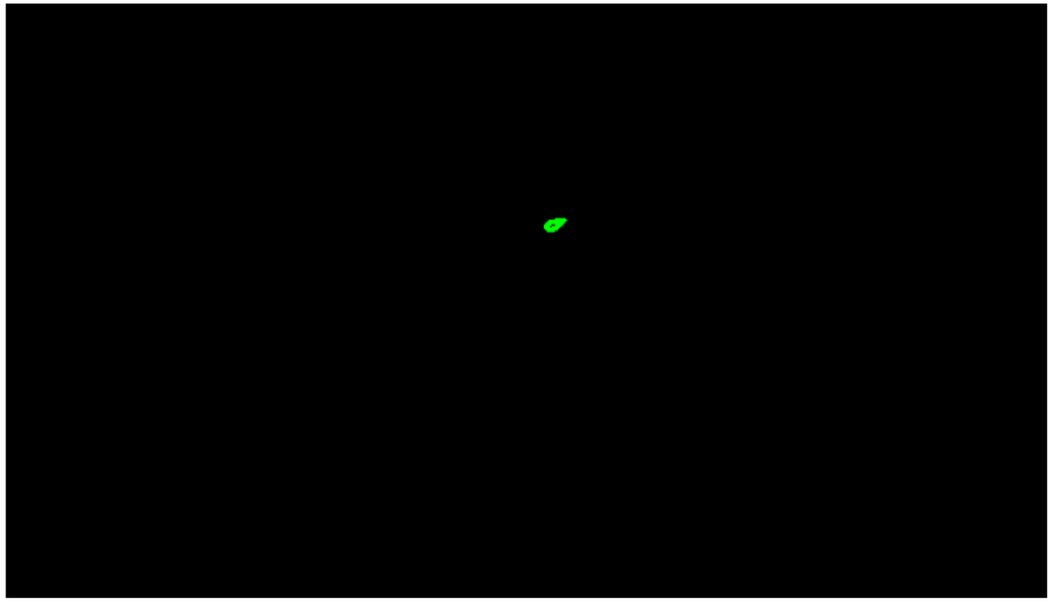
Contour 2



Contour 3

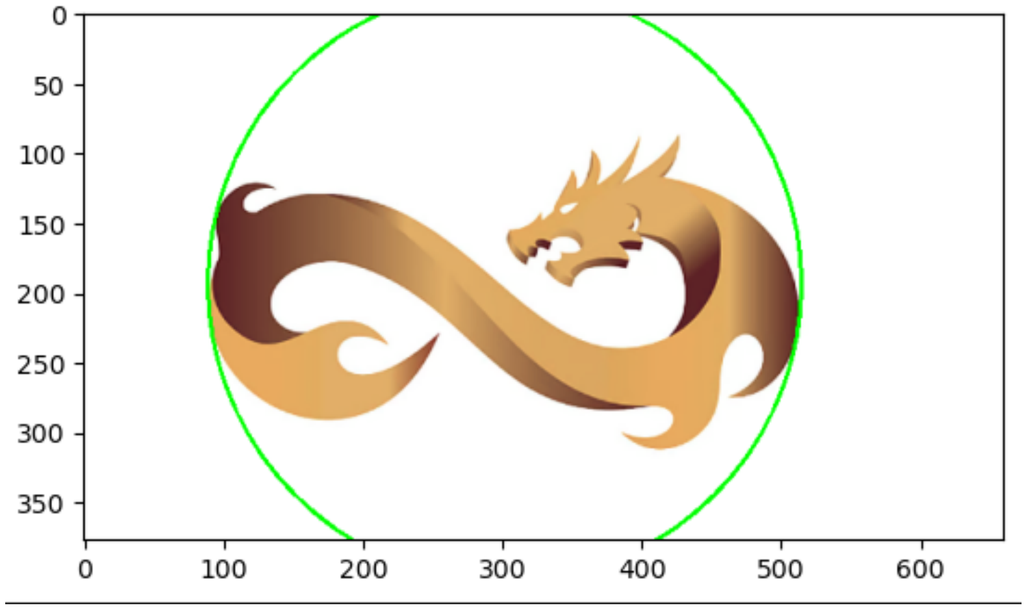


Contour 4

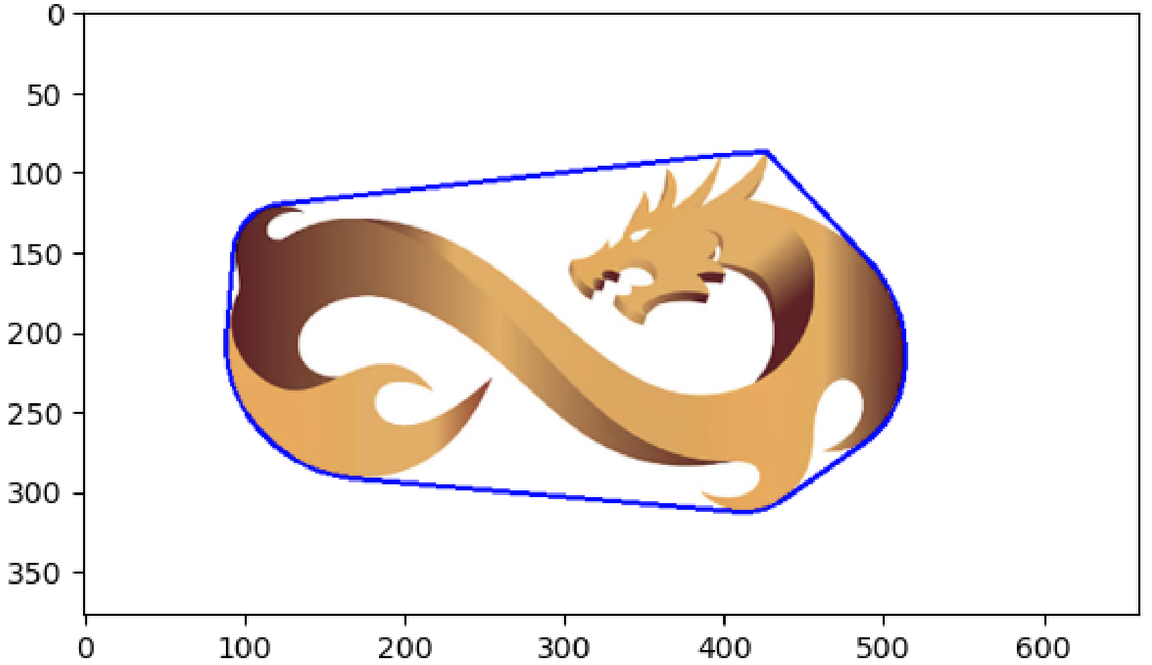


Draw Contours with Different Shapes(Circle ,Rectangle ,convexHull)

```python
contours,hierarchy = cv2.findContours(binary,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

contour = contours[1]

(x,y),radius = cv2.minEnclosingCircle(contour)
center = (int(x),int(y))
radius = int(radius)

cv2.circle(image_rgb_copy2,center,radius,(0,255,0),2);

# draw circle to original image
plt.imshow(image_rgb_copy2)
```

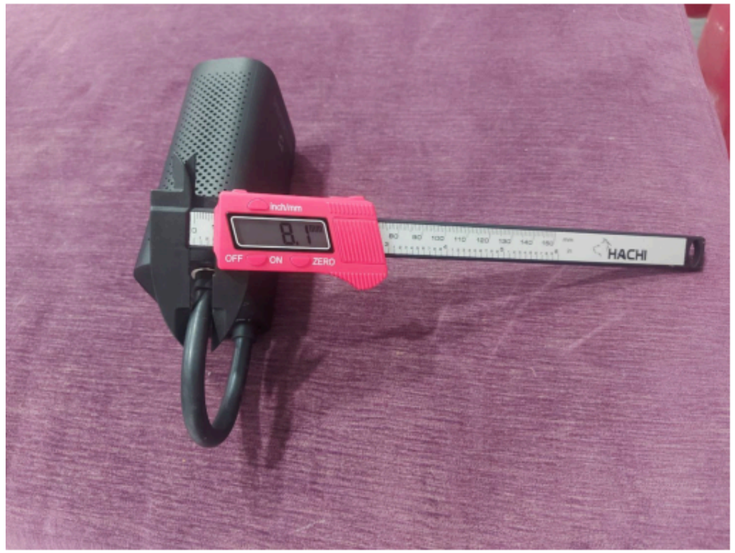



```python
contours,hierarchy = cv2.findContours(binary,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

# Get the first contour from the list
contour = contours[1]

# Compute the convex hull of the contour
hull = cv2.convexHull(contour)

# draw convexHull to original image
cv2.drawContours(image_rgb_copy3, [hull], 0, (0, 0, 255), 2)


# draw circle to original image
plt.imshow(image_rgb_copy3)
```




```python
contours,hierarchy = cv2.findContours(binary,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

# Get the first contour from the list
contour = contours[1]

# Compute the convex hull of the contour
hull = cv2.convexHull(contour)

# draw convexHull to original image
cv2.drawContours(image_rgb_copy3, [hull], 0, (0, 0, 255), 2)
```

Home Work


inch/mm
81
OFF ON ZERO
HACHI


inch/mm
456
OFF ON ZERO
HACHI


inch/mm
123.9
OFF ON ZERO


inch/mm
OFF ON ZERO