Computer Vision

Home Work 4

Name – ROHIT DAS Student ID - 61047086s

Project – Mathematical Morphology.

Language and library used: Python, OpenCV, Numpy.

Description: This program will perform the following functions while executing lena.bmp image file:

1. Dilation
2. Erosion
3. Opening
4. Closing
5. Hit and Miss

Parameters: None. Please Copy-paste the image path inside the program.

Algorithms Used –

Part 1: Dilation Morphology of Image

Basic Formula =

$$A \oplus B = \{c \in E^N \mid c = a + b \text{ for some } a \in A \text{ and } b \in B\}$$

Principal Code:

```
def binary_Dilation(image,Kernel):
    # Copy the Image
    dilated_Image = np.copy(image)
    # Centre value of the Kernel
    kernel_centre_point = tuple(x//2 for x in Kernel.shape)
    # Iterating over each pixel in original image
    for h_row in range(0, row):
        for w_col in range(0, col):
            # If the pixel is white
            if image[h_row,w_col] != 0:
                # Iterating over Kernel Shape
                for K_row in range(0,Kernel.shape[0]):
                    for K_col in range(0,Kernel.shape[1]):
                        # Get kernel value 1 from each iteration
                        if Kernel[K_row,K_col] == 1:
                            # Get the destination pixel location to be
filled out
                            n_row_val = h_row+(K_row-
kernel_centre_point[0])
                            n_col_val = w_col+(K_col-
kernel_centre_point[1])
                            # Avoiding out of range and putting the value
255 to destination pixel
                            if n_row_val < image.shape[0] and n_col_val <
```

```
image.shape[1]:
                                dilated_Image[n_row_val,n_col_val] = 255

    cv2.imshow("Applying Dilation",dilated_Image)
    cv2.waitKey(0)
```

Part 2: Erosion Morphology of Image

Basic Formula =

$$A \ominus B = \{x \in E^N | x + b \in A \text{ for every } b \in B\}$$

Principal Code:

```
def binary_Erosion(image,Kernel):
    # Copy the Image
    eroded_Image = np.zeros(image.shape, dtype=np.uint8)
    # Centre value of the Kernel
    kernel_centre_point = tuple(x//2 for x in Kernel.shape)
    # Iterating over each pixel in original image
    for h_row in range(0, row):
        for w_col in range(0, col):
            flag = True
            # Iterating over Kernel Shape
            for K_row in range(0,Kernel.shape[0]):
                for K_col in range(0,Kernel.shape[1]):
                    # Get kernel value 1 from each iteration
                    if Kernel[K_row,K_col] == 1:
                        # Get the destination pixel location to be filled
out
                        n_row_val = h_row+(K_row-kernel_centre_point[0])
                        n_col_val = w_col+(K_col-kernel_centre_point[1])
                        # Avoiding out of range
                        if n_row_val < image.shape[0] and n_col_val <
image.shape[1]:
                            # If point doesn't match with Kernel Value
                            if image[n_row_val,n_col_val] == 0:
                                flag = False
                                break
            # if full Kernel match at original image (r,c) then put white
pixel
            if flag is True:
                eroded_Image[h_row,w_col] = 255

    cv2.imshow("Applying Erosion",eroded_Image)
    cv2.waitKey(0)
```

Part 3: Opening Morphology of Image

Basic Formula-

$$BoK = (B \ominus K) \oplus K$$

Principal Code –

```
def binary_Opening(image,Kernel):
    opened_image = binary_Dilation(binary_Erosion(image,Kernel),Kernel)
    cv2.imshow("Opening",opened image)
    cv2.waitKey(0)
    return binary_Dilation(binary_Erosion(image,Kernel),Kernel)
```

Part 4: Closing Morphology of Image

Basic Formula –

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

Principal Code-

```
def binary_Closing(image,Kernel):
    closed_image = binary_Erosion(binary_Dilation(image,Kernel),Kernel)
    cv2.imshow("Closing",closed_image)
    cv2.waitKey(0)
    return binary_Dilation(binary_Erosion(image,Kernel),Kernel)
```

Part 5: Hit and Miss Morphology of Image

Basic Formula –

$$A \otimes (J, K) = (A \ominus J) \cap (A^c \ominus K)$$

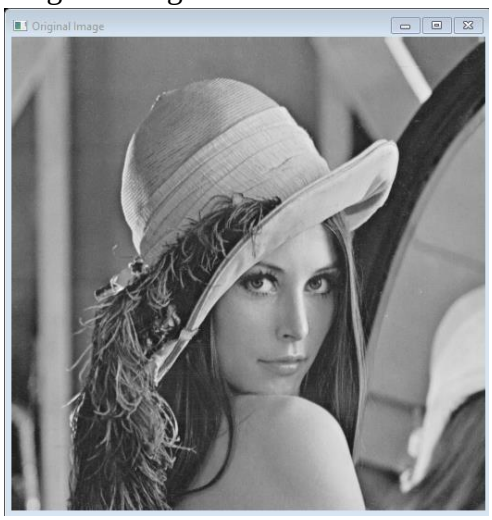
Principal Code-

```
def Hit_and_Miss(image):
    # Complement of Binary image.Basically this will reverse the pixel
values
    image_complement = -image + 255
    # Declaring J_kernel and K_kernel
    J_Kernel =np.array([[0, -1], [0, 0], [1, 0]])
    K_Kernel = np.array([[-1, 0], [-1, 1], [0, 1]])
    # Creating Separate J_kernel and K_kernel image
    image_with_j_kernel = binary_Erosion(image, J_Kernel)
    image_with_k_kernel = binary_Erosion(image_complement,K_Kernel)
    # Performing Intersection of two images
    hit_and_miss_image =
intersection(image_with_j_kernel,image_with_k_kernel)
    cv2.imshow("Hit and Miss Image",hit_and_miss_image)
    cv2.waitKey(0)
def intersection(image1,image2):
    row = image1.shape[0]
    col = image1.shape[1]
    intersected_image = np.zeros(image1.shape,dtype=np.uint8)
    # intersected_image = np.copy(image1)
    for h_row in range(0, row):
        for w_col in range(0, col):
            if image1[h_row, w_col] == 255 and image2[h_row, w_col] == 255:
```

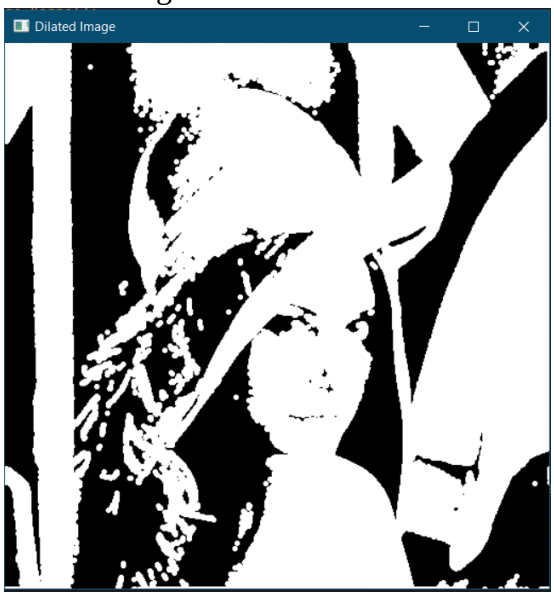
```
                intersected_image[h_row, w_col] = 255
    return (intersected_image)
```

Example:

- Original image



- Dilated Image



- Eroded Image

- Opening Morphology

- Closing Morphology

- Hit and Miss Morphology of Image