4TN4 Assignment 4

Adam Bujak (400113347)

February 20, 2022

1 Theory

1.1 Morphology Operations

Consider the following binary image.

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	1	1	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

1.1.1 Erosion

Apply erosion on the image using the following structuring element.

$$SE = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

To apply erosion on the image above, each of the pixels (surrounding the operating pixel) that are indicated as "active" by the structuring element are ANDed together.

For example, consider the calculation for the pixel [3,2] (where [x,y] are counted from 0 and the top left of the image)

If we consider the 9 pixels around the pixel [3,2] we have the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

given the structuring element above, we only consider the following pixels (x means it is not considered):

$$\begin{bmatrix} x & 0 & x \\ x & 1 & 1 \\ x & 0 & x \end{bmatrix}$$

and when we AND together all of the considered pixels we see that we get 0 (since there exists a 0 in the considered set of pixels).

Doing this for each pixel, we get the following image:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	0	0	0
0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

1.1.2 Dilation

Apply dilation on the image using the following structuring element.

$$SE = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The process for the dilation of an image is the exact same as described above for erosion, except instead of ANDing the pixels together we OR them.

For example, consider the calculation for the pixel [3,2] (where [x,y] are counted from 0 and the top left of the image)

If we consider the 9 pixels around the pixel [3,2] we have the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

given the structuring element above, we only consider the following pixels (x means it is not considered):

$$\begin{bmatrix} 0 & x & x \\ x & 1 & x \\ x & x & 1 \end{bmatrix}$$

and when we OR together all of the considered pixels we see that we get 1 (since there exists a 1 in the considered set of pixels).

Doing this for each pixel, we get the following image:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	1	1	1	0
0	1	0	0	0	0	1	1	1	1	1	0
0	0	1	0	0	0	0	1	1	1	1	0
0	0	0	1	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

1.1.3 Opening

$$SE = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

To apply opening on an image we perform erosion followed by dilation on an image.

Applying erosion on the image above using the structuring element specified we get the following image:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

and then applying dilation on the resulting image using the structuring element specified leaves us with the following image:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

1.2 Distance and Boundary

1.2.1 Boundary

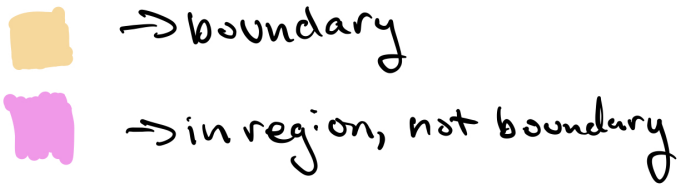
Mark the boundary pixels of the region in the sense of 4-neighbours and 8-neighbours respectively in the images below.

4-neighbours

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	1	0	0
0	1	1	1	0	0	0
0	1	0	1	0	0	0
0	0	0	0	0	0	0

8-neighbours

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	1	0	0
0	1	1	1	0	0	0
0	1	0	1	0	0	0
0	0	0	0	0	0	0



1.2.2 Distance

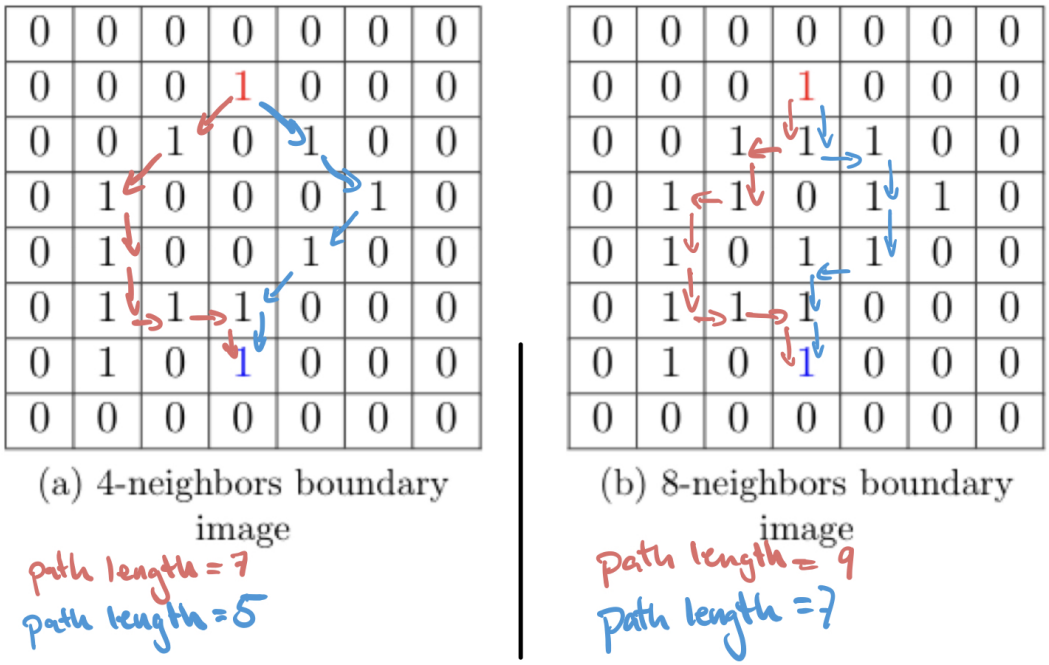
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	1	0	0	0
0	0	0	0	0	0	0

(a) 4-neighbors boundary image

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	1	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	1	0	0	0
0	1	0	1	0	0	0
0	0	0	0	0	0	0

(b) 8-neighbors boundary image

NOTE: p is the red pixel, q is the blue pixel



(a) 4-neighbors boundary image

(b) 8-neighbors boundary image

Figure 1: D_m distances between p and q in both boundary images

1.2.3 D_8 distance of 4-neighbors

The D_8 distance is found by the maximum of the distance in the x direction and the y direction. Thus:

$$D_8 = max(|x_p - x_q|, |y_p - y_q|)$$

$$D_8 = max(|3 - 3|, |1 - 6|)$$
$$D_8 = 5$$

1.2.4 D_m distance of 4-neighbors

As shown in Figure 1 a), the D_m distance between p and q is 5.

1.2.5 D_m distance of 4-neighbors

The D_8 distance is found by the maximum of the distance in the x direction and the y direction. Thus:

$$D_8 = max(|x_p - x_q|, |y_p - y_q|)$$

$$D_8 = max(|3 - 3|, |1 - 6|)$$
$$D_8 = 5$$

1.2.6 D_4 distance of 8-neighbors

The D_4 distance is found by the sum of the distance in the x direction and the y direction. Thus:

$$D_4 = |x_p - x_q| + |y_p - y_q|$$

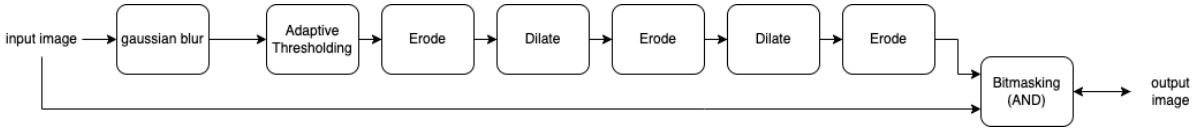
$$D_4 = |3 - 3| + |1 - 6|$$
$$D_4 = 5$$

1.2.7 D_m distance of 8-neighbors

As shown in Figure 1 b), the D_m distance between p and q is 7.

1.3 Extracting License Plate

The following is my proposal for an algorithm to detect a license plate in the given image:



2 Implementation

Figure 2: Bitmask of where the algorithm thinks the license plate is

Figure 3: Extracted license plate from image

```
import numpy as np
import cv2

s=np.array([[0,0,0],[0,1,0],[1,1,1]])

def erode(image, size):
    size = size
    shape = cv2.MORPH_ELLIPSE
    element = cv2.getStructuringElement(shape, (2 * size + 1, 2 * size + 1), (size, size))
    return cv2.erode(working_image, element)

def dilate(image, size):
    size = size
    shape = cv2.MORPH_RECT
    element = cv2.getStructuringElement(shape, (2 * size + 1, 2 * size + 1), (size, size))
    return cv2.dilate(working_image, element)

image = cv2.imread('3.png', 0)

working_image = cv2.GaussianBlur(image, (51,51),cv2.BORDER_DEFAULT)

working_image = cv2.adaptiveThreshold(working_image,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)

working_image = erode(working_image, 2)
working_image = dilate(working_image, 15)
working_image = erode(working_image, 35)
working_image = dilate(working_image, 70)
working_image = erode(working_image, 40)


cv2.imwrite('bitmask.png', working_image)
working_image = cv2.bitwise_and(working_image, image)

cv2.imwrite('license.png', working_image)
```

Figure 4: Code used to extract license plate