

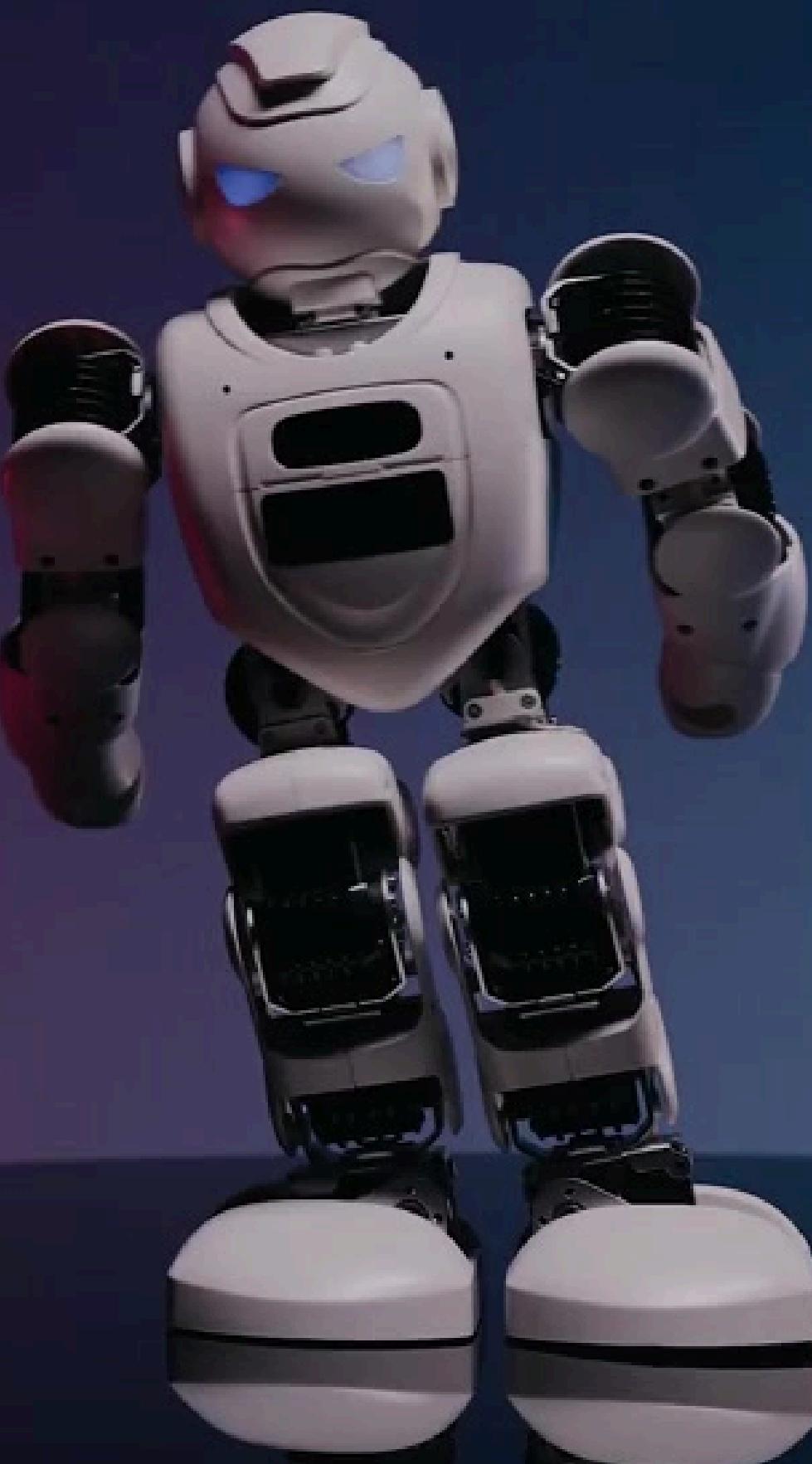


WEEK 4

DEEP LEARNING FOR COMPUTER VISION

UNLIMITED

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



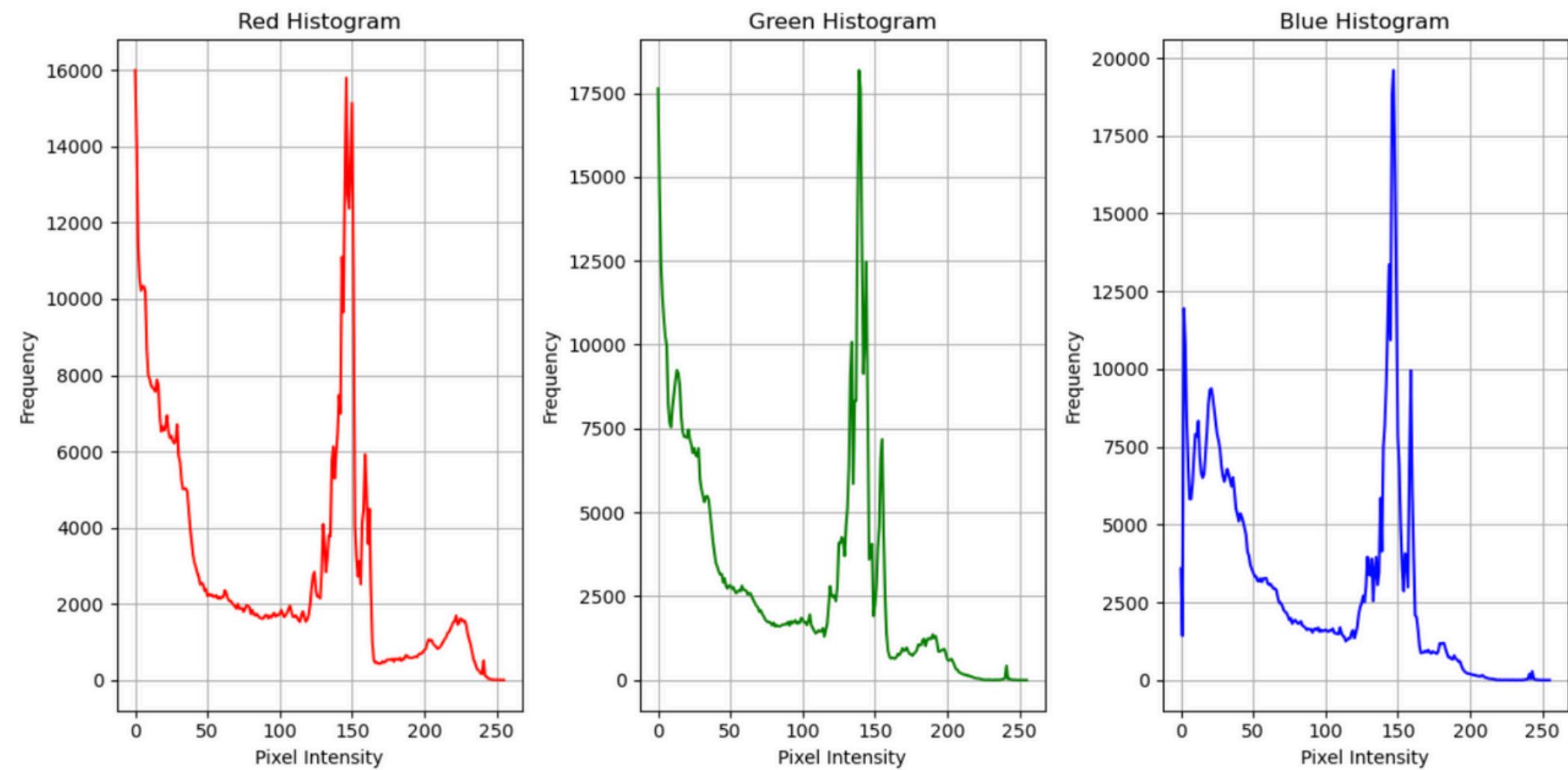


Histograms and K-Means Clustering for Dominant Colors

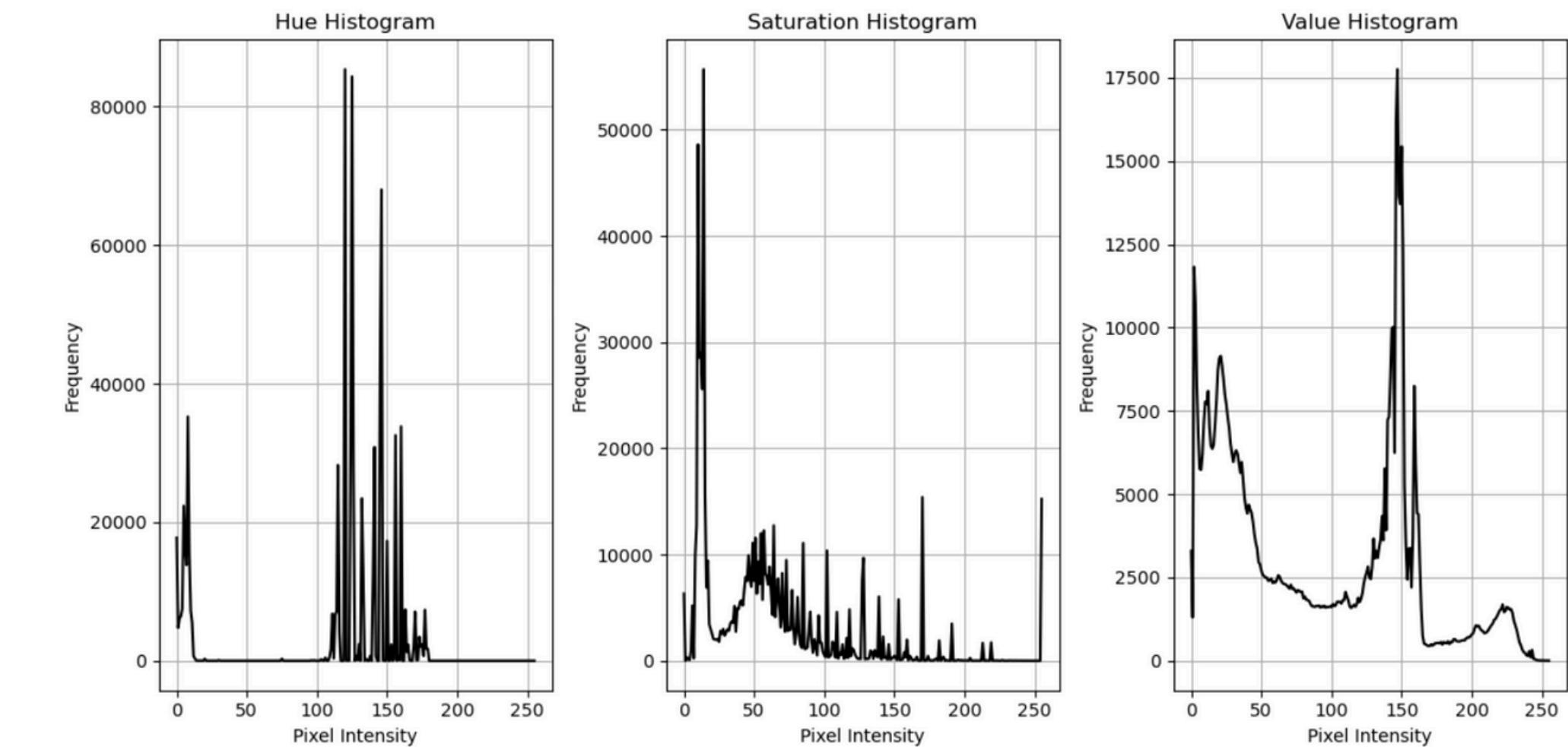
Original Image (RGB)

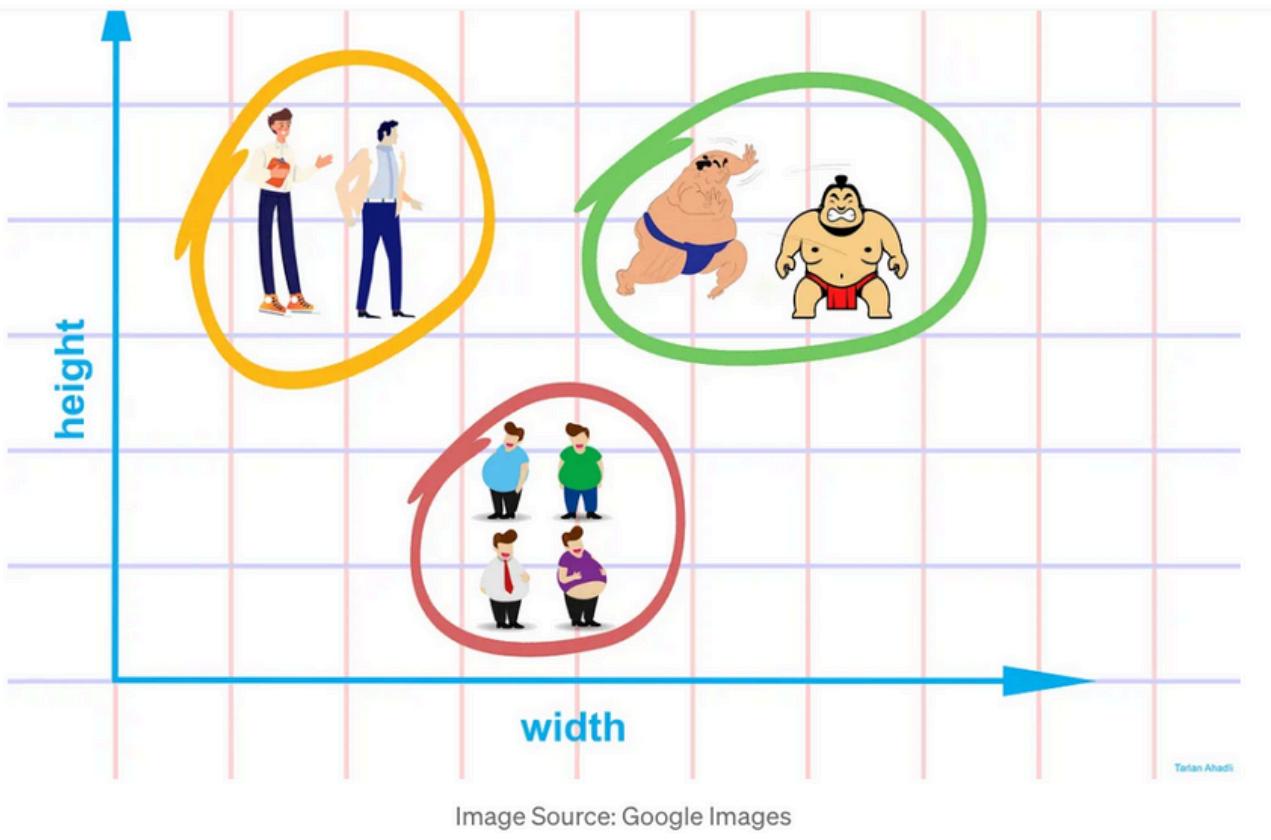


RGB Histogram

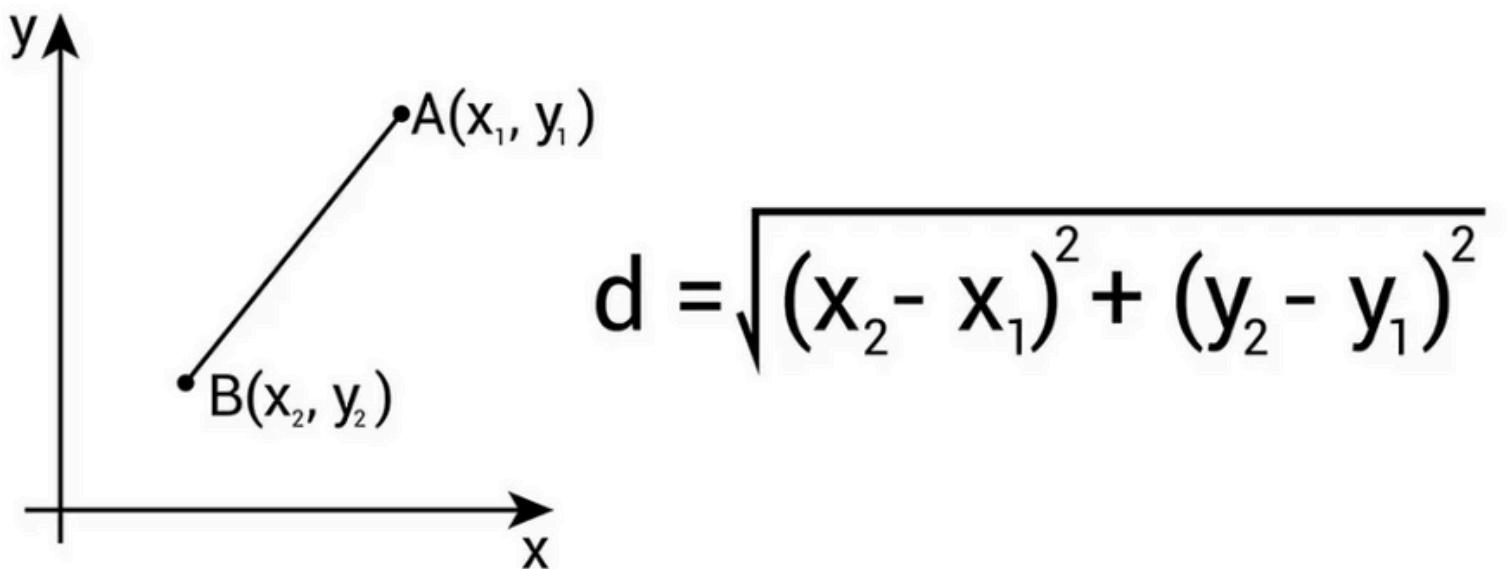


HSV Histogram



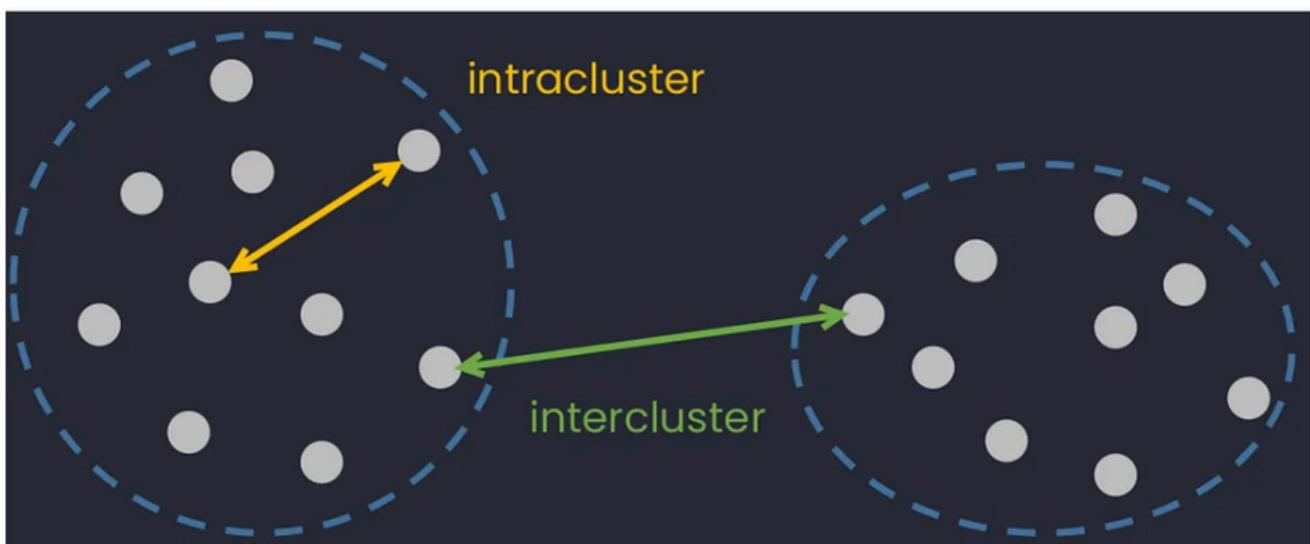


Distance Formula



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.



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

✓ 0.0s

(1048, 732, 3)

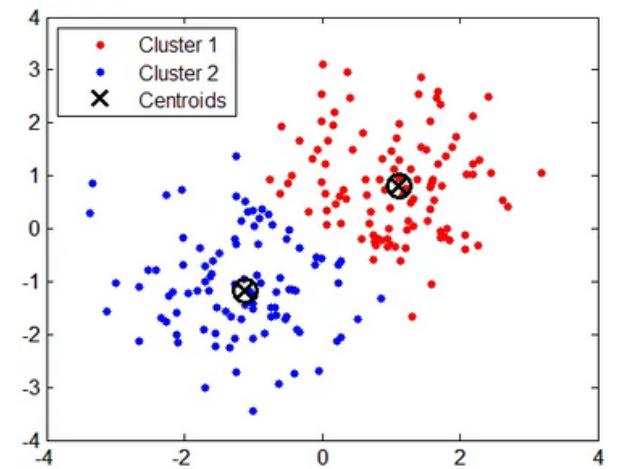
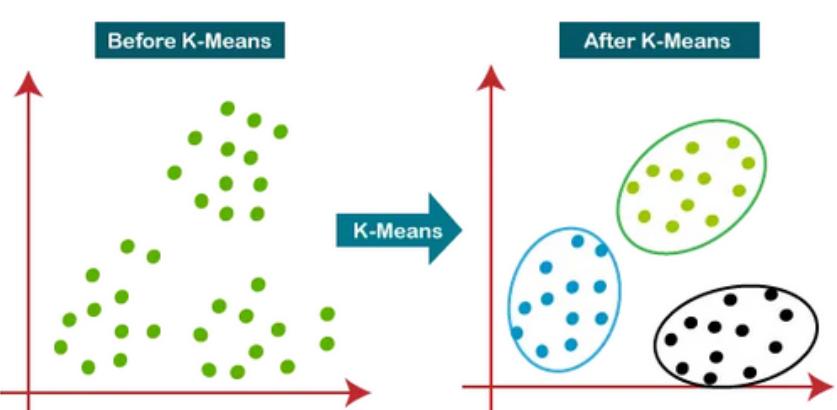
```
# Reshape the Image to a 2D Array of Pixels in HSV
pixels_hsv = image_hsv.reshape((-1, 3))
pixels_hsv.shape
```

✓ 0.0s

(767136, 3)

Find Centers as dominant 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)
```

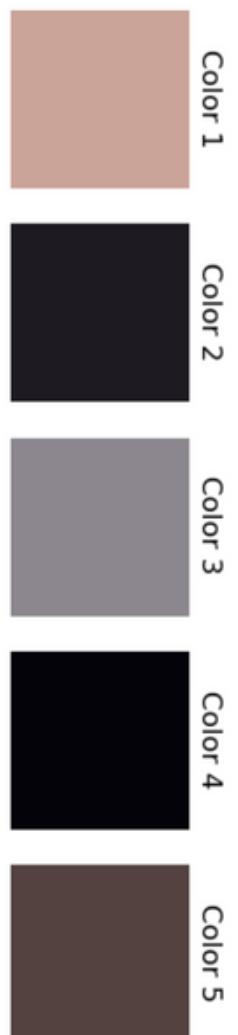


```
# Define the Number of Clusters (Colors)
num_colors = 5

# 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]
```



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]

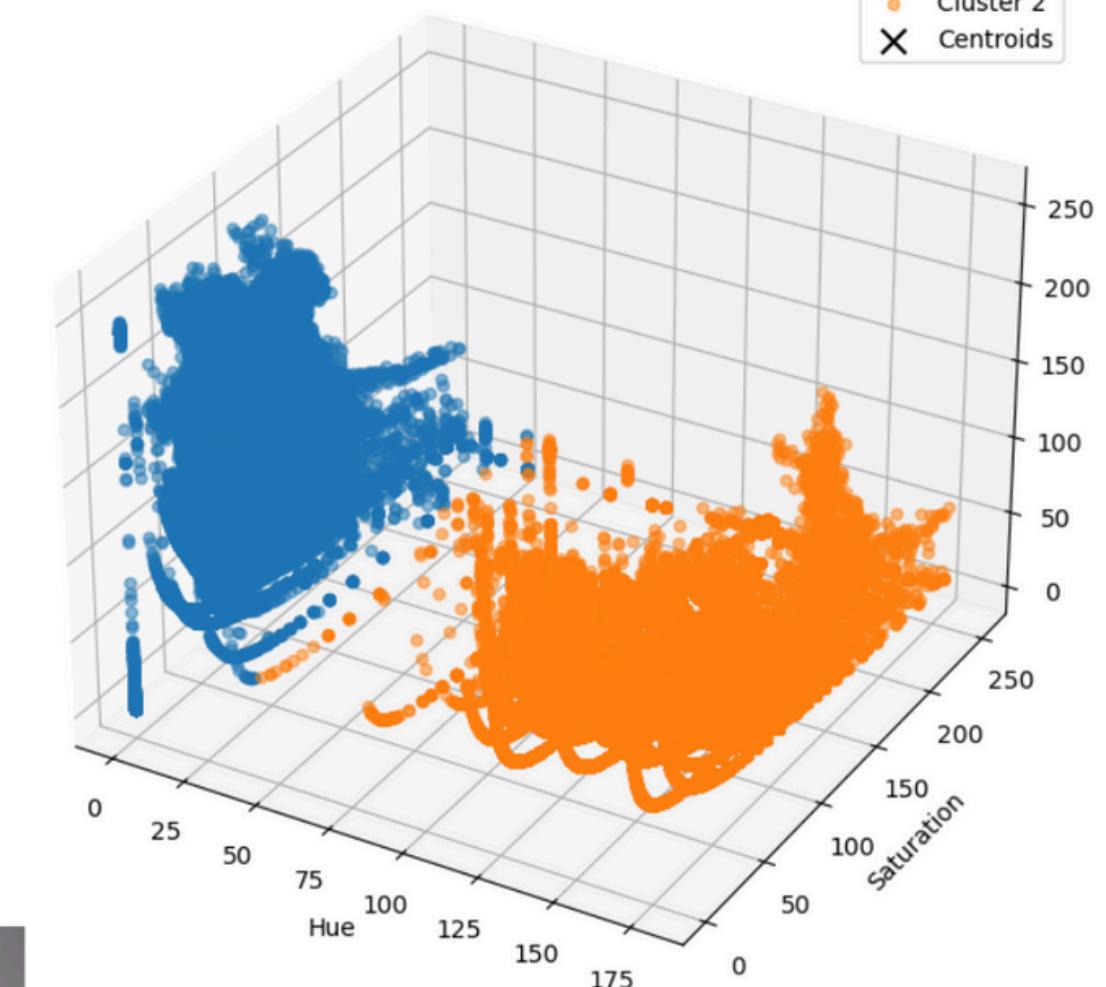
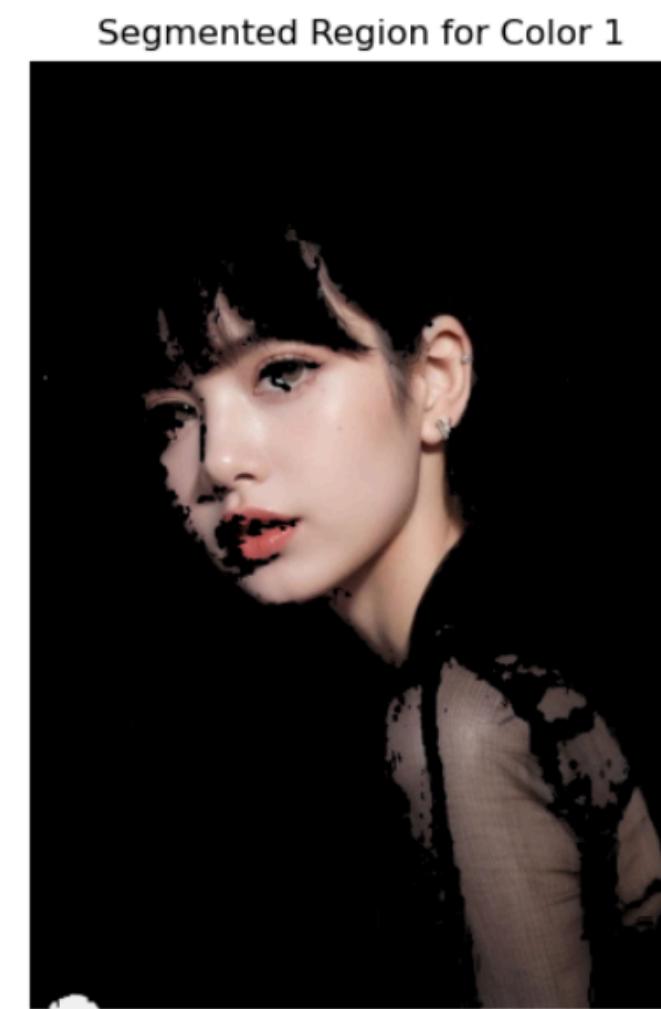
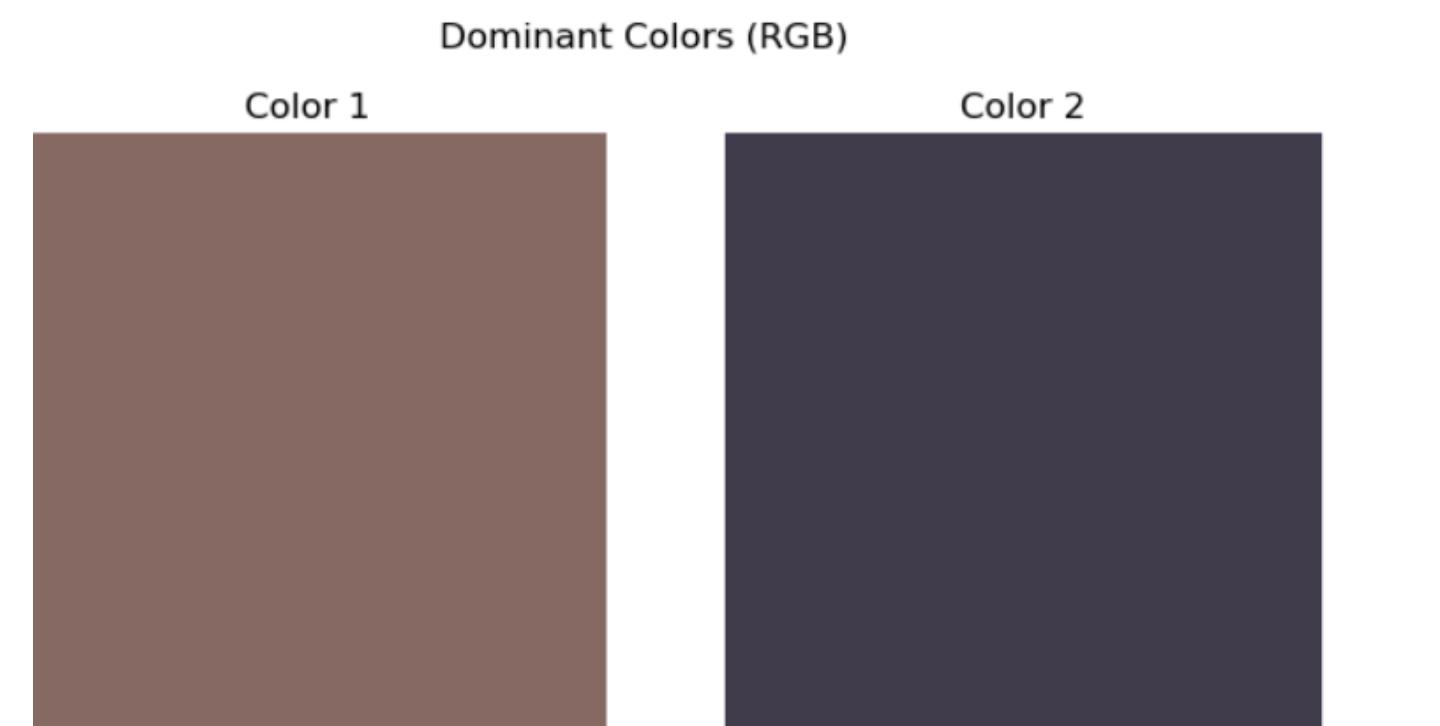
# Assign Labels to Each Pixel
labels = kmeans.labels_

# 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()
```

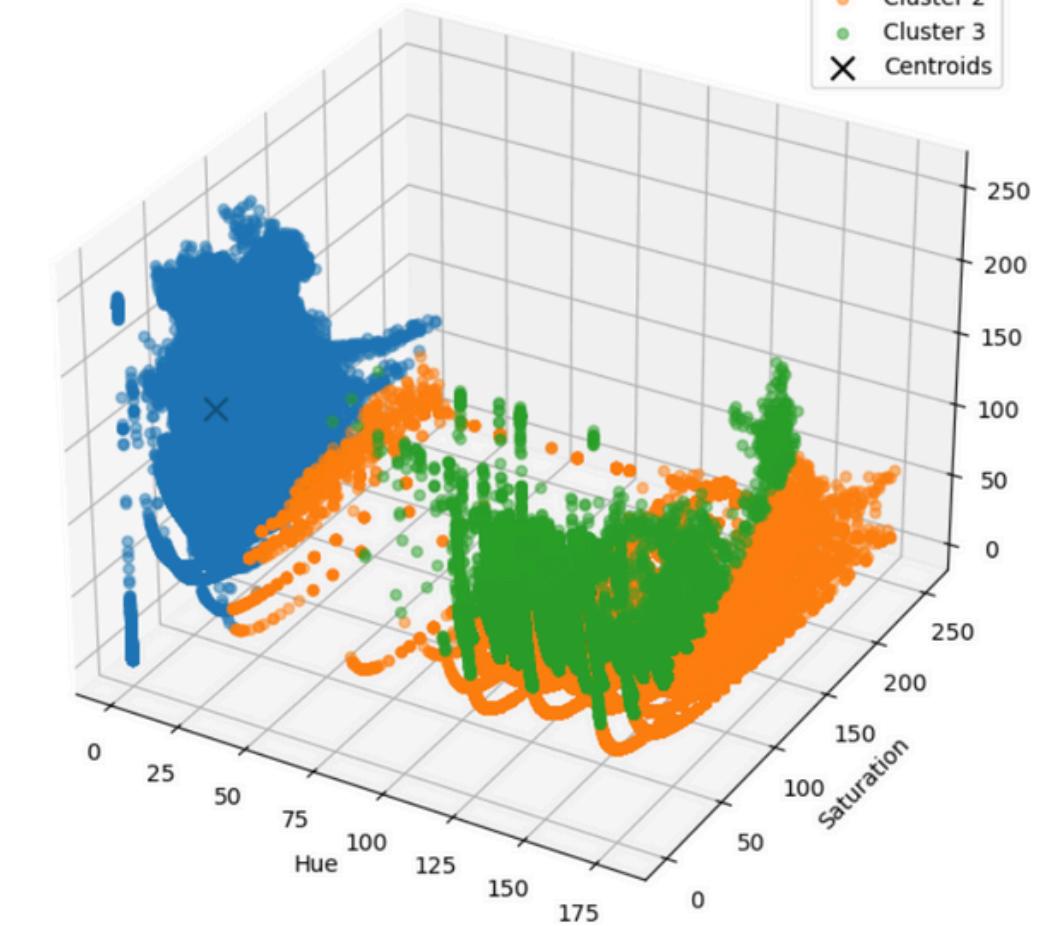
3D Visualization of Clusters in HSV Space

● Cluster 1
● Cluster 2
X Centroids



3D Visualization of Clusters in HSV Space

- Cluster 1
- Cluster 2
- Cluster 3
- ✖ Centroids



Dominant Colors (RGB)

Original Image (RGB)



Color 1



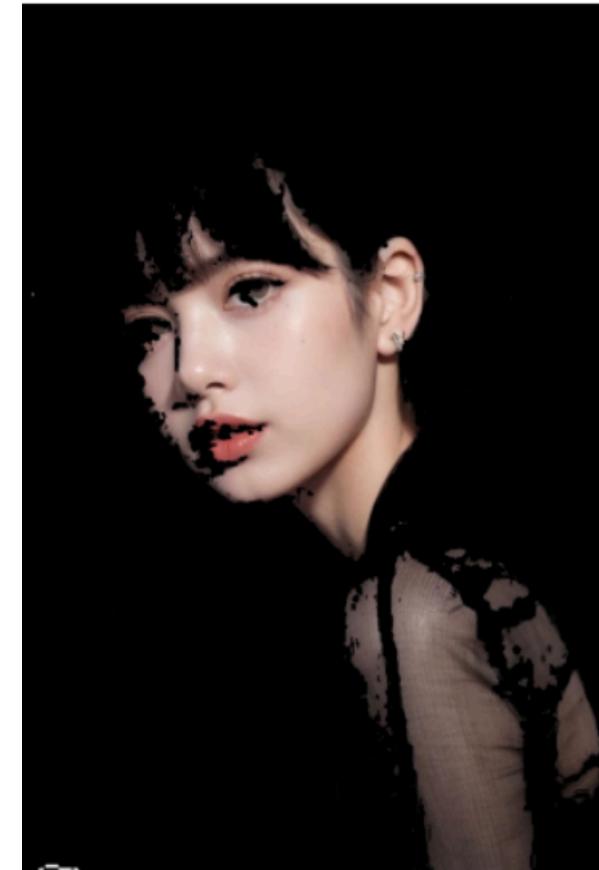
Color 2



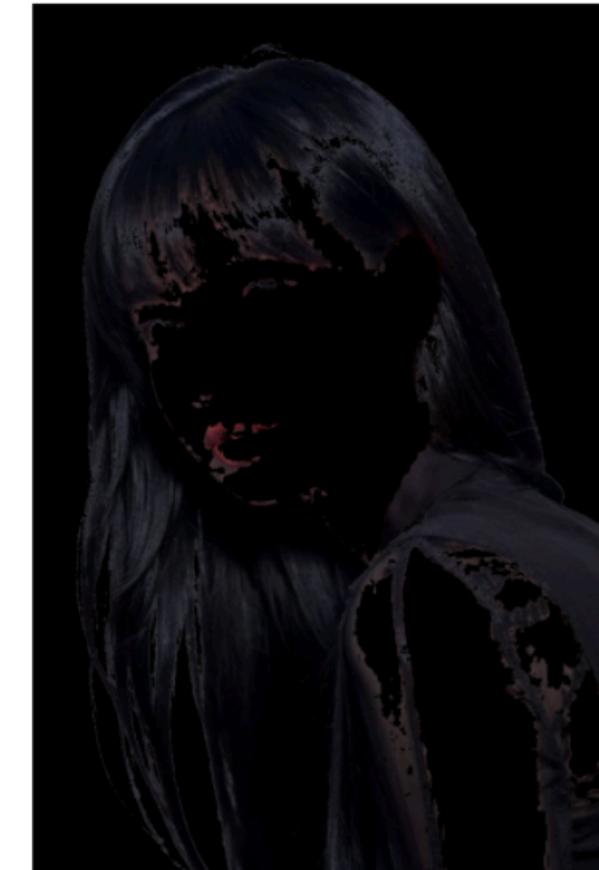
Color 3



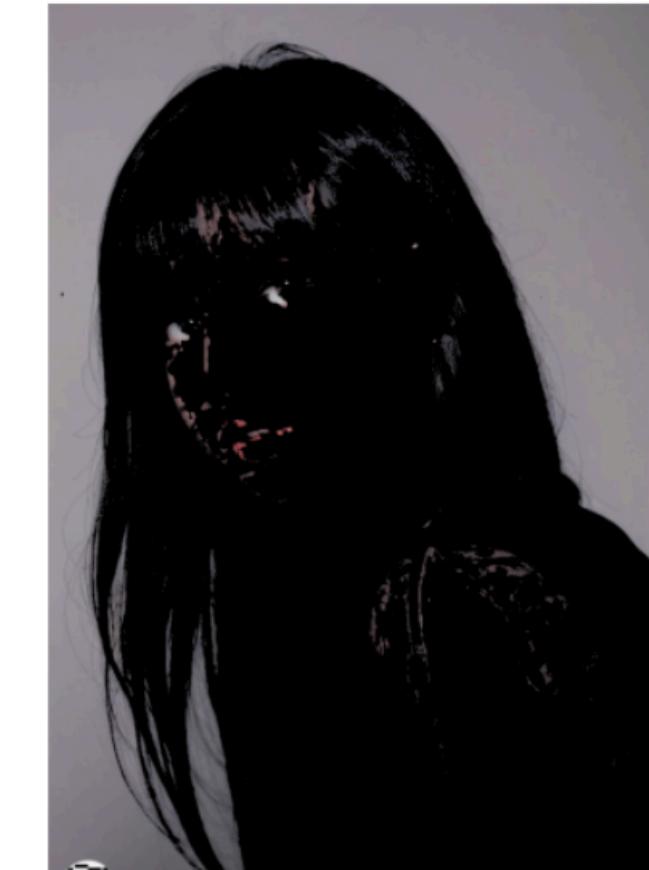
Segmented Region for Color 1



Segmented Region for Color 2

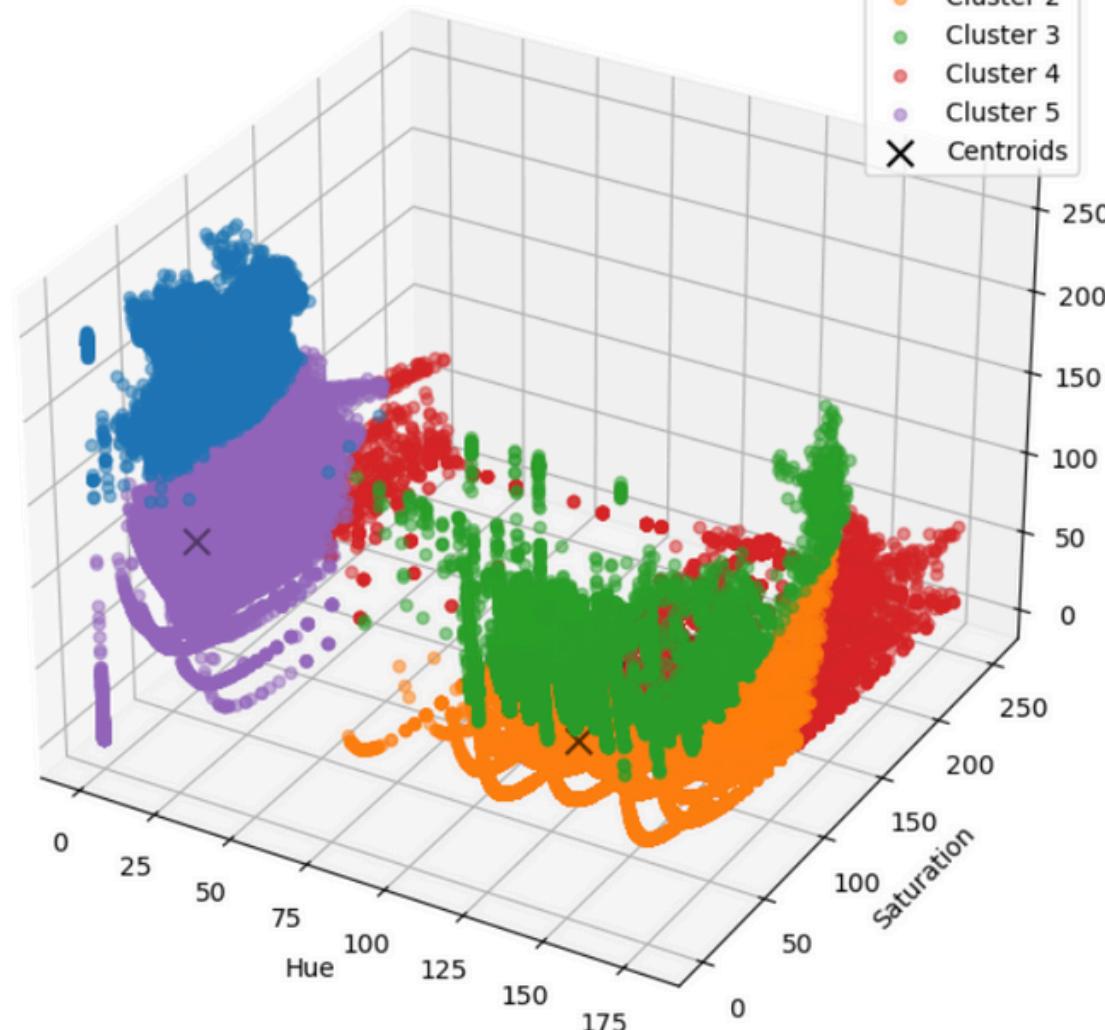


Segmented Region for Color 3



3D Visualization of Clusters in HSV Space

- Cluster 1
- Cluster 2
- Cluster 3
- Cluster 4
- Cluster 5
- X Centroids



Original Image (RGB)



Color 1



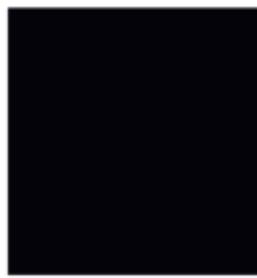
Color 2



Color 3



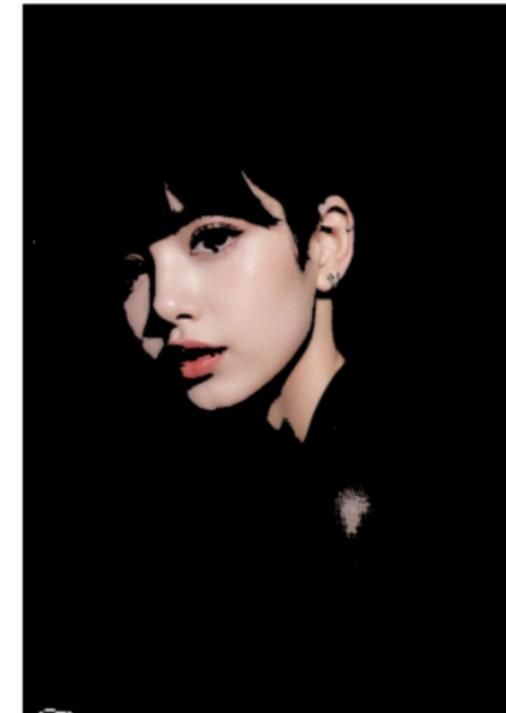
Color 4



Color 5



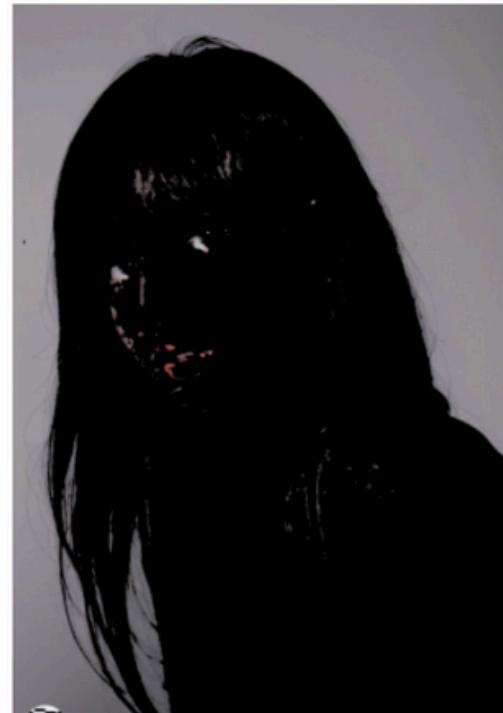
Segmented Region for Color 1



Segmented Region for Color 2



Segmented Region for Color 3



Segmented Region for Color 4



Segmented Region for Color 5

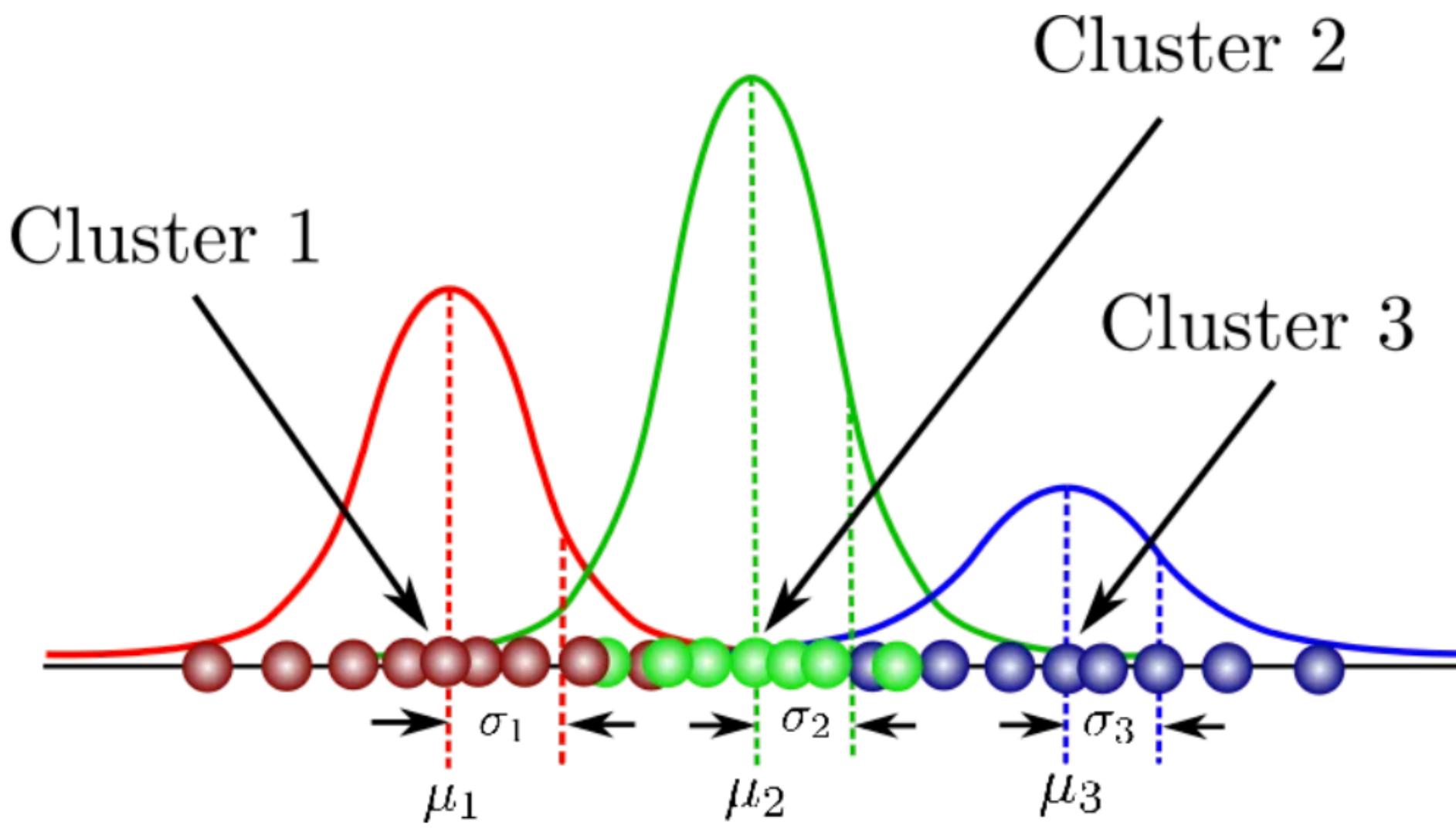
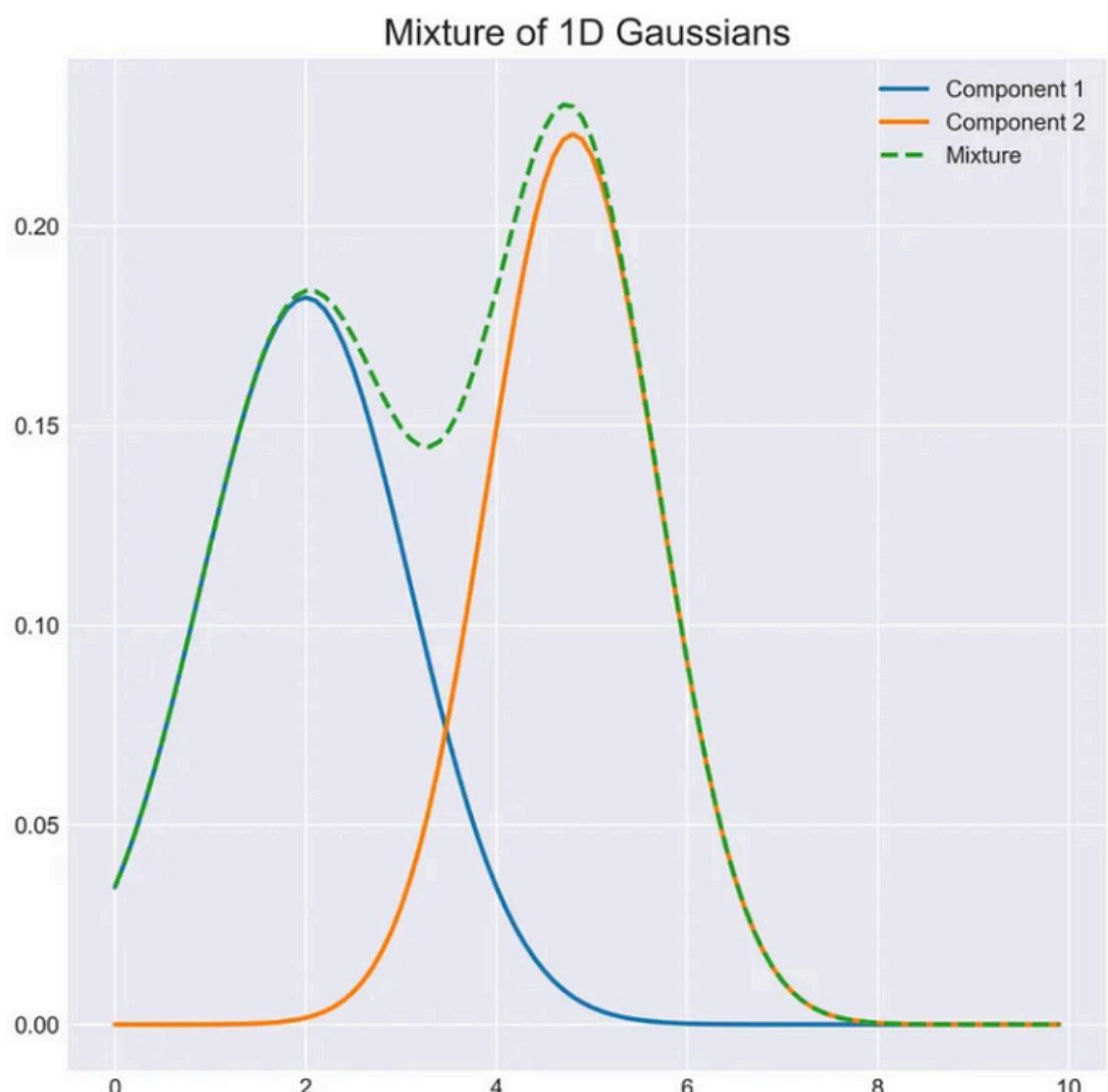




Histograms and Gaussian Mixture Models Clustering for Dominant Colors

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.



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

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.

How does it work?

As per the name states, GMM would incorporate all of the Gaussian Distributions into one model. Assumed that there are 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.

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

Posterior Probability Formula

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

Probability Density Formula

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

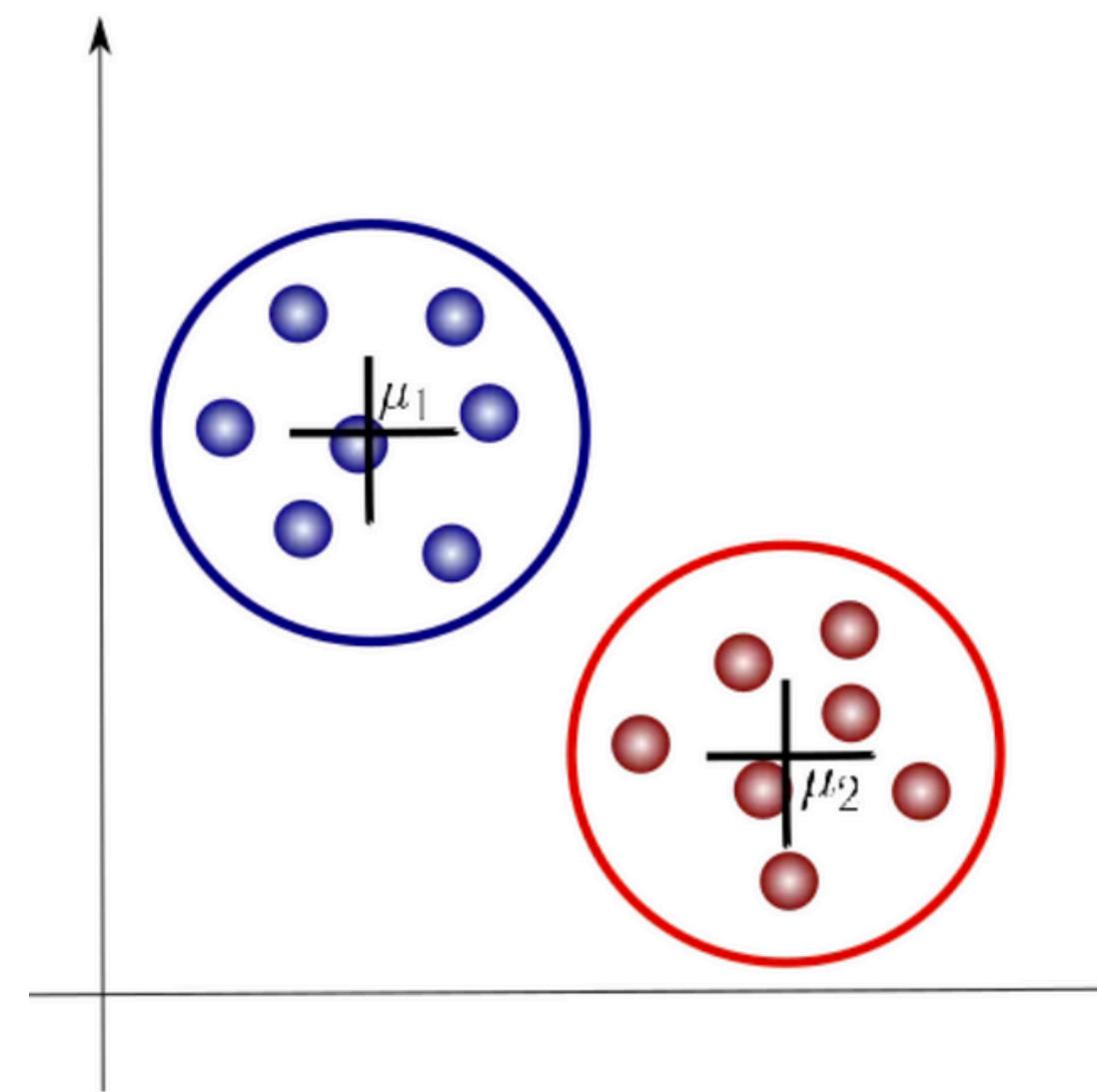
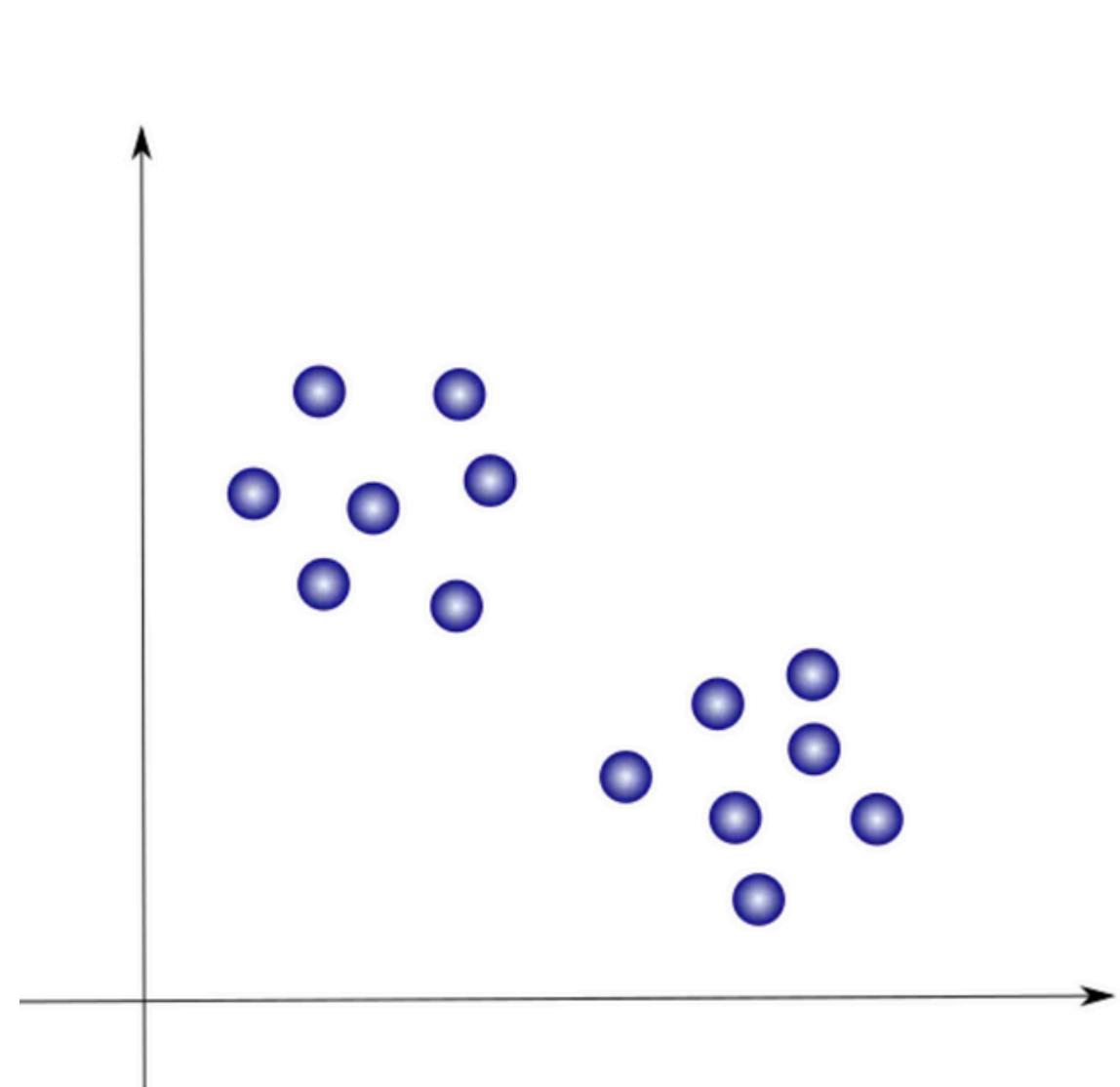
The next step after obtaining its' posterior probability, would be to update the model's parameters which are mixing coefficients and mean vectors.

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

Mixing Coefficients

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

Mean Vectors



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

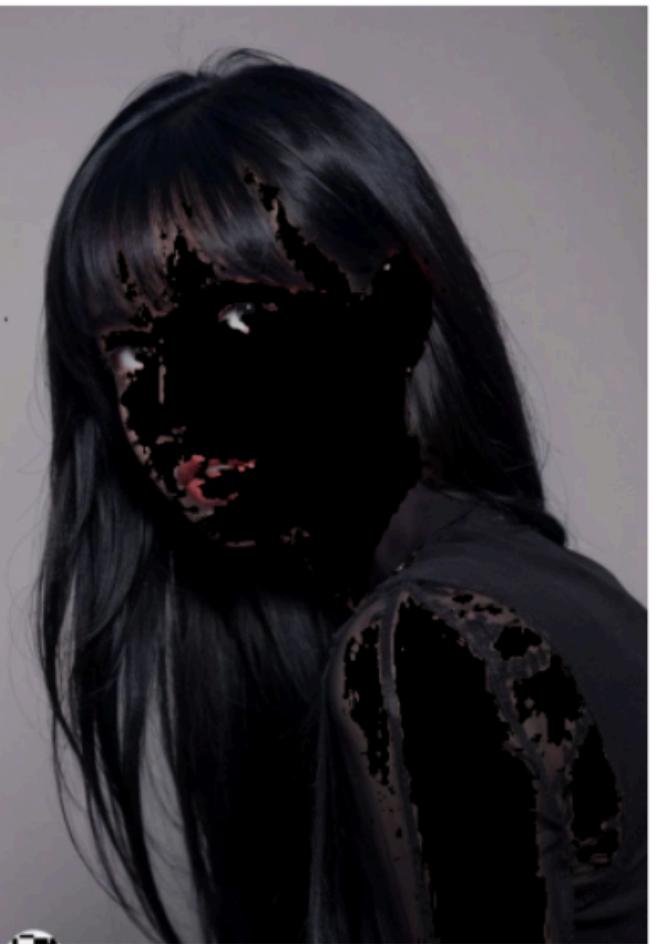
Segment 0



Segment 0



Segment 1



Segment 1

```
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
ความซิดหุ่นของรูปร่างกลุ่ม	จำกัดที่รูปทรงวงกลมเท่านั้น เนื่องจากใช้ Euclidean Distance เป็นเกณฑ์	ยืดหยุ่นกว่า สามารถจัดการกับกลุ่มที่มีการกระจายแบบ Gaussian ที่แตกต่างกันได้
ความเร็วในการทำงาน	เร็วกว่า เพราะมีการคำนวณแบบ Iterative และเรียบง่าย	ช้ากว่า เพราะมีการใช้ Expectation-Maximization (EM) Algorithm
ความสามารถในการแบ่งกลุ่มขั้นชั้น	จำกัดเมื่อกลุ่มมีความซับซ้อน เช่น ข้อมูลที่มีการซ้อนทับของกลุ่ม	เหมาะสมสำหรับข้อมูลที่ซับซ้อนและมีการซ้อนทับระหว่างกลุ่ม
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. ไม่เหมาะสมกับข้อมูลซับซ้อนหรือการกระจายแบบไม่สมมาตร 2. ไม่ต่อค่าเริ่มต้นของ 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 ซึ่งเป็นกระบวนการแปลงภาพให้ออกในรูปแบบที่ง่ายขึ้น เช่น การแปลงเป็นภาพ Binary (ขาว-ดำ) หรือการลดระดับความเข้ม (Intensity Levels) ของพิกเซลในภาพ โดยใช้ค่ากำหนด (Threshold) เพื่อแบ่งภาพออกเป็นส่วนต่างๆ

การใช้งาน Thresholding

Thresholding ใช้ในงานประมวลผลภาพที่ต้องการ:

- แยกวัตถุออกจากพื้นหลัง (Object Segmentation)
- ตรวจจับเขต (Contour Detection)
- ลดความซับซ้อนของภาพ เพื่อให้ประมวลผลต่อได้ง่ายขึ้น เช่น สำหรับการรู้จำตัวอักษร (OCR)

โครงสร้างฟังก์ชัน

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

พารามิเตอร์

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

```
1. retval:

- ค่า Threshold ที่ใช้ (สำคัญในการถ้าที่ใช้ THRESH_OTSU หรือ THRESH_TRIANGLE)

2. thresholded_image:

- ภาพที่ผ่านการ Thresholding แล้ว

```

```
# 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

การแยกข้อความออกจากภาพพื้นหลัง

ประเภท Thresholding ที่เหมาะสม

`THRESH_BINARY` หรือ `THRESH_BINARY_INV`

การวิเคราะห์ภาพที่มีความสว่างเกินไป

`THRESH_TRUNC`

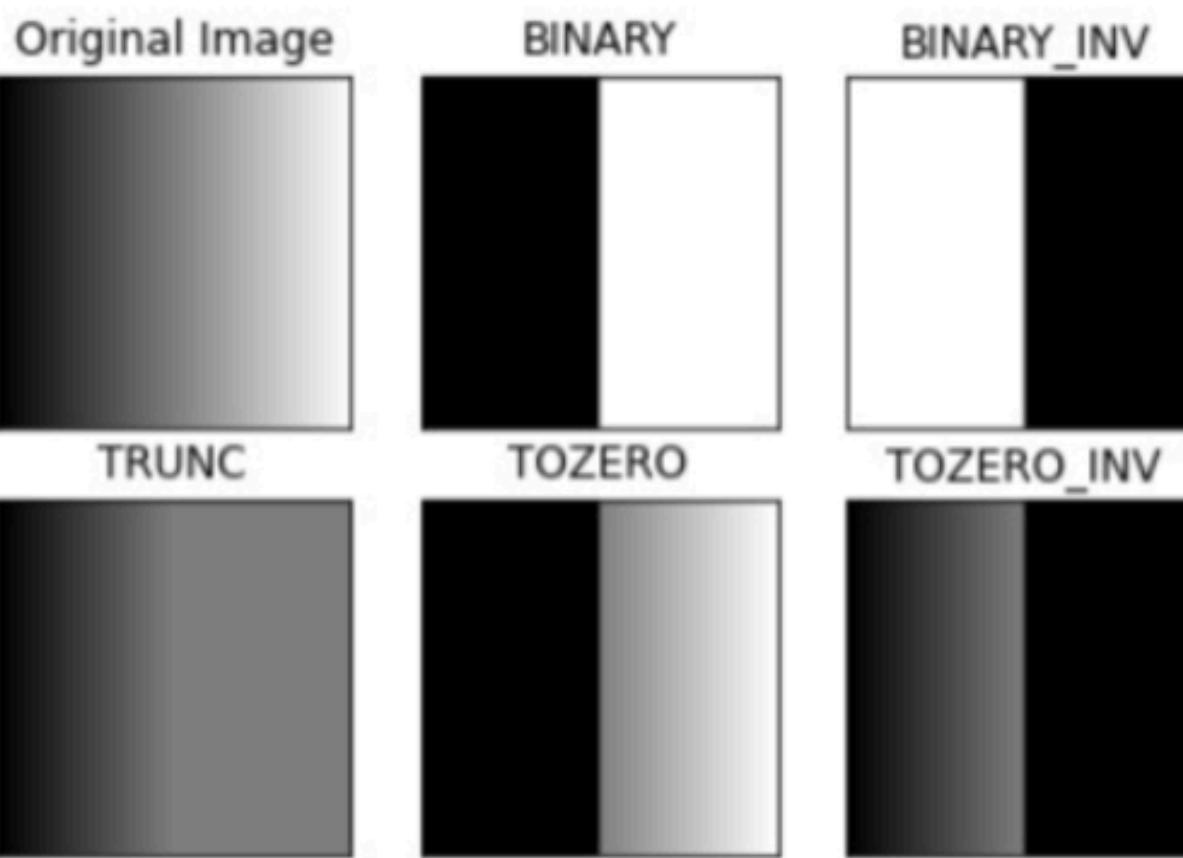
การตรวจจับส่วนที่มีความเข้มสูง (bright spots)

`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, ไม่ เช่นนั้นเป็น 0	ใช้สำหรับแยกพื้นหลังออกจากวัตถุ เช่น การแยกข้อความในภาพ
cv2.THRESH_BINARY_INV	กลับค่าของ cv2.THRESH_BINARY พิกเซลที่ค่ามากกว่า thresh จะถูกกำหนดเป็น 0 และค่าที่เหลือเป็น maxval	src(x, y) > thresh	0 ถ้าจริง, ไม่ เช่นนั้น เป็น maxval	ใช้เมื่อต้องการ วัตถุเป็นพื้นหลัง และพื้นหลังเป็น วัตถุ เช่น ข้อความ สีขาวบนพื้นดำ
cv2.THRESH_TRUNC	พิกเซลที่มีค่ามากกว่า thresh จะถูกกำหนดค่าเป็น thresh ส่วนค่าที่เหลือจะคงเดิม	src(x, y) > thresh	thresh ถ้าจริง, ไม่ เช่นนั้นเป็นค่า ดั้งเดิม	ใช้สำหรับลดค่า เกินในภาพ (clipping) เพื่อ เน้นขอบเขตค่า หนึ่ง
cv2.THRESH_TOZERO	พิกเซลที่มีค่ามากกว่า thresh จะคงเดิม ส่วนค่าที่เหลือจะถูกกำหนด เป็น 0	src(x, y) > thresh	ค่าดั้งเดิมถ้าจริง, ไม่ เช่นนั้นเป็น 0	ใช้สำหรับเปลี่ยนส่วน ที่สว่างและลับส่วน ที่บัดกรุ๊ก เช่น การหาจุดที่มีความเข้มสูง
cv2.THRESH_TOZERO_INV	กลับค่าของ cv2.THRESH_TOZERO พิกเซลที่มีค่ามากกว่า thresh จะถูกกำหนดเป็น 0 ส่วนค่าที่เหลือจะคงเดิม	src(x, y) > thresh	0 ถ้าจริง, ไม่ เช่นนั้น เป็นค่าดั้งเดิม	ใช้สำหรับลบส่วนที่ สว่างและเก็บส่วน ที่มีดีไว้ เช่น เจ้าในภาพ



```
# 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

3.3.3 Simultaneous Document and Word Cluster Discovery

The probabilistic algorithm discussed in the previous section can simultaneously discover document and word clusters. As discussed in Sect. 7.4 of Chap. 7 on high-dimensional data, this is important in the high-dimensional case because clustering characterized in terms of both rows and columns simultaneously. In the text clustering, discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the text clustering, because the factorized matrices have a natural interpretation for text data. The columns of the two factorized matrices represent word clusters and document clusters that are represented by the rows.

3.3.1 Co-clustering

-clustering is most effective for nonnegative matrices in which many entries are zero. In other words, the matrix is sparsely populated. This is the case for text clustering methods can also be generalized to dense matrices, although these techniques are relevant to the text domain. Co-clustering is also sometimes referred to as two-mode clustering because of its exploitation of both "modes" (words and documents). The co-clustering method is presented here in the context of text data. This approach is also used in the biological domain with some modifications.

matrix is the $n \times d$ document term matrix D , where rows correspond to documents, and columns correspond to words. Thus, the i th cluster is associated with a set of documents, and a set of columns V_i (words). The rows R_i are disjoint from different values of i , and the columns V_i are also disjoint from one another. Thus, the co-clustering method simultaneously leads to document clusters. From an intuitive perspective, the words representing the columns are the most relevant (or topical) words for cluster R_i . The set V_i therefore defines the set of R_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Methods discussed in this book for document clustering, such as the scatter/gather methods, and nonnegative matrix factorization (see Sect. 6.8) can find word clusters (or cluster digests) in addition to document clusters. However, different clusters are *overlapping* in these algorithms, whereas documents are overlapping in all algorithms except for the probabilistic (soft) EM clustering. In hard clustering, the word clusters and document clusters are *both* non overlapping, which means that each word is strictly associated with a particular cluster. One nice feature of co-clustering is that it explicitly explores the duality between word clusters and document clusters. Coherent word clusters can be shown to induce coherent document clusters. For example, if meaningful word clusters were already available, one could use them to cluster documents by assigning each document to the word cluster that contains the most words in common. In co-clustering, the goal is to do this simultaneously, so that word clusters and document clusters depend on each other in an optimal way.

6.3.3 Simultaneous Document and Word Cluster Discovery

A probabilistic algorithm discussed in the previous section can simultaneously document and word clusters. As discussed in Sect. 5.1 of Chap. 7 on high-dimensional matrices, this is important in the high-dimensional case because clusters are scattered in terms of both rows and columns *simultaneously*. In the text domain, the main advantage of such methods is that the topical words of a cluster provide insights about that cluster. Another example is the nonnegative matrix factorization, discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the sense that the factorized matrices have a natural interpretation for text data. This simultaneously discover word clusters and document clusters that are represented as columns of the two factorized matrices. This is also closely related to the clustering.

3.3.1 Co-clustering

clustering is most effective for nonnegative matrices in which many entries are zero. In other words, the matrix is sparsely populated. This is the case for text mining methods can also be generalized to dense matrices, although these techniques are relevant to the text domain. Co-clustering is also sometimes referred to as a two-mode clustering because of its exploitation of both "modes" (words and documents). The co-clustering method is presented here in the context of text data, but it can be applied to the numerical domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in the data matrix so that most of the nonzero entries become arranged into blocks. In the context of document clustering, the data matrix is the $n \times d$ document term matrix D , where rows correspond to documents and columns correspond to words. Thus, the i th cluster is associated with a set of rows \mathcal{R}_i (documents), and a set of columns \mathcal{V}_i (words). The rows \mathcal{R}_i are disjoint from different values of i , and the columns \mathcal{V}_i are also disjoint from one another (as is i). Thus, the co-clustering method simultaneously leads to document clusters. From an intuitive perspective, the words representing the columns are the most relevant (or topical) words for cluster \mathcal{R}_i . The set \mathcal{V}_i therefore defines the columns of \mathcal{R}_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Methods discussed in this book for document clustering, such as the scatter/gather method and non-negative matrix factorization (see Sect. 6.8 of word clustering) can also be applied to words in addition to document clusters. However, there are some important differences between word clustering and document clustering in these algorithms, whereas document clustering is a well-studied problem except for the probabilistic (soft) EM algorithm. In contrast, word clustering and document clusters are both non overlapping sets, and each document is associated with a particular cluster. One nice way to approach word clustering is to explore the duality between word clustering and document clustering. It can be shown to induce coherent document clusters if meaningful word clusters were already available. If meaningful word clusters were already available, one can simply assign each document to the word cluster it belongs to. In co-clustering, the goal is to do the opposite: to find both document and word clusters depend on each other in an iterative fashion.

3.3 Simultaneous Document and Word Cluster Discovery
 A probabilistic algorithm described in the previous section can simultaneously document and word clusters. As discussed in Sect. 3.1 of Chap. 7 on high-dimensional methods, this is important in the high-dimensional case because clustering is characterized in terms of both rows and columns *simultaneously*. In the text domain advantage of such methods is that the topical words of a cluster provide insights about that cluster. Another example is the nonnegative matrix factorization discussed in Sect. 1.8 of Chap. 1. This approach is very popular in the sense the factorized matrices have a natural interpretation for text data. This simultaneously discover word clusters and document clusters that are represented by the columns of the two factorized matrices. This is also closely related to the clustering.

3.3.1 Co-clustering

clustering is most effective for nonnegative matrices in which many entries are zero. In other words, the matrix is sparsely populated. This is the case for text mining methods can also be generalized to dense matrices, although these techniques are relevant to the text domain. Co-clustering is also sometimes referred to as a *bi-mode clustering* because of its exploitation of both "modes" (words and documents) in the biological domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in the data matrix so that most of the nonzero entries become arranged into blocks. In the context of document clustering, the data matrix is the $n \times d$ document term matrix D , where rows correspond to documents, and columns correspond to words. Thus, the i th cluster is associated with a set of documents, and a set of columns V_i (words). The rows R_i are disjoint from different values of i , and the columns V_i are also disjoint from one another (in terms of i). Thus, the co-clustering method simultaneously leads to document clusters. From an intuitive perspective, the words representing the columns V_i are the most relevant (or topical) words for cluster R_i . The set V_i therefore defines the topicality of R_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. As discussed in this book, document clustering, such as the scatter/gather methods or non-negative matrix factorization (see Sect. 6.8 of this chapter), can find word clusters in addition to document clusters. However, the way these algorithms work is very different. In these algorithms, whereas documents are assigned to clusters except for the probabilistic (soft) EM algorithm, words are not assigned to clusters. Document clusters are both non-overlapping and non-exclusive, meaning that a document can be associated with a particular cluster. One nice feature of these algorithms is that they explore the duality between word clusters and document clusters. It can be shown to induce coherent document clusters if meaningful word clusters were already available, for example by assigning each document to the words it contains. In co-clustering, the goal is to do the opposite: meaningful word clusters depend on each other in some way.

3.3 Simultaneous Document and Word Cluster Extraction
 Probabilistic algorithms discussed in the previous sections can simultaneously document and word clusters. As discussed in Sect. 3.1 and Chap. 7 on high-dimensional methods this is important since both documents and words clusters are represented in terms of both rows and columns, respectively. Another significant advantage of such methods is that the typical words of a cluster provide insight about that cluster. Another example is the nonnegative matrix factorization discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the sense the factored matrices have a natural interpretation for text data. This simultaneously discovers word clusters and document clusters that are represented by the two factorized matrices. This is also closely related to the clustering

3.3.1 Co-clustering

Clustering is most effective for nonnegative matrices in which many values are zero. In other words, the matrix is sparsely populated. This is the case for topic modeling methods can also be generalized to dense matrices, although these techniques are not as relevant to the text domain. Co-clustering is also sometimes referred to as *co-matrix clustering* because of its exploitation of both "modes" (words and topics). The co-clustering method is presented here in the context of text data, but it can be used in the biological domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in the data matrix so that most of the nonzero entries become arranged into blocks. In the context of document clustering, the data matrix is the $n \times d$ document term matrix D , where rows correspond to documents and columns correspond to words. Thus, the i th cluster is associated with a set of documents \mathcal{R}_i (rows), and a set of columns V_i (words). The rows \mathcal{R}_i are disjoint from rows with different values of i , and the columns V_i are also disjoint from one another (unless $i = j$). Thus, the co-clustering method simultaneously leads to document clusters and word clusters. From an intuitive perspective, the words representing the columns V_i are the most relevant (or topical) words for cluster \mathcal{R}_i . The set V_i therefore defines the topicality of \mathcal{R}_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Many methods discussed in this book for document clustering, such as the scatter/gather method, probabilistic methods, and nonnegative matrix factorization (see Sect. 6.8 of this chapter), find word clusters (or cluster digests) in addition to document clusters. However, the different clusters are *overlapping* in these algorithms, whereas document clusters are *nonoverlapping* in all algorithms except for the probabilistic (soft) EM clustering. In hard clustering, the word clusters and document clusters are *both* non overlapping, which means that each word is strictly associated with a particular cluster. One nice feature of co-clustering is that it explicitly explores the duality between word clusters and document clusters. Coherent word clusters can be shown to induce coherent document clusters. For example, if meaningful word clusters were already available, one could use them to cluster documents by assigning each document to the word cluster with the most words in common. In co-clustering, the goal is to do this such that the word clusters and document clusters depend on each other in an optimal way.

438 13.3.3 Simultaneous Document and Word Cluster Discovery

13.3.3 Simultaneous Document Clustering

The probabilistic algorithm discussed in the previous section can simultaneously discover document and word clusters. As discussed in Sect. 7.4 of Chap. 7 on high-dimensional clustering methods, this is important in the high-dimensional case because clusters are best characterized in terms of both rows and columns simultaneously. In the text domain, the additional advantage of such methods is that the topical words of a cluster provide semantic insights about that cluster. Another example is the nonnegative matrix factorization method, discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the text domain because the factorized matrices have a natural interpretation for text data. This approach can simultaneously discover word clusters and document clusters that are represented by the columns of the two factorized matrices. This is also closely related to the concept of co-clustering.

13.3.3.1 Co-clustering

Co-clustering is most effective for nonnegative matrices in which many entries have zero values. In other words, the matrix is sparsely populated. This is the case for text data. Co-clustering methods can also be generalized to dense matrices, although these techniques are not relevant to the text domain. Co-clustering is also sometimes referred to as bi-clustering or two-mode clustering because of its exploitation of both "modes" (words and documents). While the co-clustering method is presented here in the context of text data, the broader approach is also used in the biological domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in the data matrix \mathbf{X} so that most of the nonzero entries become arranged into blocks. In the context of text data, this matrix is the $n \times d$ document term matrix D , where rows correspond to documents and columns correspond to words. Thus, the i th cluster is associated with a set of rows \mathcal{R}_i (documents), and a set of columns V_i (words). The rows \mathcal{R}_i are disjoint from one another over different values of i , and the columns V_i are also disjoint from one another over different values of i . Thus, the co-clustering method simultaneously leads to document clusters and word clusters. From an intuitive perspective, the words representing the columns of V_i are the most relevant (or topical) words for cluster \mathcal{R}_i . The set V_i therefore defines a digest of \mathcal{R}_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Most of the methods discussed in this book for document clustering, such as the scatter/gather method, probabilistic methods, and nonnegative matrix factorization (see Sect. 6.8 of Chap. 6), produce word clusters (or cluster digests) in addition to document clusters. However, the word clusters in the different clusters are *overlapping* in these algorithms, whereas document clusters are non overlapping in all algorithms except for the probabilistic (soft) EM method. In co-clustering, the word clusters and document clusters are *both* non overlapping. Each document and word is strictly associated with a particular cluster. One nice characteristic of co-clustering is that it explicitly explores the duality between word clusters and document clusters. Coherent word clusters can be shown to induce coherent document clusters and vice versa. For example, if meaningful word clusters were already available, then one could be able to cluster documents by assigning each document to the word cluster with which it has the most words in common. In co-clustering, the goal is to do this simultaneously so that word clusters and document clusters depend on each other in an optimal way.

Document and Word Cluster Discovery

Simultaneous inference as discussed in the previous sections can be conveniently done by applying the methods of Sect. 7.1 of Chap. 7 on high-dimensional

probabilistic algorithm discussed in the previous section can simultaneously discover document and word clusters. As discussed in Sect. 7.4 of Chap. 7 on high-dimensional clustering methods, this is important in the high-dimensional case because clusters are best represented in terms of both rows and columns simultaneously. In the text domain, the main advantage of such methods is that the topical words of a cluster provide some insights about that cluster. Another example is the nonnegative matrix factorization, discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the text domain as the factorized matrices have a natural interpretation for text data. This approach simultaneously discovers word clusters and document clusters that are represented by the columns of the two factorized matrices. This is also closely related to the concept of topic modeling.

Co-clustering

Co-clustering is most effective for nonnegative matrices in which many entries have zero value. In other words, the matrix is sparsely populated. This is the case for text data, where the terms in the document matrix correspond to the text domain. Co-clustering is also sometimes referred to as bi-clustering or two-mode clustering because of its exploitation of both "modes" (words and documents). A co-clustering method is presented here in the context of text data, the basic idea being to rearrange the rows and columns in the data matrix.

Idea in co-clustering is to rearrange the rows and columns in the data matrix such that the nonzero entries become arranged into blocks. In the context of text documents, the data matrix is the $n \times d$ document term matrix D , where rows correspond to documents and columns correspond to words. Thus, the i th cluster is associated with a set of rows \mathcal{R}_i (documents), and a set of columns \mathcal{V}_i (words). The rows \mathcal{R}_i are disjoint from one another over different values of i , and the columns \mathcal{V}_i are also disjoint from one another over different values of i . Thus, the co-clustering method simultaneously leads to document clusters and word clusters. From an intuitive perspective, the words representing the columns of \mathcal{V}_i are relevant (or topical) words for cluster \mathcal{R}_i . The set \mathcal{V}_i therefore defines a topical representation for cluster \mathcal{R}_i .

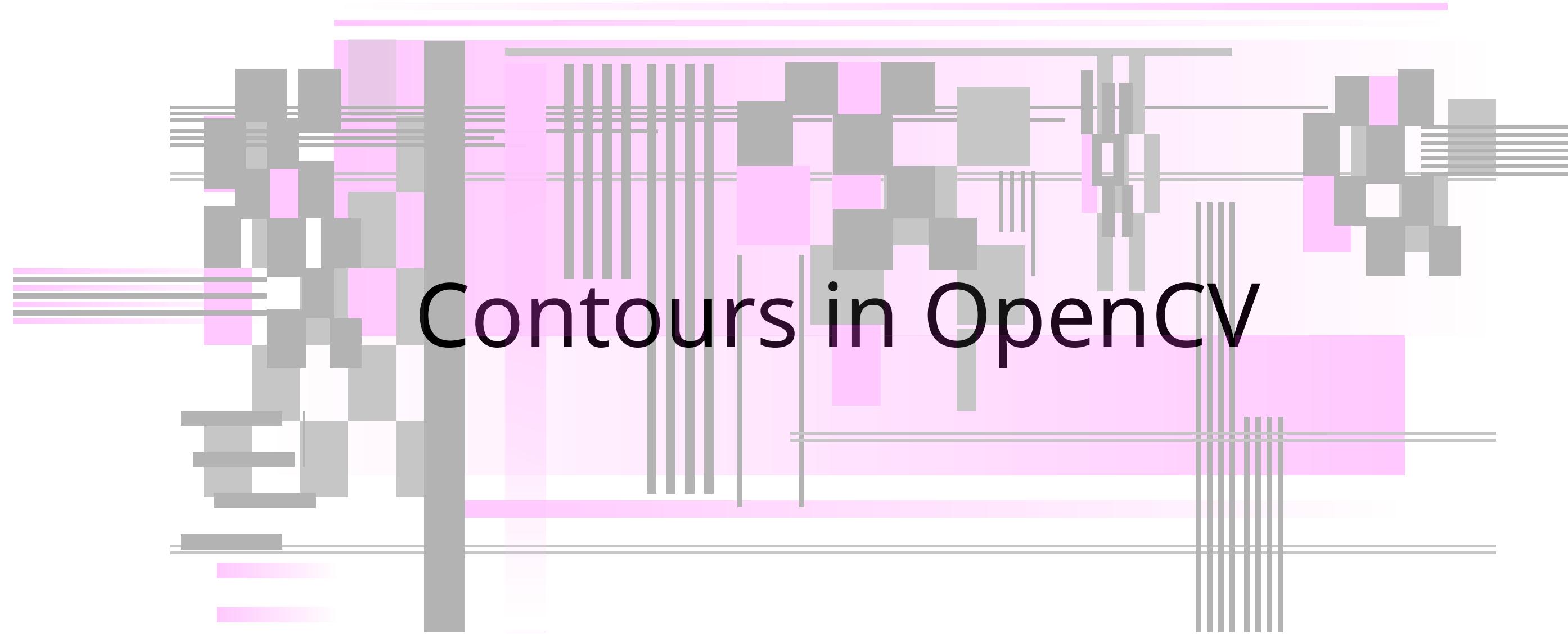
In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Most of the methods discussed in this book for document clustering, such as the scatter/gather method, probabilistic methods, and nonnegative matrix factorization (see Sect. 6.8 of Chap. 6), produce word clusters (or cluster digests) in addition to document clusters. However, the words in the different clusters are *overlapping* in these algorithms, whereas document clusters are non overlapping in all algorithms except for the probabilistic (soft) EM method. In a co-clustering, the word clusters and document clusters are *both* non overlapping. Each document and word is strictly associated with a particular cluster. One nice characteristic of co-clustering is that it explicitly explores the duality between word clusters and document clusters. Coherent word clusters can be shown to induce coherent document clusters and vice versa. For example, if meaningful word clusters were already available, then one may be able to cluster documents by assigning each document to the word cluster with which it has the most words in common. In co-clustering, the goal is to do this simultaneously so that word clusters and document clusters depend on each other in an optimal way.

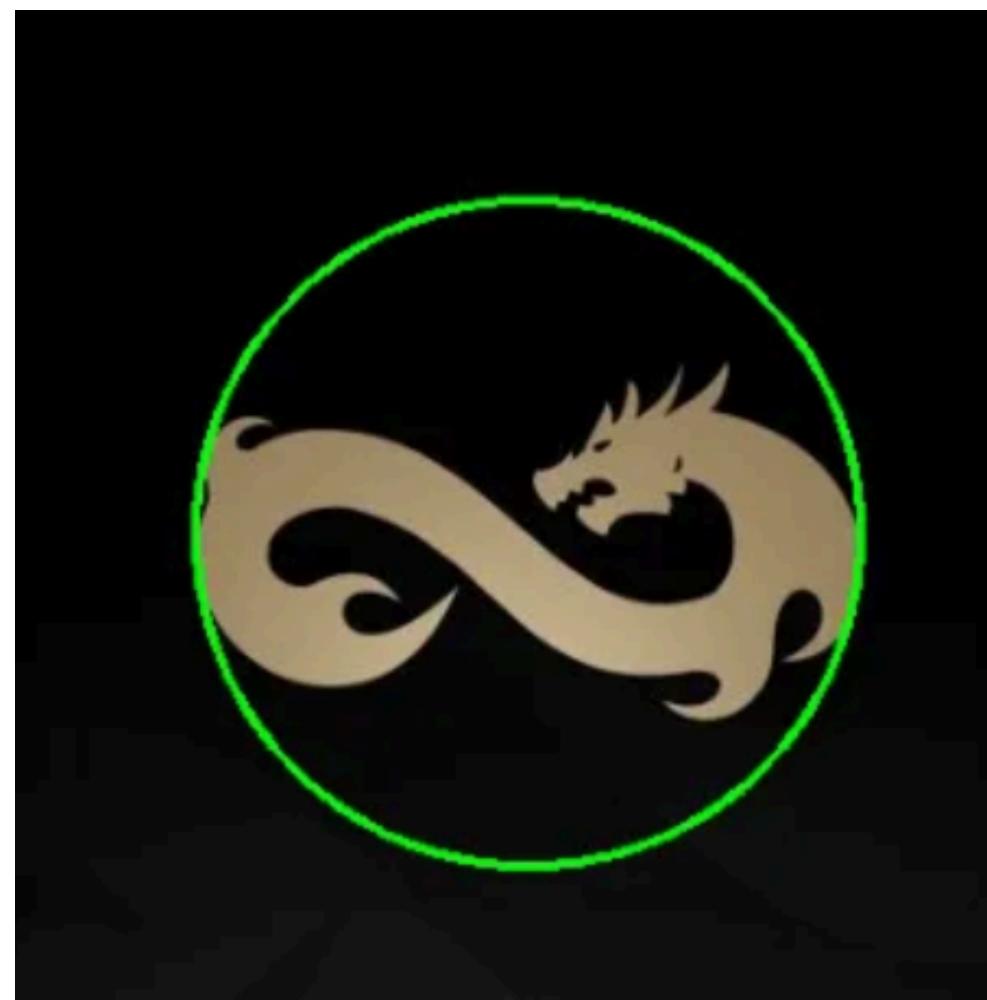
3.3.1 Co-clustering

Co-clustering is most effective for nonnegative matrices in which many entries have zero values. In other words, the matrix is sparsely populated. This is the case for text data. Co-clustering methods can also be generalized to dense matrices, although these techniques are relevant to the text domain. Co-clustering is also sometimes referred to as bi-clustering or two-mode clustering because of its exploitation of both "modes" (words and documents). While the co-clustering method is presented here in the context of text data, the basic approach is also used in the biological domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in our data such that most of the nonzero entries become arranged into blocks. In the context of text mining, the matrix is the $n \times d$ document term matrix D , where rows correspond to documents and columns correspond to words. Thus, the i th cluster is associated with a set of rows \mathcal{R}_i (documents), and a set of columns \mathcal{V}_i (words). The rows \mathcal{R}_i are disjoint from one another over different values of i , and the columns \mathcal{V}_i are also disjoint from one another over different values of i . Thus, the co-clustering method simultaneously leads to document clusters and word clusters. From an intuitive perspective, the words representing the columns of \mathcal{V}_i are the most relevant (or topical) words for cluster \mathcal{R}_i . The set \mathcal{V}_i therefore defines the extent of \mathcal{R}_i .

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. In fact, many methods discussed in this book (e.g., document clustering, such as the scatter/gather algorithm) can be extended to word clustering (or cluster co-clustering) in addition to document clusters. However, the two types of clusters are *overlapping*. In these algorithms, whereas document clustering is overlapping in all algorithms except for the probabilistic (soft) EM method, word clustering and document clustering are both non-overlapping. Each word is strictly associated with a particular cluster. One plus of this type of clustering is that it explicitly explores the duality between word clustering and document clustering. Coherent word clusters can be shown to induce coherent document clusters, and vice versa. For example, if meaningful word clusters were already available, then it would be possible to cluster documents by assigning each document to the word cluster with the most words in common. In co-clustering, the goal is to define simultaneously word clusters and document clusters based on a co-relationship or co-occurrence. This is done by defining a joint probability distribution over word clusters and document clusters.





พารามิเตอร์

1. `image` (ภาพ):

- ภาพที่ใช้ต้องอยู่ในรูปแบบใบหน้า เช่น ภาพที่ผ่านการ Thresholding (`cv2.threshold`) หรือ Edge Detection (`cv2.Canny`) และ
- ตัวอย่างการแปลงภาพให้อยู่ในรูปแบบใบหน้า:

```
python คัดลอกโค้ด
_, binary_image = cv2.threshold(grayscale_image, 127, 255, cv2.THRESH_BINARY)
```

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

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

Steps to Find Contours with OpenCV

1. Load the Image

- Read the image using OpenCV.
- Example:

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

- Example using edge detection:

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

4. Find Contours

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

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

5. Draw Contours

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

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

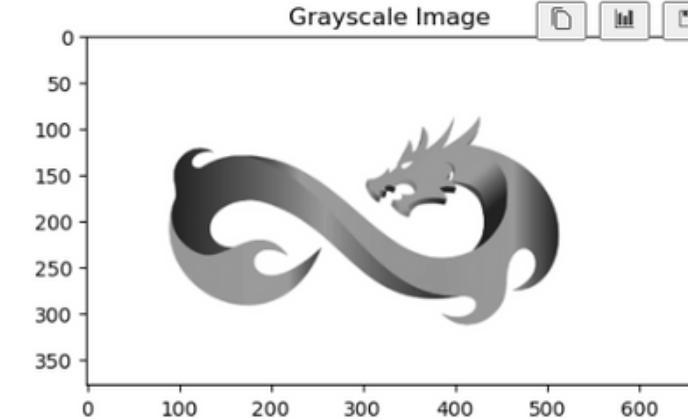
Summary

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.

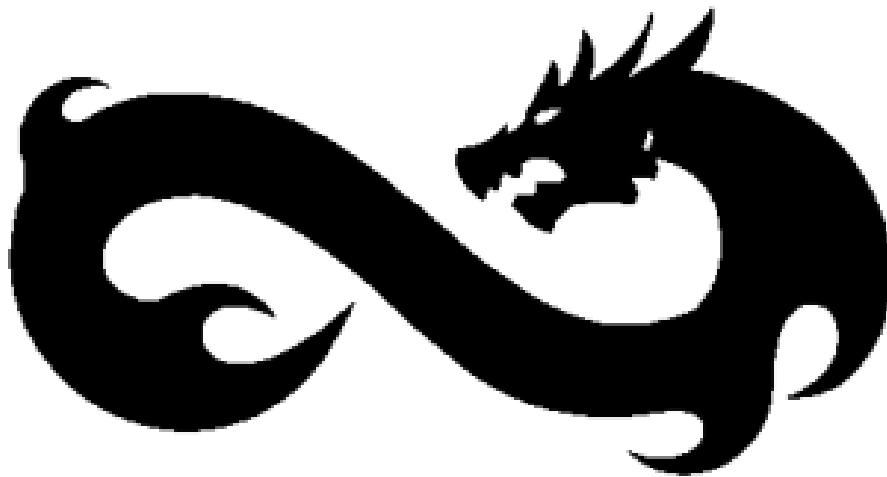
```
# Load the image
image = cv2.imread('./images/dragon.png', cv2.IMREAD_COLOR)

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

Original Image



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



```
# 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

```
# 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()
```

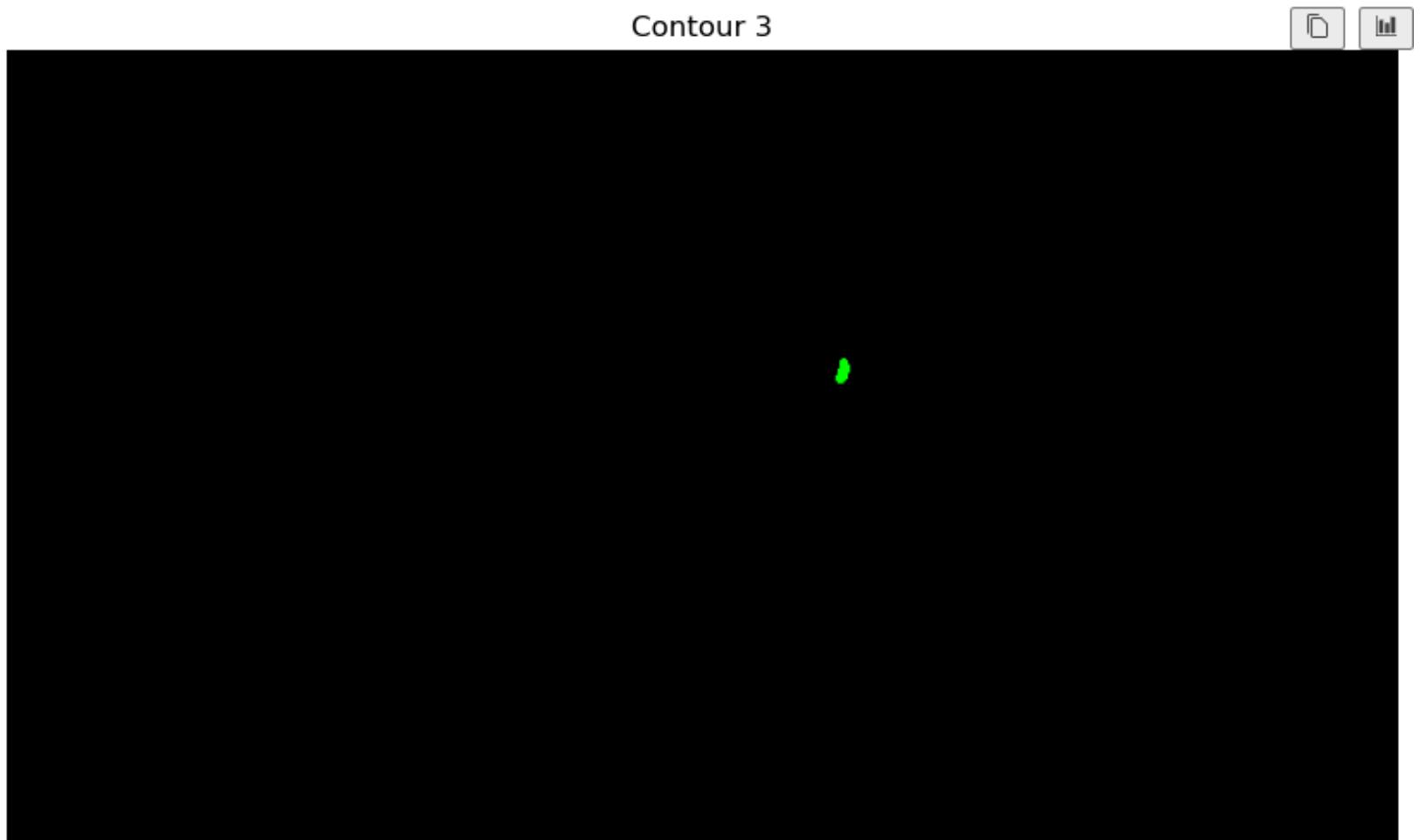
Contour 1



Contour 2



Contour 3



Contour 4



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

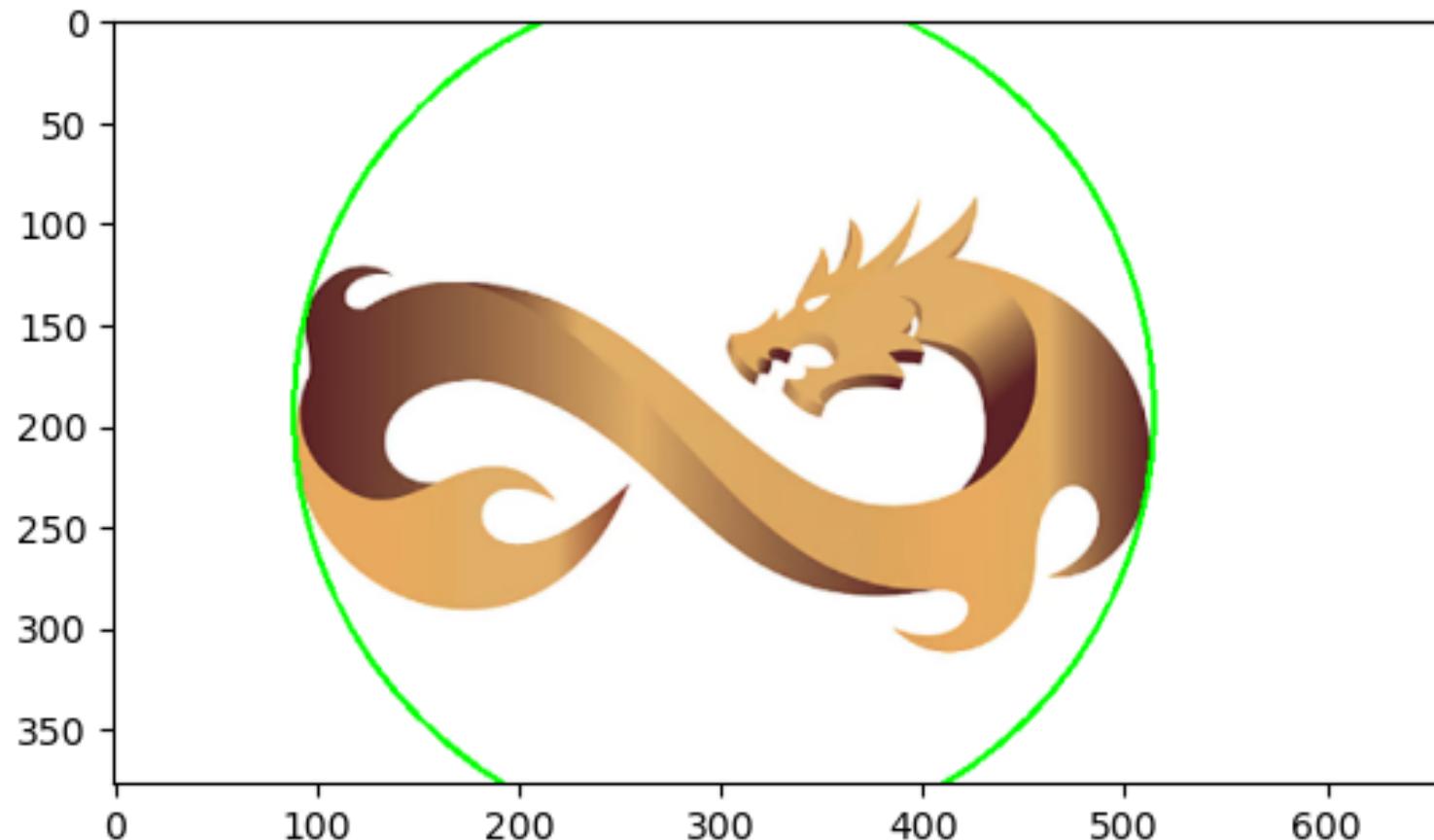
```
contours,hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

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)
```



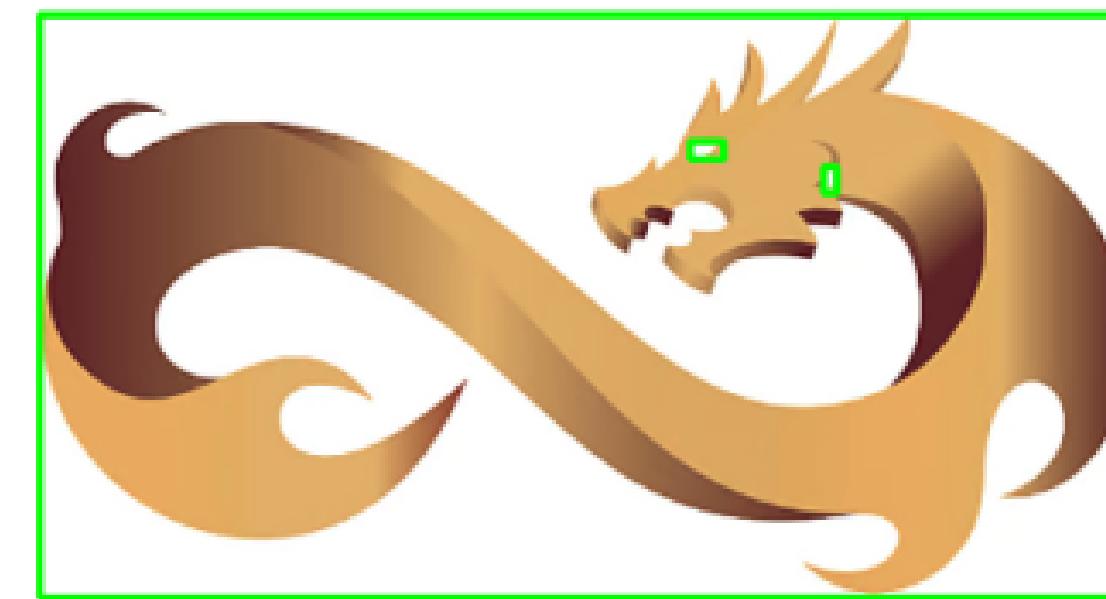
```
contours,hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

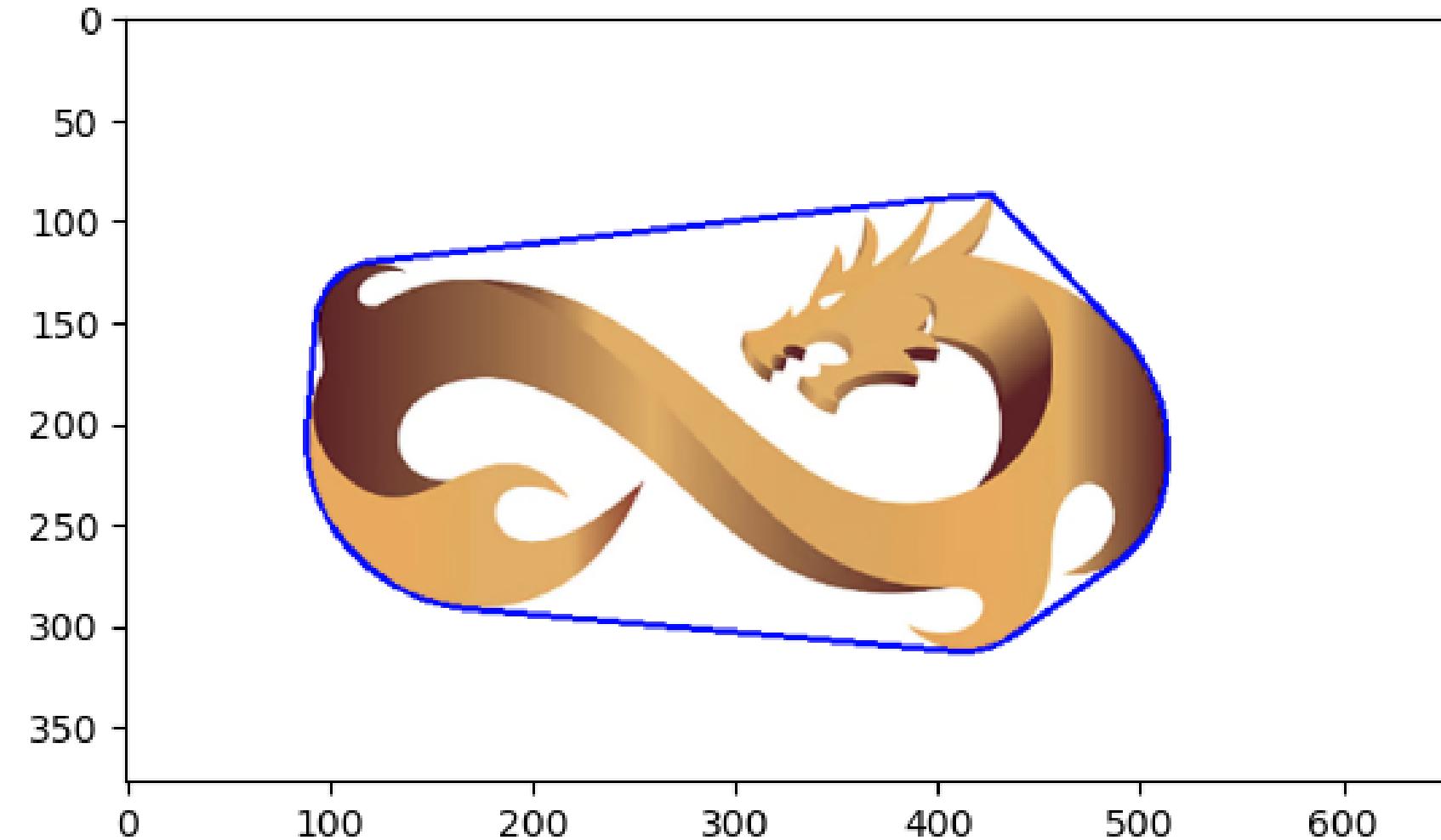
# Get the first contour from the list
contour = contours[1]

# 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)
```





```
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

