HW 3 Report

Name: 陳翰雯

ID：B08902092

Problem 1: Morphological Processing

(a) Boundaries Extraction

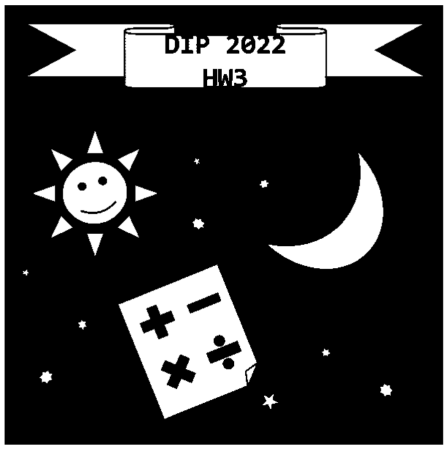


Fig 1. sample1.png

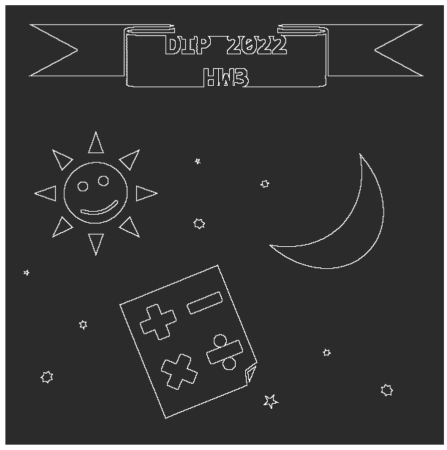


Fig 2. result1.png

The boundaries extraction are done in 2 steps:

1. Erosion
2. Subtracting the eroded image from the original image

The erosion are done with the following structuring element:

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

(b) Hole Filling

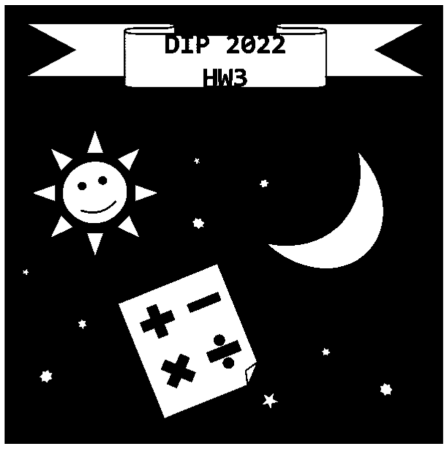


Fig 1. sample1.png

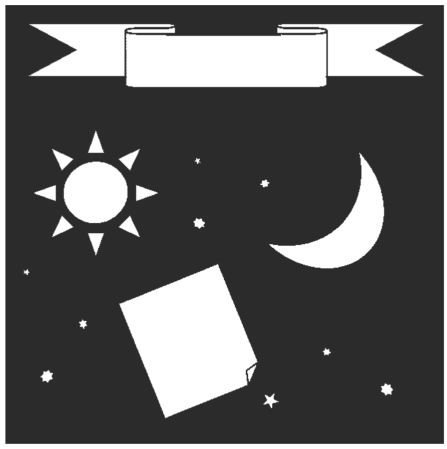


Fig 3. result2.png

Here, coordinates of the starting point and the ending point are first given. This is to make sure that the hole filling process are only done within the boundaries. The coordinates chosen are the following (left: starting coordinates, write: ending coordinates):

```
The Smiley Face boundaries:
[[93, 230], [153, 285]]

Banner boundaries:
[[179, 42], [429, 112]]

Math signs boundaries: (Plus, Minus, Times, Divide)
[[179, 407], [238, 456], [241, 390], [300, 421], [204, 474], [266, 524], [262, 452], [335, 501]]
```

Once coordinates are decided, the hole filling are then done with the following steps:

1. If the current pixel is not foreground (not 1), mark it as 1
2. Do dilation. The dilation is done with the following structuring element

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

1. Intersect the dilated image with F^c (the inverted version of the image (all pixel 1 as 0, all pixel 0 as 1))
2. Step 1 and 3 are done for every pixel inside the assigned boundaries. The relevant equation could be written as:

$$G_i(j,k) = (G_{i-1}(j,k) \oplus H(j,k)) \cap F^c(j,k)$$

Once done, do union with the original image. The relevant equation could be written as:

$$G(j,k) = G_i(j,k) \cup F(j,k)$$

Below are the whole process of hole filling:

Fig 4. after filling hole for 1st object

Fig 5. after filling hole for the 2nd object

Fig 6. after filling hole for the 3rd object

Fig 7. after filling all holes

Fig 8. after adding both images together

(c) Skeletonizing

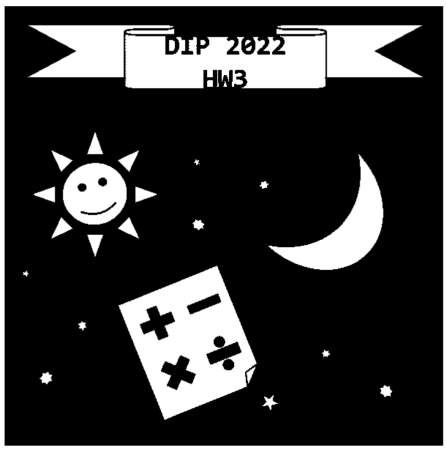


Fig 1. sample1.png

Fig 10. result3.png

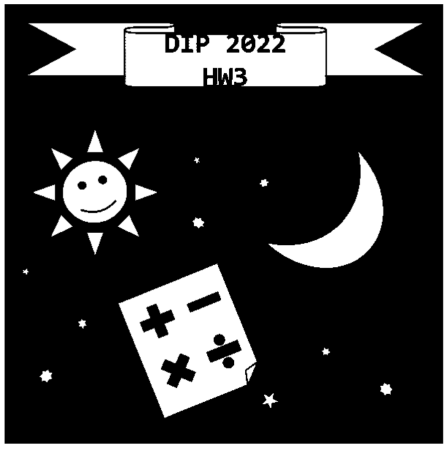


Fig 11. reversed sample1.png

Fig 12. result4.png

2 methods are tried to accomplish the task:

1. Zhang-Suen thinning Algorithm
2. Conditional mark patterns

Zhang-Suen algorithm work as follows

We first obtain the pixel's neighbors with the pattern:

$\begin{bmatrix} P9 & P2 & P3 \\ P8 & P1 & P4 \\ P7 & P6 & P5 \end{bmatrix}$ where, $P1$ is the current pixel and $P2$ to $P9$ is its neighbors

- Step 1: all pixels that satisfy the following conditions are noted.
  - The pixel is a foreground and has eight neighbors
  - $2 \leq S \leq 6$ where S is the number of neigbors that are foreground pixels
  - $2 \leq A \leq 6$ where A is the number of $0 \rightarrow 1$ transitions in the sequence $P2, P3, P4, P5, P6, P7, P8, P9, P2$
  - At least one of $P2$ and $P4$ and $P6$ is background pixels
  - At least one of $P4$ and $P6$ and $P8$ is background pixels
- Step 2:
  - The pixel is a foreground and has eight neighbors
  - $2 \leq S \leq 6$ where S is the number of neigbors that are foreground pixels
  - $2 \leq A \leq 6$ where A is the number of $0 \rightarrow 1$ transitions in the sequence $P2, P3, P4, P5, P6, P7, P8, P9, P2$
  - At least one of $P2$ and $P4$ and $P8$ is background pixels
  - At least one of $P2$ and $P6$ and $P8$ is background pixels

The skeletonizing process by Zhang-Suen Algorithm is as follows:

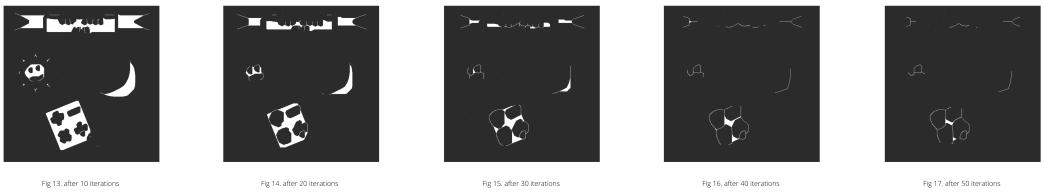


Fig 13. after 10 iterations | Fig 14. after 20 iterations | Fig 15. after 30 iterations | Fig 16. after 40 iterations | Fig 17. after 50 iterations

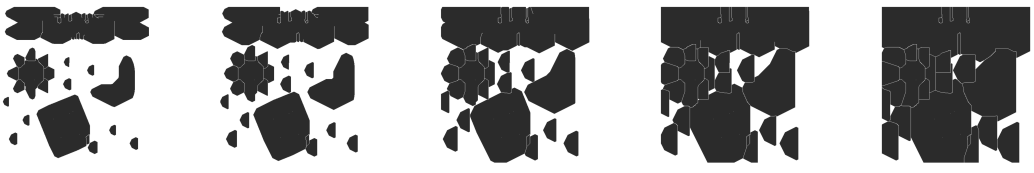


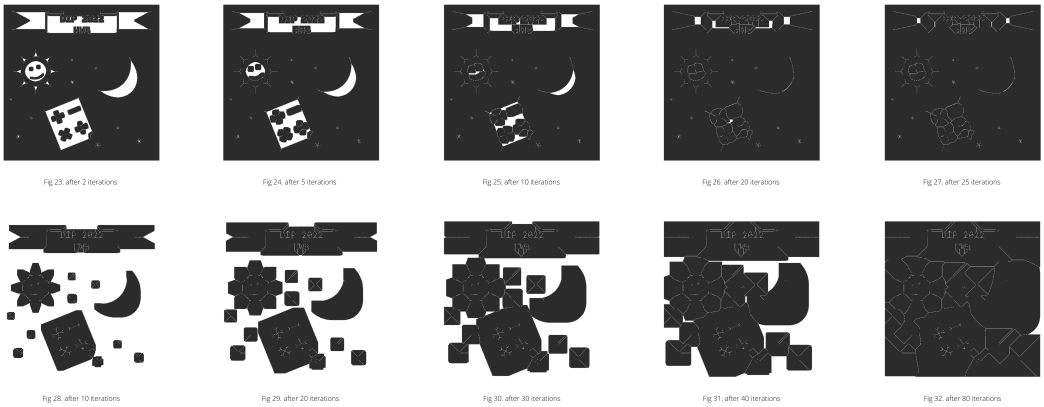
Fig 18. after 30 iterations | Fig 19. after 50 iterations | Fig 20. after 80 iterations | Fig 21. after 110 iterations | Fig 22. after 140 iterations

However, In the end, the conditional mark patterns written on Table 14.3 inside the Digital Image Processing book by William K. Pratt is used. This is mainly because it showed a great result in skeletonizing.

The code implementations of those conditional mark patterns is as follows:

```
stage1_mask = np.array([
        # Bond = 4
        [[0,1,0],[0,1,1],[0,0,0]],
        [[0,1,0],[1,1,0],[0,0,0]],
        [[0,0,0],[1,1,0],[0,1,0]],
        [[0,0,0],[0,1,1],[0,1,0]],
        [[0,0,1],[0,1,1],[0,0,1]],
        [[1,1,1],[0,1,0],[0,0,0]],
        [[1,0,0],[1,1,0],[1,0,0]],
        [[0,0,0],[0,1,0],[1,1,1]],
        # Bond = 6
        [[1,1,1],[0,1,1],[0,0,0]],
        [[0,1,1],[0,1,1],[0,0,1]],
        [[1,1,1],[1,1,0],[0,0,0]],
        [[1,1,0],[1,1,0],[1,0,0]],
        [[1,0,0],[1,1,0],[1,1,0]],
        [[0,0,0],[1,1,0],[1,1,1]],
        [[0,0,0],[0,1,1],[1,1,1]],
        [[0,0,1],[0,1,1],[0,1,1]],
        # Bond = 7
        [[1,1,1],[0,1,1],[0,0,1]],
        [[1,1,1],[1,1,0],[1,0,0]],
        [[1,0,0],[1,1,0],[1,1,1]],
        [[0,0,1],[0,1,1],[1,1,1]],
        # Bond = 8
        [[0,1,1],[0,1,1],[0,1,1]],
        [[1,1,1],[1,1,1],[0,0,0]],
        [[1,1,0],[1,1,0],[1,1,0]],
        [[0,0,0],[1,1,1],[1,1,1]],
        # Bond = 9
        [[1,1,1],[0,1,1],[0,1,1]],
        [[0,1,1],[0,1,1],[1,1,1]],
        [[1,1,1],[1,1,1],[1,0,0]],
        [[1,1,1],[1,1,1],[0,0,1]],
        [[1,1,1],[1,1,0],[1,1,0]],
        [[1,1,0],[1,1,0],[1,1,1]],
        [[1,0,0],[1,1,1],[1,1,1]],
        [[0,0,1],[1,1,1],[1,1,1]],
        # Bond = 10
        [[1,1,1],[0,1,1],[1,1,1]],
        [[1,1,1],[1,1,1],[1,0,1]],
        [[1,1,1],[1,1,0],[1,1,1]],
        [[1,0,1],[1,1,1],[1,1,1]],
        # Bond = 11
        [[1,1,1],[1,1,1],[0,1,1]],
        [[1,1,1],[1,1,1],[1,1,0]],
        [[1,1,0],[1,1,1],[1,1,1]],
        [[0,1,1],[1,1,1],[1,1,1]],
    ])
```

The skeletonizing process by conditional mark patterns is as follows:



We could clearly see the great difference between the original image and the inverted image. In the original image the skeletonization goes inward as if the white pixels area are shrinking, however, in inverted image it goes outward is it as if the black pixels area are growing.

(d) Connected Component Labeling

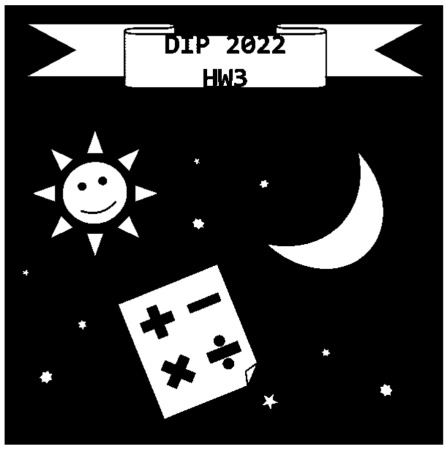


Fig 1. sample1.png

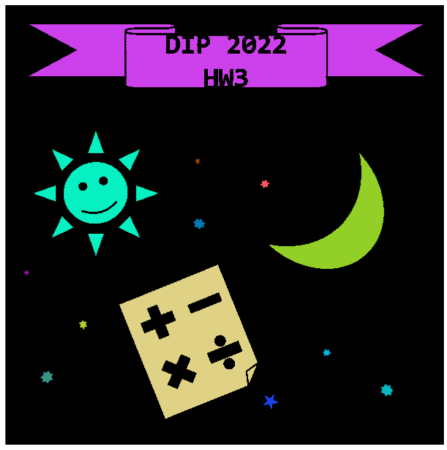


Fig 33. result5.png

2-Pass Algorithm with 8-connectivity representation is used to finish the following task. The 2 pass algorithm works as follows:

The First Pass:

For each non-zero pixels, we check its neighbors:

1. if it has NO non-zero neighbors, it is a new component, so a new label is given
2. if it has non-zero neighbors and all non-zero neighbors has same label, it is connected, so same label is given
3. if it has non-zero neighbors but the neighbors have different labels, we give label with the smallest value. This also means that both labels are actually the same component. Hence, we take note of it by saving it in the form of dictionary. This will be solved in the second pass.

The Second Pass:

We go through each non-zero pixel and check if it has equivalent labels. We then change its label to its equivalent label

Here we tried different kernel sizes and decided to use kernel size 31x31 as it gives the best result.

Below are the few experiments done with different kernel sizes:

Fig 34. kernel size = 3x3

Fig 35. kernel size = 11x11

Fig 36. kernel size = 21x21

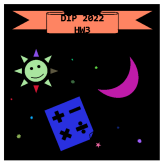


Fig 37. kernel size = 25x25

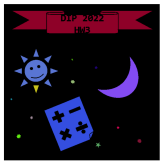


Fig 38. kernel size = 29x29

Problem 2: Texture Analysis

(a) Law's method

The feature vectors are obtained in 2 steps:

1. Convolution
2. Energy Computation

In the first step the convolution is done based on the equation $M_i(j,k) = F(j,k) \otimes H_i(j,k)$. The H is a 3x3 mask with a total of 9 laws. The 9 laws used is as following:

$$H_1 = \frac{1}{36}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad H_2 = \frac{1}{12}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad H_3 = \frac{1}{12}\begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix}$$

$$H_4 = \frac{1}{12}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad H_5 = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad H_6 = \frac{1}{4}\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{bmatrix}$$

$$H_7 = \frac{1}{12}\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix} \quad H_8 = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_9 = \frac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

The convolution is then done with the following code:

```
M = signal.convolve2d(img, mask, mode='same')
```

Once done, we then compute the energy based on the equation $T_i(j,k) = \sum\sum |M_i(j+m, k+n)|^2$ with the following code:

```
T = signal.convolve2d(M * M, size, mode='same')
```

The window size here is set to 13x13

Below are the images after each laws:

Fig 41. Law 1

Fig 42. Law 2

Fig 43. Law 3

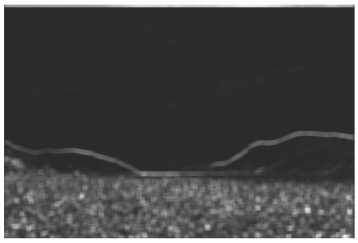


Fig 44. Law 4

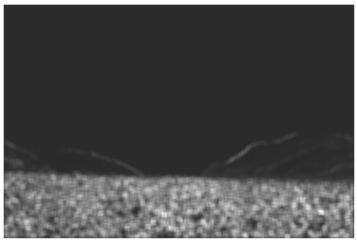


Fig 45. Law 5

Fig 46. Law 6

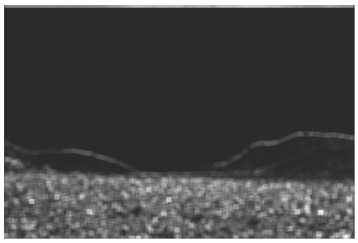


Fig 47. Law 7

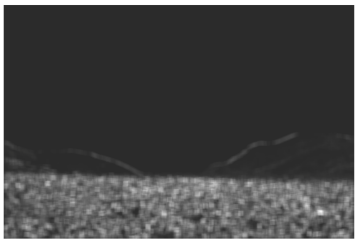


Fig 48. Law 8

Fig 49. Law 9

(b) K-means algorithm

Fig 50. sample2.png

Fig 51. result6.png

Here, we first change the shaping and ordering of the obtained features from (a), which is of size (9, 600, 900) with the following code:

```
features = part_a()
features = np.moveaxis(features, 0, -1)
features = features.reshape(-1,9)
```

Next, first of all, we then select K centroids. These centroids are selected randomly.

Second, we then compute the sum of the squared distance between data points and all the centroids

Third, we then assign the data points to the closer cluster / centroid.

Once done, if the obtained number of clusters is lower than K, we randomly select another K centroids and repeat the step.

However if it is more than K, we then compute the centroids for clusters by taking the average of all the data points that belong to each cluster and repeat the step.

The whole process is repeated for n iterations. Here we choose $n = 110$ with $K = 5$.

Below are the progress for different iterations:

Fig 52. iterations = 30

Fig 53. iterations = 60

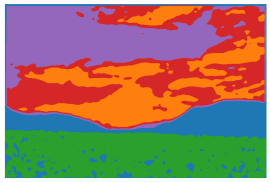


Fig 54. iterations = 90

Fig 55. iterations = 110

(c) Improving classification

Fig 56. sample2.png after edge crispening

Fig 57. result7.png

The modifications I made is by doing edge crispening to the image first before moving on to finding feature vectors and doing classification.

Here several methods are tried such as increasing brightness, power law method, etc. However, edge crispening showed to have a better result. A reason for this may be due to the accentuation made after edge crispening. The image after edge crispening is shown to have better pattern details.

These are the modified original images:

Fig 58. sample2.png

Fig 59. sample2.png after Power's Law

Fig 60. sample2.png after edge crispening

These are how modification to original images affects classification:

Fig 61. iterations = 30

Fig 65. iterations = 30
(Power's Law)

Fig 69. iterations = 30
(Edge crispening)

Fig 62. iterations = 60

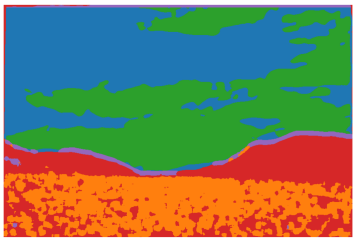


Fig 66. iterations = 60
(Power's Law)

Fig 70. iterations = 60
(Edge crispening)

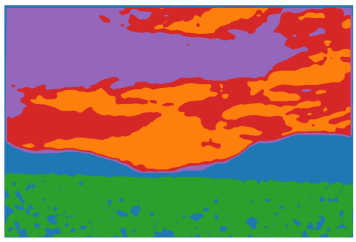


Fig 63. iterations = 90

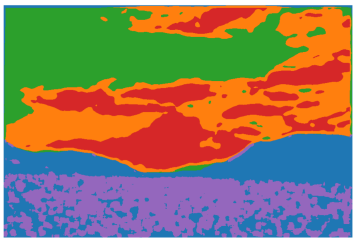


Fig 67. iterations = 90
(Power's Law)

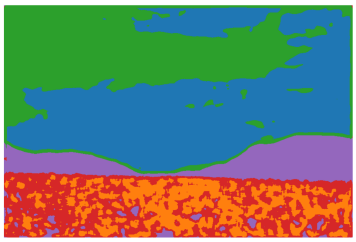


Fig 71. iterations = 90
(Edge crispening)

Fig 64. iterations = 110

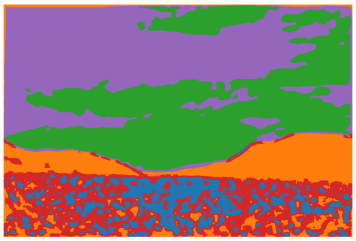


Fig 68. iterations = 110
(Power's Law)

Fig 72. iterations = 110
(Edge crispening)