Digital Image Processing - Lab Session 9

Sofia Noemi Crobeddu
student number: 032410554

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

February 4, 2025

Contents

Introduction **3**

Descriptors **3**

Introduction

In this project the aim is to write and use an algorithm that lets to classify the objects in *Cards* image. There are four categories: spade, club, heart and diamond. These symbols were analyzed before from the original cards.

Descriptors

First, the images considered to understand the shapes and the characteristics of the symbols, i.e. the cards *Spade*, *Club*, *Heart*, *Diamond*, are loaded. The same is also done with *Cards* image that has to be analyzed for the classification. The result is shown in Figure 1.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

#Load images
card_spade = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 9 - IP\\IP8\\IP8\\spade.png")
card_club = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment
     9 - IP\\IP8\\IP8\\club.png")
card_heart = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 9 - IP\\IP8\\IP8\\heart.jpg")
card_diamond = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 9 - IP\\IP8\\IP8\\diamond.jpg")
image_cards = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 9 - IP\\IP8\\IP8\\cards.bmp")

#To array
card_spade_array = np.array(card_spade)
card_club_array = np.array(card_club)
card_heart_array = np.array(card_heart)
card_diamond_array = np.array(card_diamond)
image_array_cards = np.array(image_cards)

#Plot
fig, ax = plt.subplots(1, 5, figsize=(20, 5))

ax[0].imshow(card_spade)
ax[0].set_title("Card Spade image.")
ax[0].axis('off')

ax[1].imshow(card_club)
ax[1].set_title("Card Club image.")
ax[1].axis('off')

ax[2].imshow(card_heart)
ax[2].set_title("Card Heart image.")
ax[2].axis('off')

ax[3].imshow(card_diamond)
ax[3].set_title("Card Diamond image.")
ax[3].axis('off')

ax[4].imshow(image_cards, cmap='gray')
ax[4].set_title("Cards image.")
ax[4].axis('off')

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

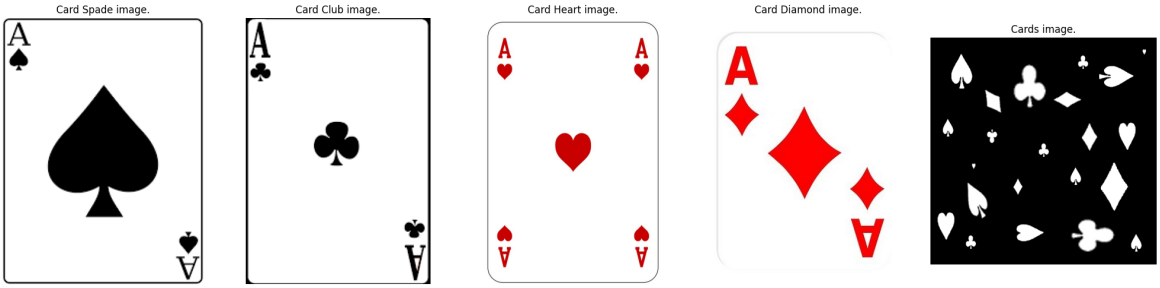

Fig. 1: Initial images.

After, the images are binarized. For *Heart* and *Diamond* images that are originally in RGB, it is applied **Otsu method** that finds an optimal global threshold by separating the gray levels into two classes, after transforming the two images in grayscale. The same method is also applied to *Cards* image that has black background and white foreground.
For *Spade* and *Club* images, the situation is different due to white background and black objects. In general Otsu's algorithm calculates a single global threshold value based on the image histogram. This works well if the illumination is uniform and the distribution is homogeneous. In this case, the objects are very small compared to the white background, so the histogram is unbalanced and the chosen threshold by Otsu would not be correct.
For this reason it was chosen the **Bradley-Roth Adaptive method**, that is a form of adaptive thresholding, i.e. it calculates the threshold locally instead of using a global value like Otsu. It works in this way:

- it is calculated the local average of the pixels in a window of size *window_r* around each pixel;
- the value of the pixel is compared with a threshold adapted to that region;
- the threshold is finally calculated as

$$T(x, y) = (average\ of\ pixels\ in\ the\ window)(1 - threshold)$$

where *threshold* is a value between 0 and 1 that controls how permissive the final threshold T should be in relation to the local average.

For both the images the parameter *window_r* was set at low values, after many trials, in order to not include too much variation in the background and not separating the object well. Same for *threshold* parameter, that was set to low values in order to not lose details in the images.
After thresholding, the binarized images were inverted to have the objects of interest (i.e. symbols) in white and the background in black, facilitating the analysis of the properties of the regions later. The resulting images are shown in Figure 2.

```
from skimage.filters import threshold_otsu
from skimage.color import rgb2gray

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

#Function for Bradley-Roth Adaptive Thresholding
def faster_bradley_threshold(image, threshold=75, window_r=5):
    percentage = threshold / 100.0
    window_diam = 2 * window_r + 1
    img = np.array(image.convert('L')).astype(np.float32)  #Converting into grayscale
    means = ndimage.uniform_filter(img, window_diam)  #Calculate local mean
    result = np.zeros(img.shape, dtype=np.uint8)  #Empty binarized image
    result[img >= percentage * means] = 255  #Adaptative thresholding
    return result

#Converting to grayscale Heart and Diamond images
card_heart_gray = rgb2gray(card_heart_array)
```

```
card_diamond_gray = rgb2gray(card_diamond_array)

#Binarization of the images
binary_card_spade = faster_bradley_threshold(card_spade, threshold=30, window_r=25)
binary_card_club = faster_bradley_threshold(card_club, threshold=10, window_r=15)
binary_card_heart, otsu_threshold_heart = otsu_method(card_heart_gray)
binary_card_diamond, otsu_threshold_diamond = otsu_method(card_diamond_gray)
binary_image_cards, otsu_threshold_cards = otsu_method(image_array_cards)

#Inversion
binary_card_spade = 1-binary_card_spade
binary_card_club = 1-binary_card_club
binary_card_diamond = 1-binary_card_diamond
binary_card_heart = 1-binary_card_heart

#Plot for check
fig, ax = plt.subplots(1, 5, figsize=(20, 5))

ax[0].imshow(binary_card_spade, cmap='gray')
ax[0].set_title("Binary Card Spade image.")
ax[0].axis('off')

ax[1].imshow(binary_card_club, cmap='gray')
ax[1].set_title("Binary Card Club image.")
ax[1].axis('off')

ax[2].imshow(binary_card_heart, cmap='gray')
ax[2].set_title("Binary Card Heart image.")
ax[2].axis('off')

ax[3].imshow(binary_card_diamond, cmap='gray')
ax[3].set_title("Binary Card Diamond image.")
ax[3].axis('off')

ax[4].imshow(binary_image_cards, cmap='gray')
ax[4].set_title("Binary Cards image.")
ax[4].axis('off')

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

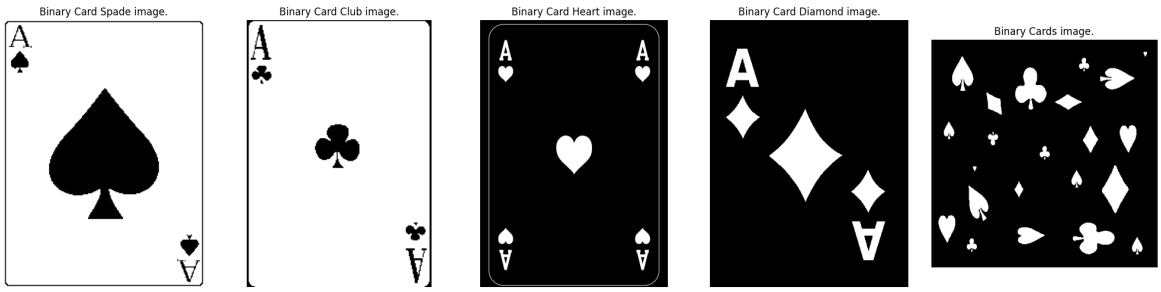

Fig. 2: Binarized images.

Using `label` and `regionprops` functions from skimage, the properties such as Centroid, Area, Perimeter, Eccentricity and Solidity were extracted for each region (i.e. object) identified in the image. The other characteristics Width, Height and Circularity were calculated using the information of the objects already extracted. All these values were registered in datasets, one per image.
This process was applied on the binarized images of the symbols, but also on *Cards* image. This was done since, in the image were classification will be performed, the shapes of the objects are completely different in some cases to the ones of the original images for spade, club, heart and diamond symbols. Hence, to set the limits and the range of values of each class, it is also necessary to check *Cards* image. Moreover, for this reason, the characteristics of Centroid, Area, Perimeter, Width, Height and Circularity don't generalized and are not very useful in this case. Then, the limits of the classes will be set using Eccentricity and Solidity.
The images resulting, with also the numbered labels identified, are shown in Figure 3. As it is possible to see, the informative labels for this task are:

- 1,2,3,7,8,9 for *Spade* dataset (see the rows highlighted in yellow in Table 1);
- 5,7,9 for *Club* dataset (see the rows highlighted in red in Table 2);
- 4,5,6,7,8 for *Heart* dataset (see the rows highlighted in green in Table 3);
- 2,3,4 for *Diamond* dataset (see the rows highlighted in purple-blue in Table 4).

Table 5 shows the parameters information for its objects, and the colors of the rows correspond to the symbol used in previous tables (yellow for Spade, red for Club, green for Heart and purple-blue for Diamond).

```python
import pandas as pd
from skimage.measure import label, regionprops
from scipy import ndimage

#Function to extract the characteristics of each object in an image
def analyze_regions(binary_img, title=""):
    #Labeling the regions in the binary image
    labeled_img = label(binary_img, connectivity=binary_img.ndim)
    props = regionprops(labeled_img)

    #Creating a dataframe to store the properties of each region
    data_extrcted = []
    for region in props:
        minr, minc, maxr, maxc = region.bbox
        width = maxc - minc
        height = maxr - minr
        circularity = (4 * np.pi * region.area) / (region.perimeter ** 2) if region.perimeter > 0 else 0
        data_extrcted.append([
            region.label, region.centroid, width, height, region.area, region.perimeter, circularity, region.eccentricity, region.solidity
        ])

    df = pd.DataFrame(data_extrcted, columns=["Label", "Centroid", "Width", "Height", "Area", "Perimeter", "Circularity", "Eccentricity", "Solidity"])

    return df, labeled_img, props

#Analyzing the regions for each individual card
df_spade, img_spade, _ = analyze_regions(binary_card_spade, "card_spade")
df_club, img_club, _ = analyze_regions(binary_card_club, "card_club")
df_heart, img_heart, _ = analyze_regions(binary_card_heart, "card_heart")
df_diamond, img_diamond, _ = analyze_regions(binary_card_diamond, "card_diamond")
df_cards, img_cards, props_cards = analyze_regions(binary_image_cards, "image_cards")

#Plot
fig, ax = plt.subplots(1, 5, figsize=(22, 15))
images = [img_spade, img_club, img_heart, img_diamond, img_cards]
titles = ["card_spade", "card_club", "card_heart", "card_diamond", "image_cards"]

#Function to plot all images in a single row
def plot_labeled_image(ax, img, title, df):
    ax.imshow(img, cmap='nipy_spectral')
    ax.set_title(title)
    ax.axis('off')

    #Plot centroids with labels
    for _, row in df.iterrows():
        centroid = row["Centroid"]
        ax.scatter(centroid[1], centroid[0], color='none', s=50, marker='x')
        ax.text(centroid[1], centroid[0], str(row["Label"]),
                color='white', fontsize=14, ha='center', va='center',
                bbox=dict(facecolor='none', edgecolor='none'))

#Application of the function to each subplot
for i in range(5):
    plot_labeled_image(ax[i], images[i], titles[i], eval(f'df_{titles[i].split("_")[1]}'))

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

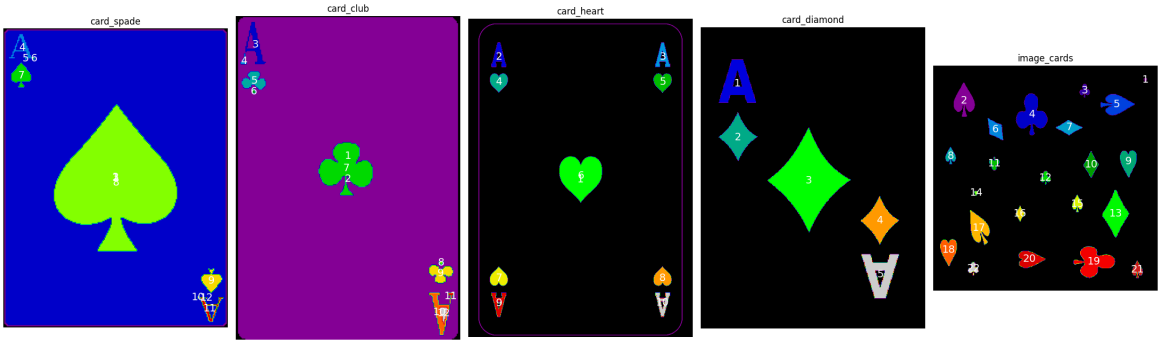

Fig. 3: Labeled images.

Label	Centroid	Width	Height	Area	Perimeter	Circularity	Eccentricity	Solidity
1	(129.0, 97.0)	195	259	916.0	915.313708	0.013739	0.591846	0.018137
2	(129.0, 97.0)	193	257	896.0	896.000000	0.014025	0.592447	0.018069
3	(128.29, 96.95)	191	255	40414.0	1717.755447	0.172115	0.671283	0.829976
4	(16.09, 15.76)	20	21	99.0	83.763456	0.177311	0.601884	0.400810
5	(25.5, 19.0)	1	2	2.0	0.000000	0.000000	1.000000	1.000000
6	(25.5, 26.0)	1	2	2.0	0.000000	0.000000	1.000000	1.000000
7	(40.16 14.95)	17	21	204.0	64.148232	0.622975	0.360400	0.853556
8	(132.84, 97.30)	107	127	7671.0	435.487373	0.508289	0.320442	0.879904
9	(218.07, 180.26)	18	21	201.0	60.941125	0.680119	0.235977	0.837500
10	(232.5, 169.0)	1	2	2.0	0.000000	0.000000	1.000000	1.000000
11	(242.02, 178.90)	20	21	97.0	82.798990	0.177800	0.601705	0.388000
12	(233.0, 176.0)	1	1	1.0	0.000000	0.000000	0.000000	1.000000

Table 1: Dataset *df_spade*.

Label	Centroid	Width	Height	Area	Perimeter	Circularity	Eccentricity	Solidity
1	(115.81, 93.14)	187	269	1303.0	1268.970563	0.010168	0.097578	0.025903
2	(134.57, 92.95)	183	268	46989.0	1563.706674	0.241488	0.734950	0.958822
3	(23.19, 15.75)	17	35	194.0	114.491378	0.185980	0.956750	0.557471
4	(36.65, 6.24)	5	9	17.0	13.621320	1.151384	0.909797	0.586207
5	(53.43, 14.67)	20	15	187.0	70.769553	0.469201	0.686313	0.809524
6	(61.875, 14.625)	3	4	8.0	7.414214	1.828815	0.851469	0.888889
7	(125.79, 91.55)	45	44	1144.0	205.166522	0.341525	0.457812	0.793343
8	(204.57, 170.57)	4	3	7.0	6.621320	2.006405	0.643079	0.875000
9	(213.27, 170.19)	20	15	188.0	69.941125	0.482950	0.695480	0.810345
10	(245.63, 169.28)	17	35	198.0	115.041631	0.188003	0.954701	0.568966
11	(232.35, 178.77)	5	9	17.0	13.621320	1.151384	0.909797	0.586207
12	(246.63, 172.71)	6	14	51.0	29.449747	0.738953	0.925844	0.850000

Table 2: Dataset *df_club*.

To set the limits of the classes in `classify_label()` function, it was performed a merge between the values of the objects in the original images and the ones from *Cards* image. In particular, for Club symbol it is considered the solidity with a range taken from both the original image and *Cards*. For Heart symbol the range of solidity and for eccentricity to identify the small hearts, was taken from both the images, while the range of eccentricity for big hearts was taken just from *Cards* (i.e. it was taken manually in this case). For Spade symbol the range of solidity was taken from both the images, and for Diamond symbol the ranges in the two images were different in decimals, to it is set as the last options within the cases.
The code to implement the analysis is shown below, and the results of the classification are shown in Figure 4 and Table 6 with the occurrences.

Label	Centroid	Width	Height	Area	Perimeter	Circularity	Eccentricity	Solidity
1	(499.51, 349.12)	636	970	6246.0	6081.492424	0.002122	0.683339	0.010166
2	(117.63, 94.89)	46	76	1638.0	323.438600	0.196761	0.848763	0.726708
3	(117.56, 605.48)	46	76	1639.0	323.195959	0.197177	0.849769	0.724580
4	(194.63, 94.91)	55	60	2276.0	196.450793	0.741096	0.271510	0.944398
5	(194.69, 605.51)	56	60	2288.0	198.622366	0.728802	0.270743	0.942728
6	(487.18, 350.25)	133	144	13121.0	480.399062	0.714451	0.277628	0.950177
7	(805.27, 94.81)	55	60	2275.0	195.622366	0.747058	0.272032	0.949896
8	(805.29, 605.34)	56	60	2287.0	197.450793	0.737154	0.291563	0.939992
9	(883.38, 94.75)	47	75	1625.0	324.852814	0.193504	0.845098	0.719663
10	(883.16, 605.46)	46	75	1625.0	322.367532	0.196499	0.846579	0.724476

Table 3: Dataset *df_heart*.

Label	Centroid	Width	Height	Area	Perimeter	Circularity	Eccentricity	Solidity
1	(86.32, 56.65)	58	67	2021.0	315.130988	0.255737	0.670195	0.778206
2	(169.52, 57.07)	60	75	1806.0	199.492424	0.570262	0.615033	0.796999
3	(236.82, 166.75)	129	162	8358.0	435.340187	0.554185	0.613030	0.795470
4	(299.35, 277.50)	59	75	1817.0	196.735065	0.589931	0.607337	0.811161
5	(382.25, 278.54)	58	68	2027.0	319.338095	0.249783	0.674397	0.768385

Table 4: Dataset *df_diamond*.

Label	Centroid	Width	Height	Area	Perimeter	Circularity	Eccentricity	Solidity
1	(32.07, 483.5)	8	11	60.0	29.727922	0.853163	0.678192	0.882353
2	(79.40, 69.99)	48	77	2122.0	269.865007	0.366153	0.716747	0.886383
3	(56.63, 344.95)	23	30	367.0	100.597980	0.455719	0.610720	0.771008
4	(111.25, 223.89)	72	95	3549.0	338.149278	0.390031	0.637071	0.792541
5	(87.97, 417.61)	77	48	2122.0	269.865007	0.366153	0.716747	0.886383
6	(144.56, 141.66)	36	49	1036.0	147.154329	0.601206	0.829407	0.895419
7	(140.49, 309.058)	60	34	1030.0	147.095454	0.598203	0.827943	0.903509
8	(205.27, 39.25)	24	39	530.0	132.361436	0.380157	0.724812	0.861789
9	(217.30, 444.85)	38	61	1531.0	175.438600	0.625079	0.746246	0.939264
10	(224.94, 359.50)	34	60	1030.0	147.095454	0.598203	0.827943	0.903509
11	(222.37, 138.95)	23	30	367.0	100.597980	0.455719	0.610720	0.771008
12	(254.63, 255.95)	23	30	367.0	100.597980	0.455719	0.610720	0.771008
13	(337.07, 415.42)	61	108	3267.0	269.563492	0.564985	0.829874	0.910281
14	(289.07, 97.50)	8	11	60.0	29.727922	0.853163	0.678192	0.882353
15	(314.27, 328.26)	24	39	530.0	132.361436	0.380157	0.724812	0.861789
16	(336.60, 197.62)	20	36	359.0	84.704581	0.628768	0.829528	0.890819
17	(368.85, 103.65)	45	79	2076.0	281.421356	0.329400	0.824381	0.870075
18	(419.30, 34.85)	38	61	1531.0	175.438600	0.625079	0.746246	0.939264
19	(445.89, 365.75)	95	72	3549.0	338.149278	0.390031	0.637071	0.792541
20	(440.15, 219.30)	61	38	1531.0	175.438600	0.625079	0.746246	0.939264
21	(464.27, 465.26)	24	39	530.0	132.361436	0.380157	0.724812	0.861789
22	(461.63, 90.95)	23	30	367.0	100.597980	0.455719	0.610720	0.771008

Table 5: Dataset *df_cards*.

A final consideration is that, in this case, the objects in the original images don't completely reflect the shapes and the measures of the objects in *Cards* images. For this reason it was necessary to check them in the image to be analyzed and classified. In general, for classification problems in image processing and computer vision, the CNN or RNN are suitable, but in this case it would have been necessary to have more images or symbols to train a model like this. Hence, *regionprops* and *label* are more appropriate.

```
#Function to classify the symbols' labels
def classify_label(region):
    #Club classification
    if 0.77 <= region.solidity <= 0.81:
        #range taken from cards and club
        return "Club"

    #Heart classification
    if 0.88 <= region.solidity <= 0.95 and (0.73 <= region.eccentricity <= 0.76 or 0.66 <= region.
     eccentricity <= 0.68):
```

```
        #range taken from cards and heart    #range taken from cards (bigger heart)    #range taken from cards and heart (smaller heart)
        return "Heart"

    #Spade classification
    elif 0.82 <= region.solidity <= 0.89: #range taken from cards and spade
        return "Spade"

    #Diamond classification
    else: #remaining symbols
        return "Diamond"

#Classify each region based on its properties
df_cards["Symbol"] = df_cards.apply(lambda row: classify_label(props_cards[int(row["Label"]) - 1]), axis=1)

#Count the occurrences
symbol_counts = df_cards["Symbol"].value_counts().reset_index().rename(columns={"index": "Symbol", "Symbol": "Count"})

#Display the card with bounding boxes and labels
fig, ax = plt.subplots(figsize=(6, 8))
ax.imshow(binary_image_cards, cmap='gray')
ax.axis('off')

#Draw bounding boxes and labels for each detected prop
for region in props_cards:
    minr, minc, maxr, maxc = region.bbox
    rect = plt.Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edgecolor='dodgerblue', linewidth=2)
    ax.add_patch(rect)
    ax.text(minc, minr, classify_label(region), bbox=dict(facecolor='lightblue', alpha=0.6), fontsize=8, color='black')

plt.show()
```

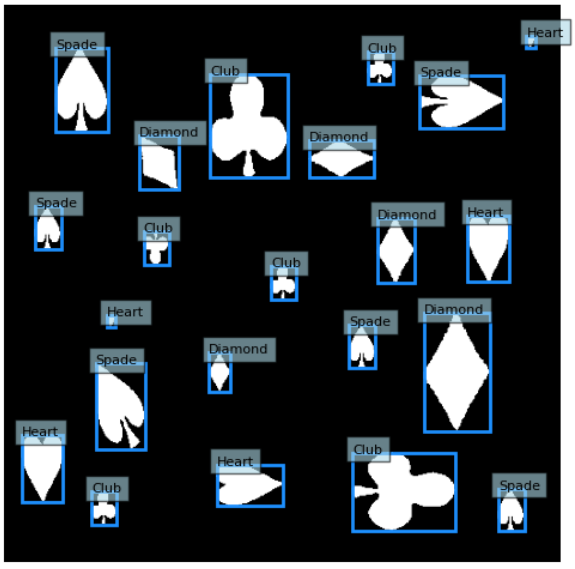

Fig. 4: Symbol recognition in *Cards* image.

Symbol	Occurrences
Spade	6
Club	6
Heart	5
Diamond	5

Table 6: Symbols count.

References

[1] YouTube: Image Processing Tutorial - Region Properties and Object Labeling (2023). https://www.youtube.com/watch?v=u3nG5_EjfM0

[2] YouTube: Object Detection with skimage.measure.regionprops (2023). https://www.youtube.com/watch?v=Gq7mp3G94ao

[3] scikit-image: Region Properties Example - Skimage.measure.regionprops. (2024). https://scikit-image.org/docs/0.24.x/auto_examples/segmentation/plot_regionprops.html

[4] Forum, I.s.: In what order does skimage.measure.regionprops label the objects? (2021). https://forum.image.sc/t/in-what-order-does-skimage-measure-regionprops-label-the-objects/51323

[5] Overflow, S.: How can I use bwlabel or regionprops to extract set of pixels of each label? (2014). https://stackoverflow.com/questions/23141223/how-can-i-use-bwlabel-or-regionprops-to-extract-set-of-pixels-of-each-label

[6] Overflow, S.: Bradley-Roth adaptive thresholding algorithm - how do I get better performance? (2015). https://stackoverflow.com/questions/33091755/bradley-roth-adaptive-thresholding-algorithm-how-do-i-get-better-performance

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