Digital Image Processing - Lab Session 7

Sofia Noemi Crobeddu
student number: 032410554

International Master of Biometrics and Intelligent Vision
Université Paris-Est Cretéil (UPEC)

January 14, 2025

Contents

Introduction 3

1 Morphology Algorithms 3

Introduction

In this project different operations and algorithms of morphology in the context of image processing are analyzed. In particular, the focus is on Boundary extraction, Region filling, Hit-and-miss transformation and Skeletonization.

1 Morphology Algorithms

In this part of the assignment, it is applied the **Boundary extraction** technique. It is a method used in image processing, based on morphological operations, to identify the boundaries of an object in an image. This goal is achieved through the erosion of the object using a structuring element and, after, subtracting the result from the original binary image. The operation is defined as followed:

$$A - (A \ominus B)$$

where A is the original image, $\ominus$ is the erosion operation, and B is the structuring element used for erosion. In this project the structuring element chosen is of 3x3 dimension, and it is the **4-connectivity** or **V4**, where a pixel is connected with its neighbors that lie horizontally or vertically, i.e. it has 4 possible neighbors (up, down, left, right).
Moreover, the binarization of *boundary* image is performed using **Otsu's method** to separate the object from the background, and the erosion is made through one iteration.
The code for this analysis is shown below and the resulting images are in Figure 1. Note that the original image was also read and shown in grayscale.

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import NoNorm
from PIL import Image
import cv2
from skimage.filters import threshold_otsu

#Load image
image_bound = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 7 - IP\\IP5\\IP5\\boundary.png").convert('L')

#To array
image_array_bound = np.array(image_bound)

#Function for Otsu thresholding
def otsu_method(img):
    otsu_threshold = threshold_otsu(img)
    #Binarization
    binary_image = (img > otsu_threshold).astype(np.uint8)
    return binary_image, otsu_threshold

#Binarization
binary_image_bound, otsu_threshold_bound = otsu_method(image_array_bound)

#Erosion through V4 structuring element
struct_el_v4 = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
eroded_v4_bound = cv2.erode(binary_image_bound, struct_el_v4, iterations=1) #1 iteration

#Boundary extraction
boundary_extr = binary_image_bound - eroded_v4_bound

#Plot
fig, ax = plt.subplots(1, 4, figsize=(16, 4))

#Original image
ax[0].imshow(image_bound, cmap='gray', norm=NoNorm())
ax[0].set_title("Boundary image.")
ax[0].axis('off')

#Otsu binarization
ax[1].imshow(binary_image_bound, cmap='gray')
ax[1].set_title("Binary boundary image with Otsu thresholding.")
ax[1].axis('off')

#Eroded image
ax[2].imshow(eroded_v4_bound, cmap='gray')
ax[2].set_title("Eroded boundary image with V4.")
```

```
ax[2].axis('off')

#Boundary extraction
ax[3].imshow(boundary_extr, cmap='gray')
ax[3].set_title("Boundary extract. of boundary image.")
ax[3].axis('off')

plt.tight_layout()
plt.show()
```

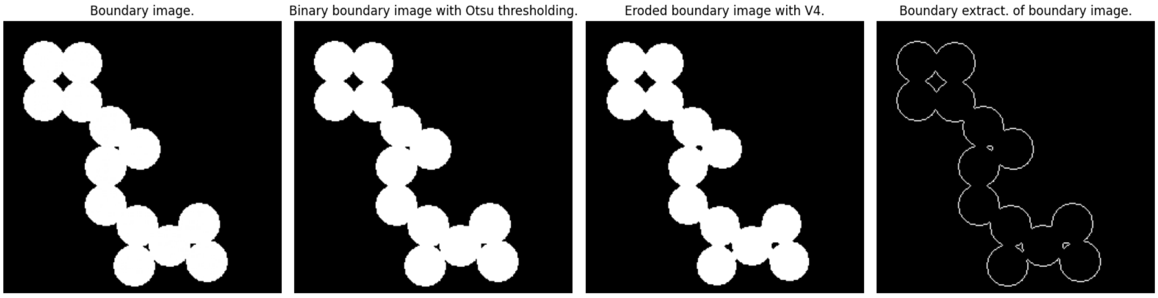

Fig. 1: Boundary extraction on *boundary* image.

In this second part of the task, it is performed **Region filling** technique. It is a method used in image processing to fill holes, i.e. empty regions or 0 value pixels completely surrounded by 1 value ones, in grayscale or binary images. The operation is defined as followed:

$$x_k = (x_{k-1} \oplus B) \cap A \quad \text{where } k = 1, 2, 3, \ldots \quad \text{until } x_k = x_{k-1}$$

where A is the original image, x_k is the region filled at k step, and B is the structuring element.
The first application of this technique is here done to the binary *coins* image, while the second one involves the grayscale version of the same image. The function used is *binary_fill_holes* from *scipy* library. Moreover, as for the point before, the binarization is made through Otsu's thresholding.
The code is shown below, and the grayscale image with filled holes is obtained by the multiplication of pixel by pixel with the binary image with filled holes (Figure 2). In this sense, the binary image with filled holes represents a sort of mask, and the resulting image is a filled grayscale version of *coins* (Figure 3).

```
from scipy.ndimage import binary_fill_holes

#Load image
image_coins = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 5 - IP\\IP5_v2\\IP5_v2\\coins.bmp").convert('L')

#To array
image_array_coins = np.array(image_coins)

#Binarization
binary_image_coins, otsu_threshold_coins = otsu_method(image_array_coins)

#Filling holes in the binary image
filled_binary_coins = binary_fill_holes(binary_image_coins).astype(np.uint8)

#Plot
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

#Original image
ax[0].imshow(image_coins, cmap='gray', norm=NoNorm())
ax[0].set_title("Coins image.")
ax[0].axis('off')

#Otsu binarization
ax[1].imshow(binary_image_coins, cmap='gray')
ax[1].set_title("Binary coins image with Otsu thresholding.")
ax[1].axis('off')

#Eroded image
```

```python
ax[2].imshow(filled_binary_coins, cmap='gray')
ax[2].set_title("Filled binary coins image.")
ax[2].axis('off')
```

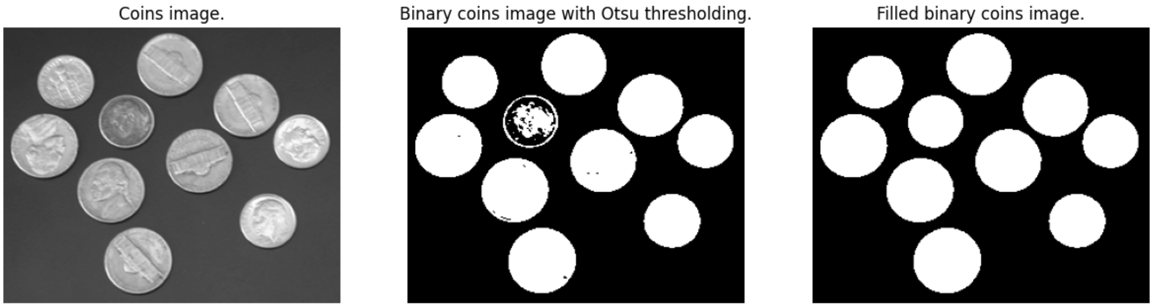

Fig. 2: Region filling on *coins* image.

```python
#Filling holes in the gray binary image
#filled_binary_coins is like a mask to fill the holes in the binary one
#Now we apply the mask to the gray level image
grayscale_filled_coins = image_array_coins * filled_binary_coins

#Plot
plt.figure(figsize=(6, 6))
plt.imshow(grayscale_filled_coins, cmap='gray')
plt.axis('off')
plt.show()
```

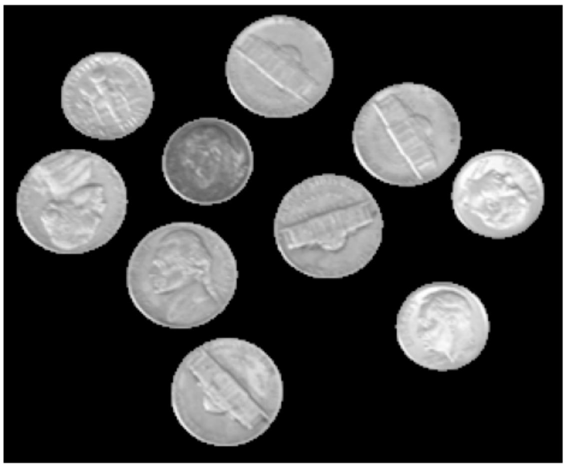

Fig. 3: Grayscale filled *coins* image.

In the third part of the task, it is performed the **Hit-and-miss transformation** technique on *text* image. It is a binary morphological operation used to detect specified patterns in images. Basically, it does a joint analysis on foreground and background pixels of a binary image, through a structuring element. The foreground pixels are the region usually with binary value 1, and are considered the active part. The background pixels, instead, are the region usually with binary value 0, and are considered the inactive part. The structuring element, in this context, represents the pixels (can be both foreground and background) to be detected, this transform compares it with the binary image. If a zone of the pixels in the binary image corresponds to the structuring element, the match occurs.

Hence, the function *binary_hit_or_miss* returns a boolean array with False if the single pixel match hasn't occur, and True otherwise. The operation is defined as followed:

$$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2)$$

where A is the original image, $B = (B_1, B_2)$ is the structuring element, B_1 is the set of elements of B associated with an object, and B_2 is the set of elements of B associated with the background.
In this part of the exercise, the structuring element is e image, caught by cutting the part of letter 'e' in *text* image, with its original dimensionality. The complete binary array of *text* image (dimensionality 248x257) is shown in Figure 4, using the function *set_printoptions* from *Numpy*, and the same is done with the binary array of e image (dimensionality 9x11), shown in Figure 5.
Finally, the result of the application of Hit-and-miss transform is obtained applying also a second structuring element, i.e. the transpose of binary array of e image, since in *text* image there are also backwards writing. The final result is shown in Figure 6.

```python
#Load image
image_text = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 7 - IP\\IP5\\IP5\\text.png").convert('L')

#To array
image_array_text = np.array(image_text)

#Binarization
binary_image_text, otsu_threshold_text = otsu_method(image_array_text)

#Dimensions
binary_image_text.shape

#To look entirely at the array of text image, we use this method:
np.set_printoptions(threshold=np.inf)
#Print entirely the array
print(binary_image_text)
#After the visualization, set again the default-previous one
np.set_printoptions(threshold=1000)
```

```
[[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 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 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 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]
 [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 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 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 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]
 [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 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 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 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]
 [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 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 1 1]]
```

Fig. 4: Output of the array of binary *text* image.

```python
#Loading the image founded by cutting the area of letter 'e' in the original one
image_e = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment 7
    - IP\\IP5\\IP5\\e.png").convert('L')

#To array
image_array_e = np.array(image_e)
```

```

#Binarization
binary_image_e, otsu_threshold_e= otsu_method(image_array_e)

#Dimensions
binary_image_e.shape

#Visualization of the entire array to build the structuring element
np.set_printoptions(threshold=np.inf)
print(binary_image_e)
#Back to default set
np.set_printoptions(threshold=1000)
```

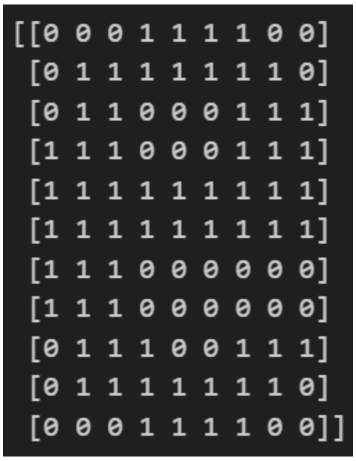

Fig. 5: Output of the array of *e* image.

```
from scipy.ndimage import binary_hit_or_miss

#Strurturing element for letter 'e' is our image array 'e'
#Strurturing element for letter 'e' rotated of 90 degrees
structuring_element_transpose = np.transpose(binary_image_e)[::-1]

#Hit-or-miss with both the structuring elements
hit_original = binary_hit_or_miss(binary_image_text, structure1=binary_image_e)
hit_transposed = binary_hit_or_miss(binary_image_text, structure1=structuring_element_transpose)

#Combining the results through the logical OR
hit_or_miss_text = hit_original | hit_transposed

#Plot
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

#Original image
ax[0].imshow(image_text, cmap='gray', norm=NoNorm())
ax[0].set_title("Text image.")
ax[0].axis('off')

#Otsu binarization
ax[1].imshow(binary_image_text, cmap='gray')
ax[1].set_title("Binary text image with Otsu thresholding.")
ax[1].axis('off')

#Hit-and-miss transformation
ax[2].imshow(hit_or_miss_text, cmap='gray')
ax[2].set_title("Hit-and-missed transformation for letter 'e'.")
ax[2].axis('off')
```

Fig. 6: Hit-or-miss transformation on *text* image.

In this last part of the task, it is performed **Skeletonization** technique. It is an operation used to reduce a binary figure to its essential skeleton, preserving its shape and connectivity, but also iteratively removing pixels until a minimal representation is achieved. The skeleton represents a central structure.
The function used to perform this operation is *skeletonize* and here it is applied on *skeleton* image. The code is shown below and the result is in Figure 7.

```
from skimage.morphology import skeletonize

image_skeleton = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 7 - IP\\IP5\\IP5\\skeleton.png").convert('L')

#To array
image_array_skeleton = np.array(image_skeleton)

#Binarization
binary_image_skeleton, otsu_threshold_skeleton= otsu_method(image_array_skeleton)

#Skeletonizing the binary image
binary_image_skeleton = binary_image_skeleton.astype(bool) #Image with values True/False
skeletonized_image = skeletonize(binary_image_skeleton)

#Plot
fig, ax = plt.subplots(1, 3, figsize=(18, 6))

#Original image
ax[0].imshow(image_skeleton, cmap='gray', norm=NoNorm())
ax[0].set_title("Skeleton image.")
ax[0].axis('off')

#Otsu binarization
ax[1].imshow(binary_image_skeleton, cmap='gray')
ax[1].set_title("Binary skeleton image with Otsu thresholding.")
ax[1].axis('off')

#Skeletonized image
ax[2].imshow(skeletonized_image, cmap='gray')
ax[2].set_title("Skeletonized skeleton image.")
ax[2].axis('off')
```

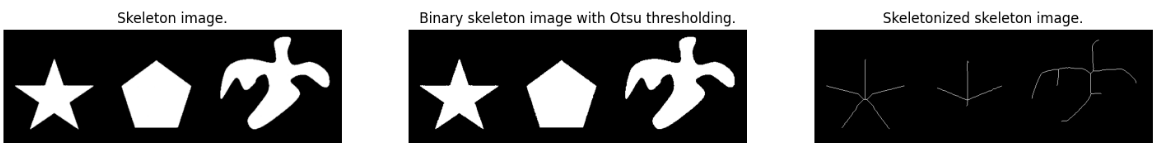

Fig. 7: Morphological skeleton on *skeleton* image.

References

[1] Documentation, S.-I.: Skeletonization Example in Scikit-Image. https://scikit-image.org/docs/stable/auto_examples/edges/plot_skeleton.html

[2] Linzhe, W.: Digital Image Processing in C - Chapter 8: Erosion, Dilation, Opening, Closing, Boundary. https://medium.com/@wilson.linzhe/digital-image-processing-in-c-chapter-8-erosion-dilation-opening-closing-boundary-5f505c731f19

[3] Technology, U.: Digital Image Processing - Lecture 11 (n.d.). https://uotechnology.edu.iq/ce/Lectures/Image_Processing_4th/DIP_Lecture11.pdf

[4] YouTube: Hit-or-Miss Transform Explanation. https://www.youtube.com/watch?v=79ggHWJqPDE

[5] YouTube: Binary Transform Demonstration. https://www.youtube.com/watch?v=pKvPF76fkMk

[6] YouTube: Scipy Binary Hit-or-Miss Explanation. https://www.youtube.com/watch?v=P8NwA_p07a8

[7] Documentation, S.: Binary Hit-or-Miss Transformation. https://tedboy.github.io/scipy/generated/scipy.ndimage.binary_hit_or_miss.html

[8] Overflow, S.: Hit-and-Miss Transform for Detecting Branched Point and Endpoint in Scikit-Image. https://stackoverflow.com/questions/16241708/hit-and-miss-transform-for-detecting-branched-point-and-endpoint-in-scikit-image

[9] Majidzadeh, F.: Resources-lecture3-08012025, (2025)

[10] MathWorks: Structuring Elements (2024). https://fr.mathworks.com/help/images/structuring-elements.html

[11] Majidzadeh, F.: Digital Image Processing and Pattern Recognition, (2023)