Department of Computer Science and Engineering (Data Science)
Image Processing and Computer Vision I (DJ19DSL603)

Experiment 7

Aim: To perform morphological operations on image (erosion, dilation, opening, closing and hit and miss transform)

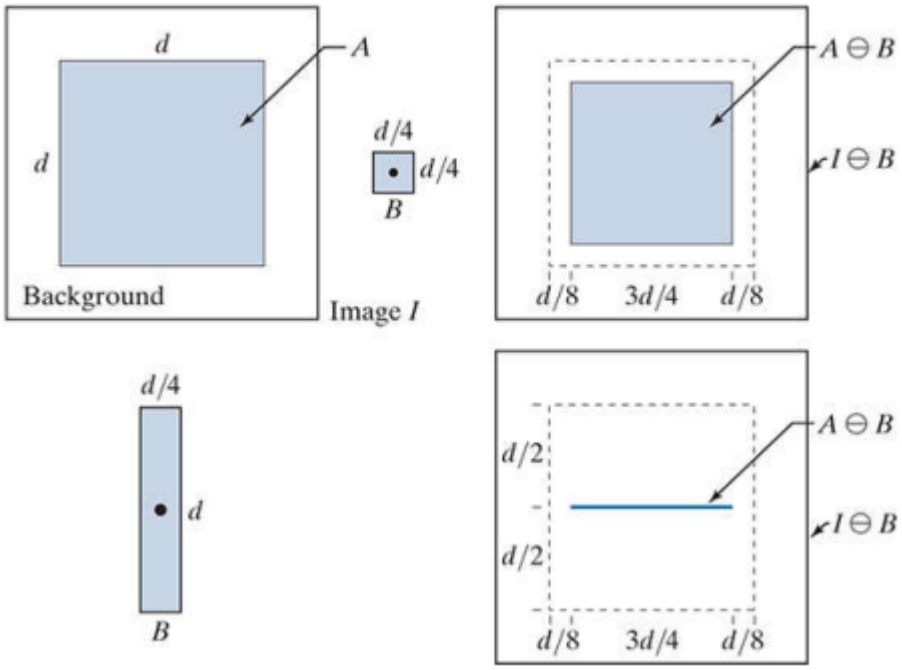
Theory:

1. **Erosion:**

Morphological expressions are written in terms of structuring elements and a set, A, of foreground pixels, or in terms of structuring elements and an image, I, that contains A. We consider the former approach first. Erosion of A by B is defined as:

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

In words, this equation indicates that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A. (Remember, displacement is defined with respect to the origin of B. The result of erosion is controlled by the shape of the structuring element. The image is eroded by two different structuring elements (B) giving the outputs as seen:

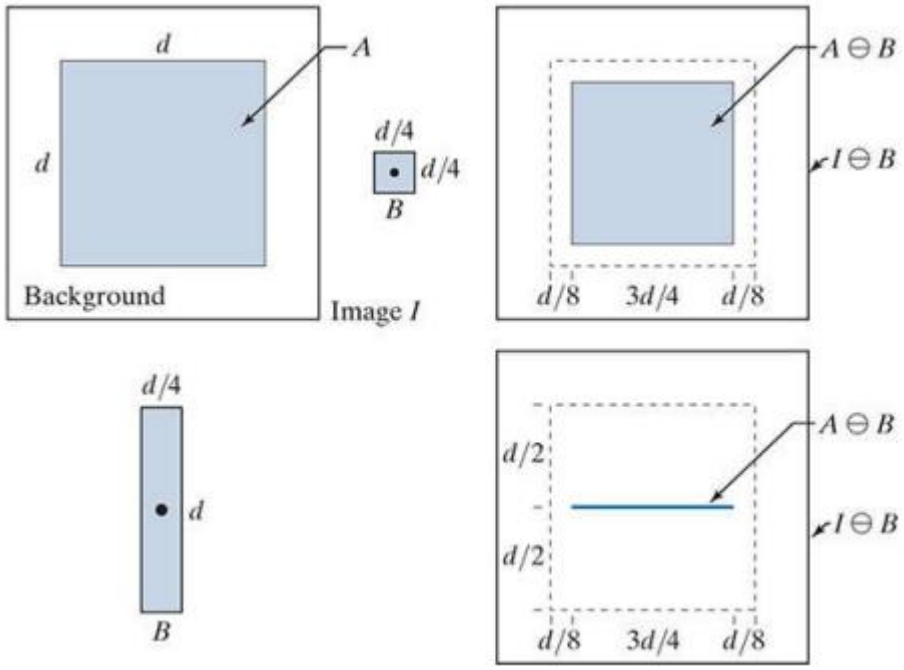


2. **Dilation:**

Dilation of A by B is defined as:

$$A \oplus B = \left\{ z \middle| [(\hat{B})_z \cap A] \subseteq A \right\}$$

The dilation of A by B then is the set of all displacements, z, such that the foreground elements of overlap at least one element of A. (Remember, z is the displacement of the origin of $\hat{B}$. The image is dilated by two different structuring elements (B) giving the outputs as seen:

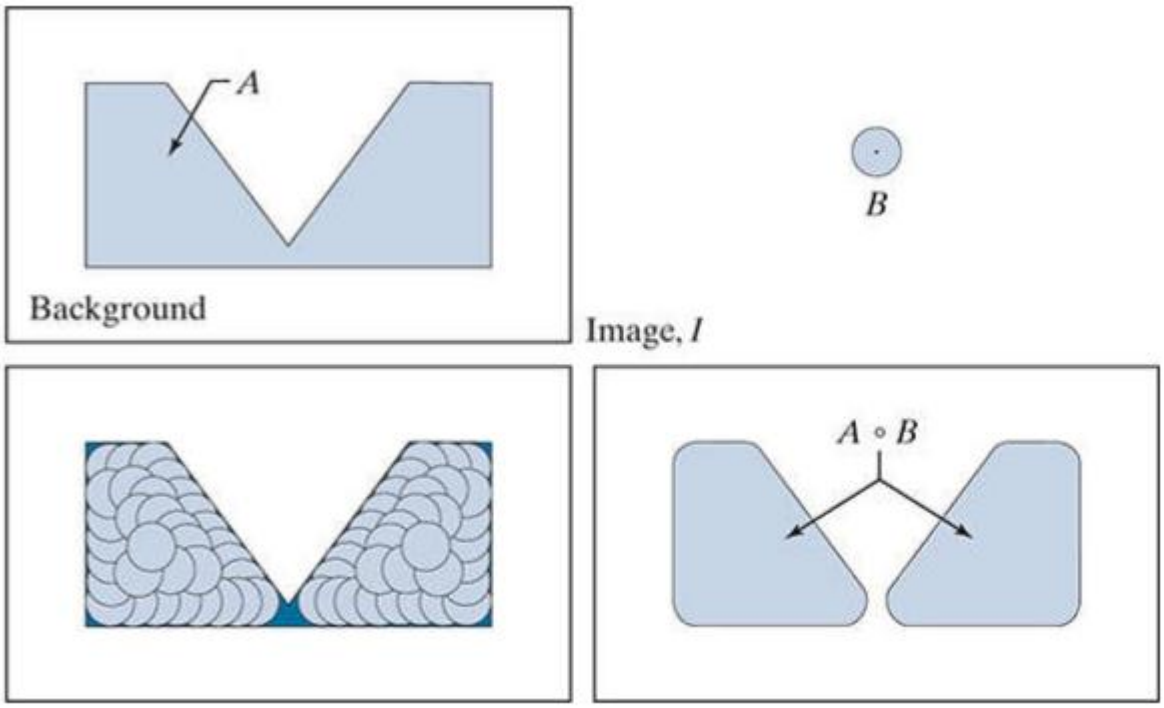


3. Opening:

Opening generally smoothens the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions. The opening of set A by structuring element B is defined as:

$$A \circ B = (A \ominus B) \oplus B$$

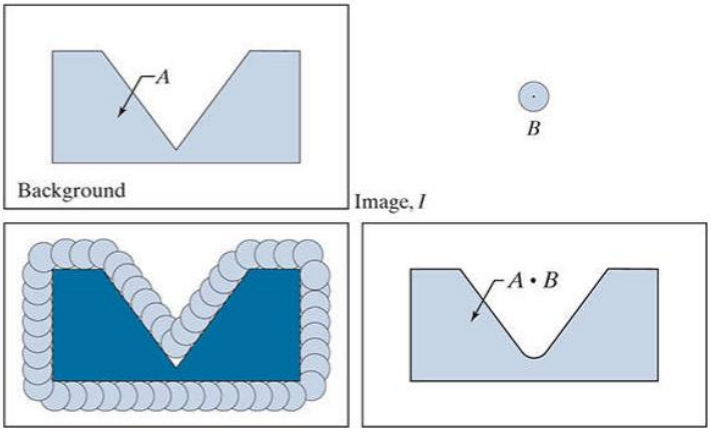
Thus, the opening A by B is the erosion of A by B, followed by a dilation of the result by B. The opening of A by B is the union of all the translations of B so that B fits entirely in A



4. **Closing:**
   The closing of set A by structuring element B is defined as:

   $A \bullet B = (A \oplus B) \ominus B$

   The closing of A by B is simply the dilation of A by B, followed by erosion of the result by B. The opening of A by B is the union of all the translations of B so that B fits entirely in A.



5. **Hit and miss transform:**
   The morphological hit-or-miss transform (HMT) is a basic tool for shape detection. Let I be a binary image composed of foreground (A) and background

pixels, respectively. Unlike the morphological methods discussed thus far, the HMT utilizes two structuring elements: B_1 for detecting shapes in the foreground, and B_2 for detecting shapes in the background. The HMT of image I is defined as

$$I \circledast B_{1,2} = \{z \mid (B_1)_z \subseteq A \text{ and } (B_2)_z \subseteq A^c\}$$
$$= (A \ominus B_1) \cap (A^c \ominus B_2)$$

In words, this equation says that the morphological HMT is the set of translations, z, of structuring elements B_1 and B_2 such that, simultaneously, B_1 found a match in the foreground (i.e., is contained in A) and B_2 found a match in the background (i.e., is contained in A^c).

Lab Assignments to complete in this session

Problem Statement: Develop a Python program utilizing the OpenCV library to morph the images in spatial domain using the morphological operations explained. The program should address the following tasks:

1. Read any image from COVID 19 Image Dataset. Take it as object **Dataset Link:** **Covid-19 Image Dataset**
2. Create a structuring element of any form that is much smaller in size than the image considered in step 1.
3. Display the before & after image(s) used in every task below:
    a. Apply erosion and show the before and after image
    b. Apply dilation and show the before and after image
    c. Apply opening and show the before and after image
    d. Apply closing and show the before and after image
    e. Apply hit and miss transformation and show the before and after image. Take the structuring element to be negation of the one used till now.

The solution to the operations performed must be produced by scratch coding without the use of built in OpenCV methods.

60009220136

Riya Chavan

D2-1

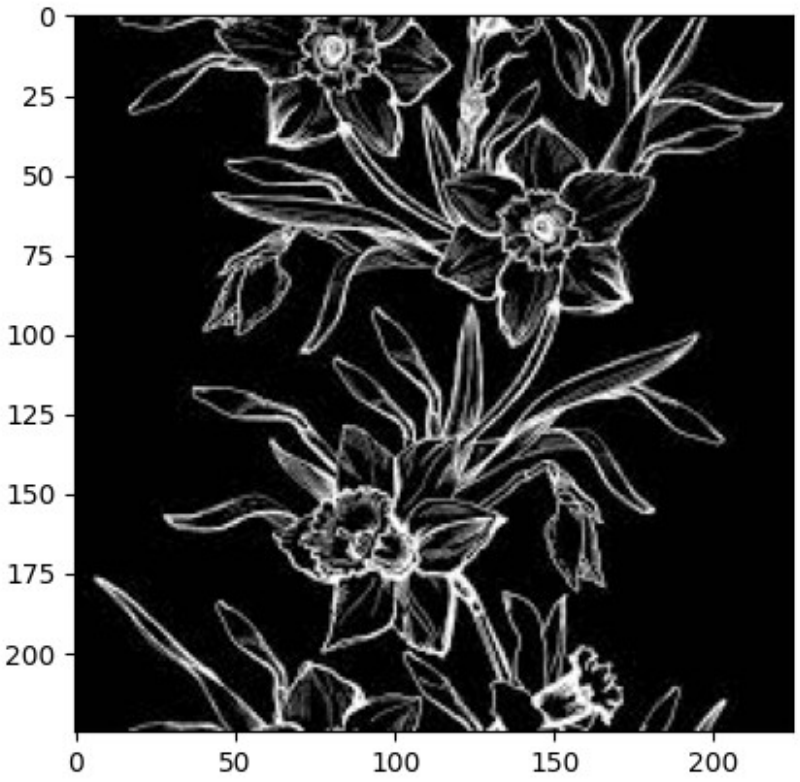
Importing Libraries

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

Image

```
image = cv2.imread('try.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1168a947450>
```

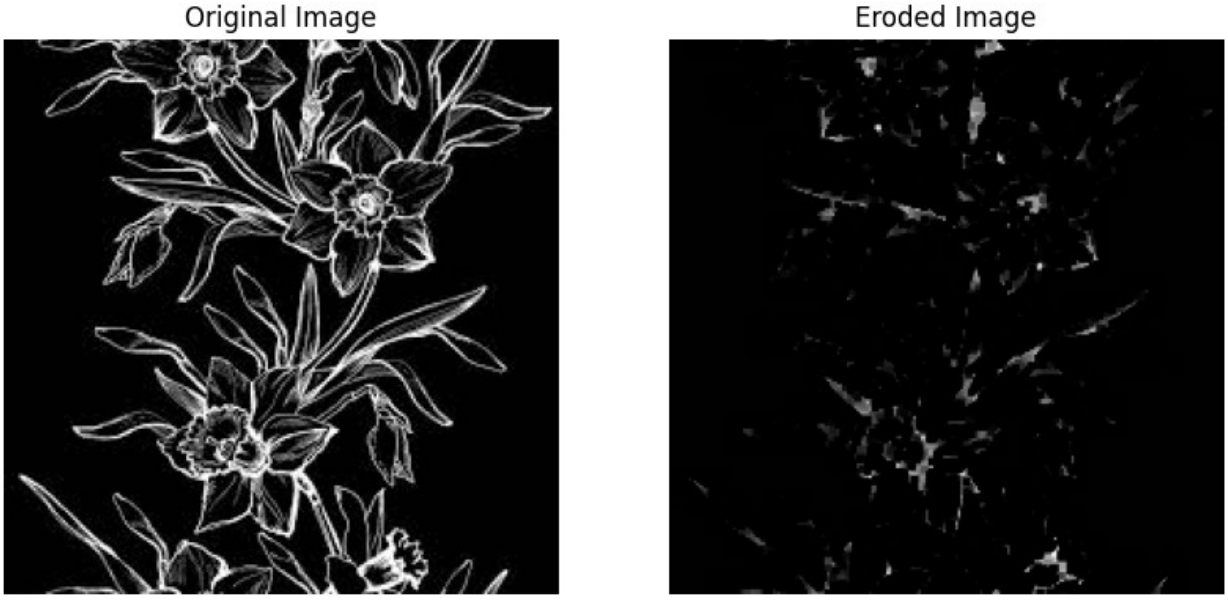


Erosion

```
def erosion(image, se):
    image_height, image_width = image.shape
    se_height, se_width = se.shape
```

```
    se_center_y = se_height // 2
    se_center_x = se_width // 2
    output_image = np.full_like(image, 255)

    for y in range(image_height):
        for x in range(image_width):
            local_min = 255
            for j in range(se_height):
                for i in range(se_width):
                    if se[j, i] == 1:
                        image_y = y + (j - se_center_y)
                        image_x = x + (i - se_center_x)
                        if 0 <= image_y < image_height and 0 <=
image_x < image_width:
                            local_min = min(local_min, image[image_y,
image_x])
            output_image[y, x] = local_min

    return output_image

def display_images(before, after, title_before, title_after):
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title(title_before)
    plt.imshow(before, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title(title_after)
    plt.imshow(after, cmap='gray')
    plt.axis('off')

    plt.show()

if image is None:
     print("Error: Image not found.")
else:
       # Example structuring element (3x3 square)
    se = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], dtype=np.uint8)

        # Negation of the structuring element
    se_neg = np.array([[0, 0, 0],
                       [0, 0, 0],
                       [0, 0, 0]], dtype=np.uint8)

        # Apply erosion
    eroded_image = erosion(image, se)
```

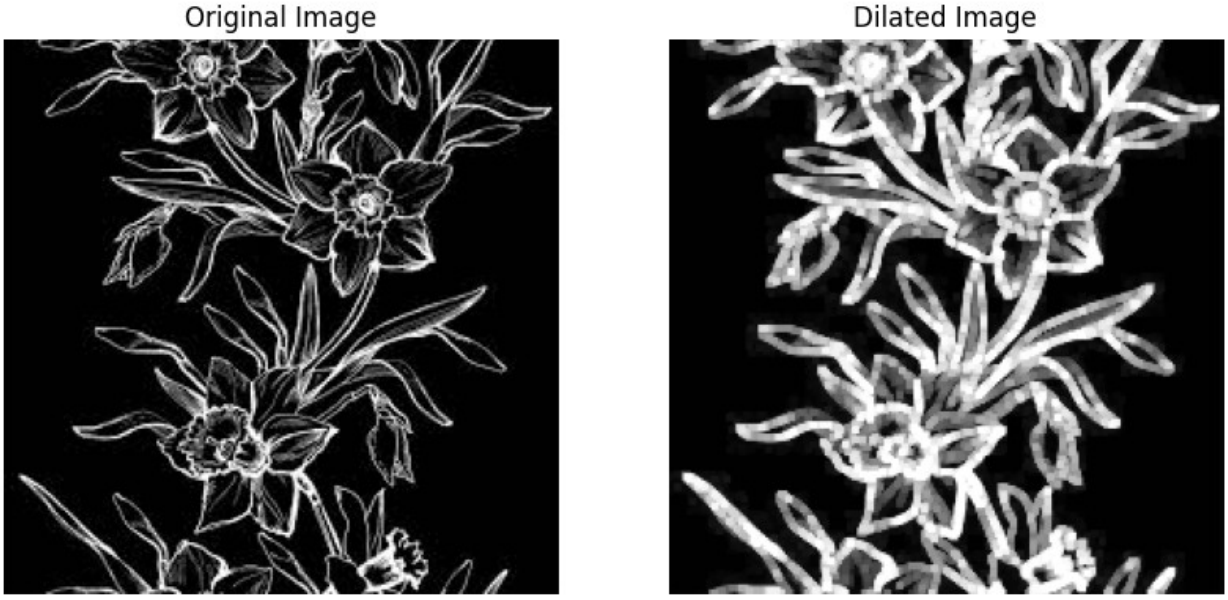
```
    display_images(image, eroded_image, 'Original Image', 'Eroded
Image')
```



Dilation

```
def dilation(image, se):
    image_height, image_width = image.shape
    se_height, se_width = se.shape
    se_center_y = se_height // 2
    se_center_x = se_width // 2
    output_image = np.full_like(image, 0)

    for y in range(image_height):
        for x in range(image_width):
            local_max = 0
            for j in range(se_height):
                for i in range(se_width):
                    if se[j, i] == 1:
                        image_y = y + (j - se_center_y)
                        image_x = x + (i - se_center_x)
                        if 0 <= image_y < image_height and 0 <=
image_x < image_width:
                            local_max = max(local_max, image[image_y,
image_x])
            output_image[y, x] = local_max

    return output_image

    # Apply dilation
dilated_image = dilation(image, se)
```

```
display_images(image, dilated_image, 'Original Image', 'Dilated Image')
```



Opening and Closing

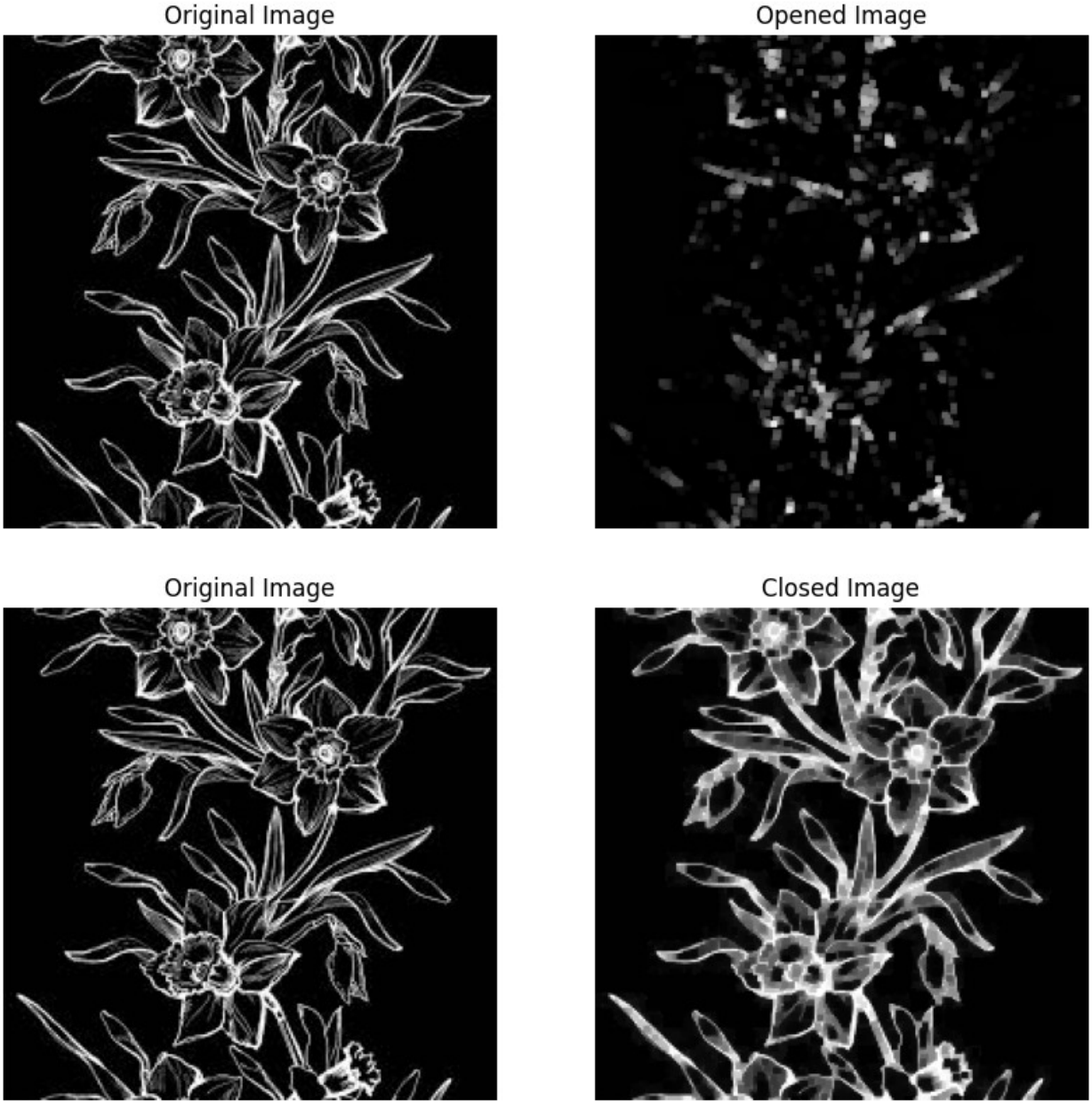
```
def opening(image, se):
    eroded_image = erosion(image, se)
    opened_image = dilation(eroded_image, se)
    return opened_image

def closing(image, se):
    dilated_image = dilation(image, se)
    closed_image = erosion(dilated_image, se)
    return closed_image
```

Hit and Miss Transformation

```
def hit_and_miss(image, element):
    negated_element = 1 - np.array(element)
    eroded_image = erode(image, np.array(element))
    inverted_image = 255 - image
    negated_eroded_image = erode(inverted_image, negated_element)
    hit_and_miss_image = np.bitwise_and(eroded_image,
negated_eroded_image)
    return hit_and_miss_image

def hit_and_miss(image, se, se_neg):
    image_height, image_width = image.shape
    se_height, se_width = se.shape
    se_center_y = se_height // 2
    se_center_x = se_width // 2
```

```
    output_image = np.zeros_like(image)

    for y in range(image_height):
        for x in range(image_width):
            match = True
            for j in range(se_height):
                for i in range(se_width):
                    if se[j, i] == 1:
                        image_y = y + (j - se_center_y)
                        image_x = x + (i - se_center_x)
                        if 0 <= image_y < image_height and 0 <=
image_x < image_width:
                            if image[image_y, image_x] == 0:
                                match = False
                    elif se_neg[j, i] == 1:
                        image_y = y + (j - se_center_y)
                        image_x = x + (i - se_center_x)
                        if 0 <= image_y < image_height and 0 <=
image_x < image_width:
                            if image[image_y, image_x] != 0:
                                match = False
            if match:
                output_image[y, x] = 255

    return output_image

if image is None:
     print("Error: Image not found.")
else:
       # Example structuring element (3x3 square)
    se = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], dtype=np.uint8)

        # Negation of the structuring element
    se_neg = np.array([[0, 0, 0],
                       [0, 0, 0],
                       [0, 0, 0]], dtype=np.uint8)



opened_image = opening(image, se)
display_images(image, opened_image, 'Original Image', 'Opened Image')

        # Apply closing
closed_image = closing(image, se)
display_images(image, closed_image, 'Original Image', 'Closed Image')

        # Apply hit and miss transformation
hit_and_miss_image = hit_and_miss(image, se, se_neg)
```

```
display_images(image, hit_and_miss_image, 'Original Image', 'Hit and Miss Image')
```

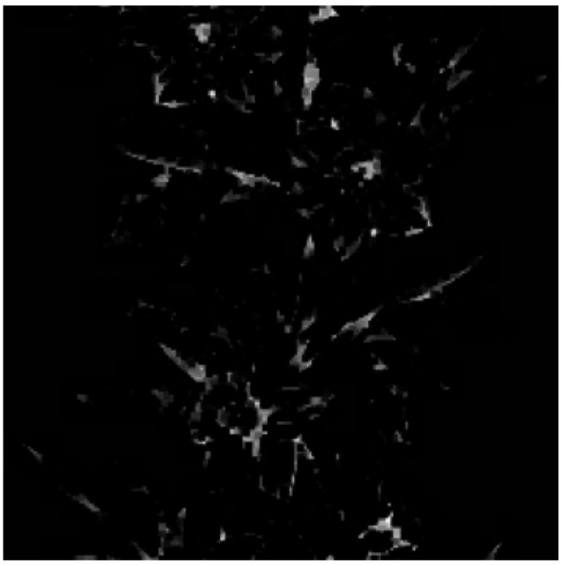


Overview of all the operation

```
    # Apply erosion
eroded_image = erosion(image, se)
display_images(image, eroded_image, 'Original Image', 'Eroded Image')

        # Apply dilation
dilated_image = dilation(image, se)
display_images(image, dilated_image, 'Original Image', 'Dilated
Image')

        # Apply opening
opened_image = opening(image, se)
display_images(image, opened_image, 'Original Image', 'Opened Image')

        # Apply closing
closed_image = closing(image, se)
display_images(image, closed_image, 'Original Image', 'Closed Image')

        # Apply hit and miss transformation
hit_and_miss_image = hit_and_miss(image, se, se_neg)
display_images(image, hit_and_miss_image, 'Original Image', 'Hit and
Miss Image')
```

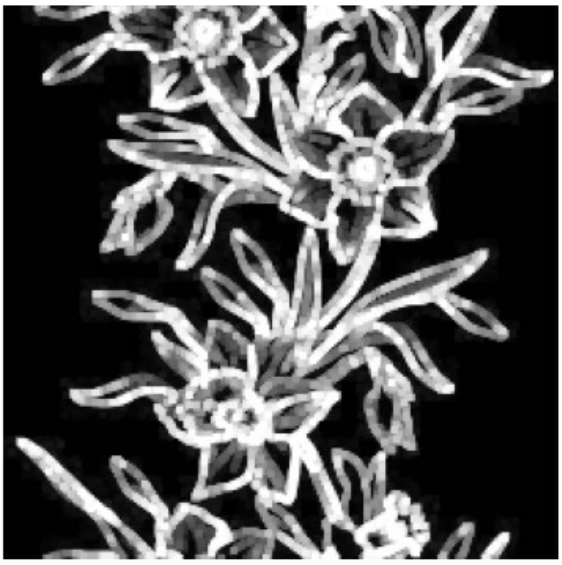
Original Image

Eroded Image



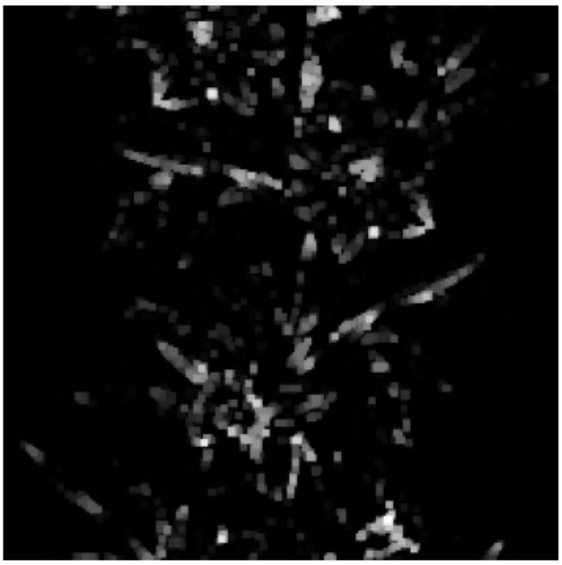
Original Image

Dilated Image



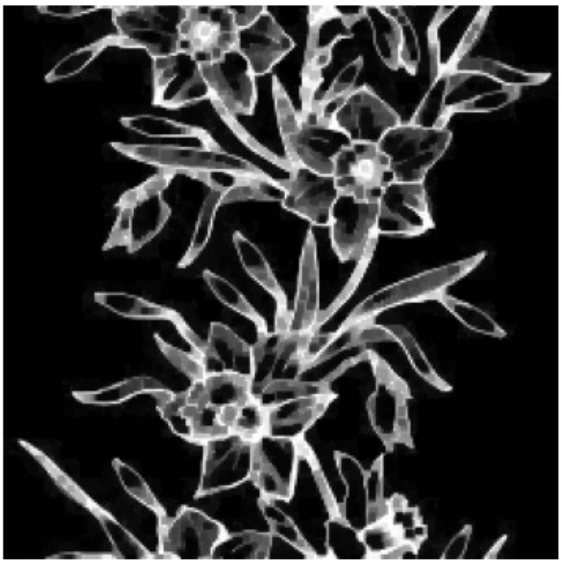
Original Image

Opened Image



Original Image

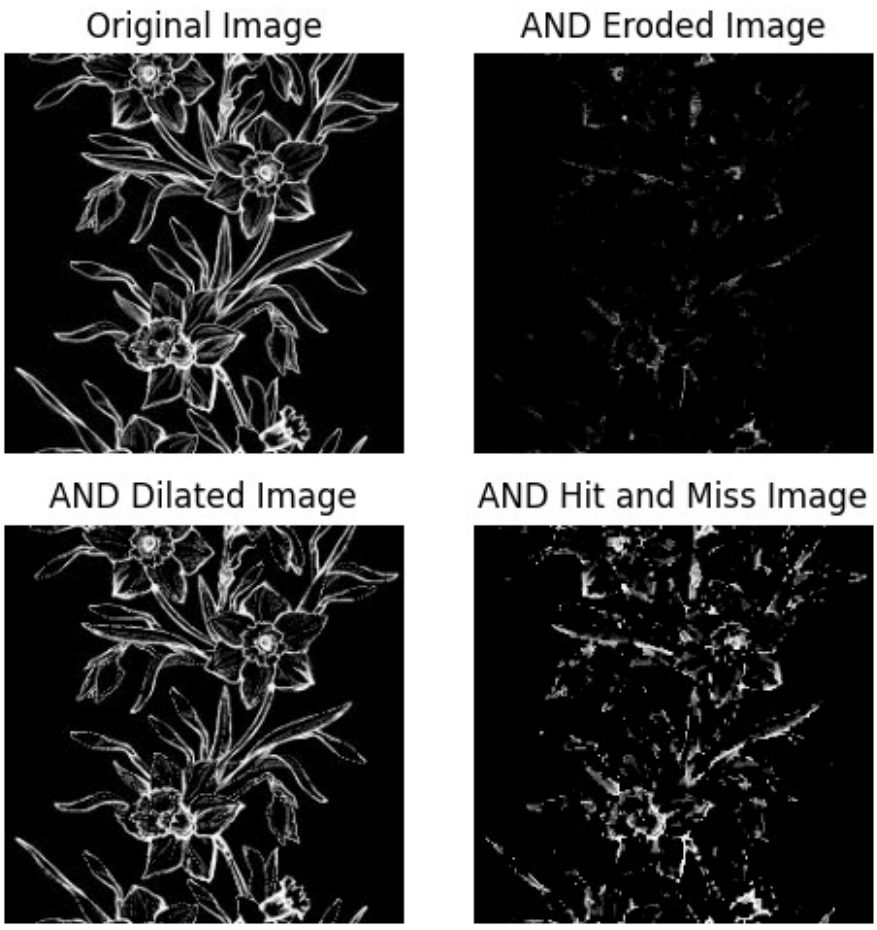
Closed Image



Performing AND Operation

```
# Perform AND operation for erosion
and_eroded_image = np.zeros_like(image)
for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        and_eroded_image[y, x] = image[y, x] & eroded_image[y, x]


# Perform AND operation for dilation
and_dilated_image = np.zeros_like(image)
for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        and_dilated_image[y, x] = image[y, x] & dilated_image[y, x]

# Perform AND operation for hit and miss
and_hit_and_miss_image = np.zeros_like(image)
for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        and_hit_and_miss_image[y, x] = image[y, x] &
hit_and_miss_image[y, x]




# Display function for 4 images with figsize
def display_images(img1, img2, img3, img4, title1, title2, title3,
title4, figsize=(5, 5)):
    fig, axes = plt.subplots(2, 2, figsize=figsize)

    axes[0, 0].imshow(img1, cmap='gray')
    axes[0, 0].set_title(title1)
```

```
    axes[0, 0].axis('off')

    axes[0, 1].imshow(img2, cmap='gray')
    axes[0, 1].set_title(title2)
    axes[0, 1].axis('off')

    axes[1, 0].imshow(img3, cmap='gray')
    axes[1, 0].set_title(title3)
    axes[1, 0].axis('off')

    axes[1, 1].imshow(img4, cmap='gray')
    axes[1, 1].set_title(title4)
    axes[1, 1].axis('off')

    plt.tight_layout()
    plt.show()

# Call the display function with 4 images
display_images(image, and_eroded_image, and_dilated_image,
and_hit_and_miss_image,
               'Original Image', 'AND Eroded Image', 'AND Dilated
Image', 'AND Hit and Miss Image',
               figsize=(5, 5))
```



Perform Complement

```
# Compute the complement of the images
complement_image = 255 - image
complement_eroded_image = 255 - eroded_image
complement_dilated_image = 255 - dilated_image
complement_hit_and_miss_image = 255 - hit_and_miss_image

# Display the complement images
display_images(complement_image, complement_eroded_image,
complement_dilated_image, complement_hit_and_miss_image,
               'Complement Original Image', 'Complement Eroded Image',
'Complement Dilated Image', 'Complement Hit and Miss Image',
               figsize=(5, 5))
```

Applications of Morphological Image Transformations

Erosion

- **Noise Removal**: Erosion is used to remove small-scale noise from binary images. It helps in eliminating small white noises and separating objects that are close to each other.
- **Shape Analysis**: Erosion can be used to analyze the shape and structure of objects within an image by reducing their size.

Dilation

- **Bridging Gaps**: Dilation is used to fill small holes and gaps in binary images. It helps in connecting disjoint objects that are close to each other.
- **Enhancing Features**: Dilation can be used to enhance specific features of an object, making them more prominent for further analysis.

Opening

- **Noise Reduction**: Opening is a combination of erosion followed by dilation. It is used to remove small objects from an image while preserving the shape and size of larger objects.

- **Object Separation**: Opening helps in separating objects that are connected by thin lines or small bridges.

Closing

- **Closing Gaps**: Closing is a combination of dilation followed by erosion. It is used to close small holes and gaps within objects in an image.
- **Smoothing Contours**: Closing helps in smoothing the contours of objects and filling small holes and gaps within the objects.

Hit and Miss Transformation

- **Pattern Recognition**: Hit and miss transformation is used for detecting specific patterns or shapes within an image. It is useful in template matching and shape recognition.
- **Skeletonization**: This transformation can be used to extract the skeleton of objects, which is useful in shape analysis and object recognition.

Conclusion:

Thus, on implementation of morphological transformation I have understood the variation in the images as in how the image looks almost vanishable when its performing erosion , how different it looks in dilation/hit and miss. I have understood that the structured element plays a very crucial role in this.