

DIGITAL IMAGE PROCESSING

ELL715

ASSIGNMENT 5

10 November 2023

Group Members

1. Bhavik Shankla (2020MT60873)
2. Ritika Soni (2020MT10838)
3. Sai Kiran Gunnala (2020MT60889)
4. Sai Niketh Varanasi (2020MT60895)
5. Yash Pravin Shirke (2020MT60986)

1

1.1 Explanation

This MATLAB code first resizes the signature image to 128 by 128 pixels and then converts it to gray-scale. To highlight the signature curve, the image is then binarized and goes through thinning operations. The centre of mass is then determined by the code after extracting the coordinates of the black pixels that represent the signature. After that, the signature is moved to the origin, and the covariance matrix for the modified coordinates is computed. The rotation angle is calculated from this matrix's minimum eigenvalue, and the signature is then rotated in accordance with that value. Relocating the signature to its original location is the last step. Using this MATLAB code we can make signatures robust which can be used in applications that recognise or verify signatures.

1.2 MATLAB Code

```
1 SigImage=imread('sig.jpg'); % Load the image file and store it as the variable  
   SigImage.  
2 figure,imshow(SigImage);  
3  
4 I2=imresize(SigImage,[128 ,128]);  
5 figure,imshow(I2);  
6  
7 I3=rgb2gray(I2);  
8 I3=im2double(I3);  
9 I3=imbinarize(I3); %converting image to black and white  
10 I3 = bwmorph(~I3, 'thin', inf); %thining the image  
11 I3=~I3;  
12 figure,imshow(I3);  
13  
14 i=1;  
15 k=1;  
16  
17 while i<=128  
18     j=1;
```

```

19     while j<=128
20         if I3(i, j)==0
21             u(k)=i;
22             v(k)=j;
23             k=k+1;
24             I3(i, j)=1;
25         end
26         j=j+1;
27     end
28     i=i+1;
29 end
30
31 C=[u;v];%the curve of the signature
32 N=k-1;%the number of pixels in the signature
33 oub=0;
34 for i=1:N
35     oub= oub+ C(1,i);
36 end
37 oub=oub/N; %the original x co-ordinate center of mass of the image
38 ovb=0;
39 for i=1:N
40     ovb= ovb+ C(2,i);
41 end
42 ovb=ovb/N; %the original y co-ordinate center of mass of the image
43
44 %moving the signature to the origin
45 for i=1:N
46     u(i)=u(i)-oub+1;
47     v(i)=v(i)-ovb+1;
48 end
49 % the new curve of the signature
50 C=[u;v];
51
52 ub=sum(C(1,:))/N;
53 vb=sum(C(2,:))/N;
54 ubSq=sum((C(1,:)-ub).^2)/N;
55 vbSq=sum((C(2,:)-vb).^2)/N;
56
57 uvb=0;
58 for i=1:N
59     uvb=uvb+ (u(i)*v(i));
60 end
61 uvb=uvb/N;
62 M=[ubSq uvb;uvb vbSq];
63 %calculating minimum eigen value of the matrix
64 minIgen=min(abs(eig(M)));
65
66 MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
67 theta=(atan((-MI(1))/MI(2))*180)/pi;
68
69 thetaRad=(theta*pi)/180;
70 %% rotating the signature and passing the new co-ordinates
71 for i=1:N
72     v(i)=(C(2,i)*cos(thetaRad))-(C(1,i)*sin(thetaRad));
73     u(i)=(C(2,i)*sin(thetaRad))+(C(1,i)*cos(thetaRad));
74 end
75 C=[u;v];
76
77 %moving the signature to its original position
78
79 for i=1:N
80     u(i)=round(u(i)+oub-1);
81     v(i)=round(v(i)+ovb-1);

```

2

2.1 Explanation

We used controlled dilation and erosion as basic morphological operations to improve images and pull out features. The approach begins with the preprocessing step of padding the original image based on the given structuring element. This ensures that the structuring element can be properly placed at each pixel location during subsequent operations.

For regulated dilation, we iterate over the original image, placing the reflected structuring element at each pixel. The reflected structuring element accounts for any symmetry present, eliminating the need for modification. At each iteration, we calculate the cardinality of the intersection between the structuring element and the corresponding region in the image. If this count exceeds a predefined strictness parameter (denoted as 's'), we set the pixel value in the resulting image to 1; otherwise, it is set to 0. This process effectively makes the edges of regions in the binary image bigger, and the strictness parameter lets you control how much it grows.

Conversely, regulated erosion involves placing the structuring element at each pixel and finding the intersection pixel count with the complemented original image. Here, the structuring element is not reflected. If the intersection count is below the specified strictness parameter ('s'), the pixel value in the resulting image is set to 1; otherwise, it is set to 0. Regulated erosion is meant to make the edges of areas in the binary image smaller, and the strictness parameter lets you change how the erosion works.

This methodical approach gives a flexible and controlled framework for controlled erosion and dilation, so the morphological operations can be made to fit the features of the input image.

2.2 Python Code

```

1 pip install Pillow
2
3 import cv2
4 import numpy as np
5 from skimage import io
6 from skimage.morphology import binary_dilation, binary_erosion, square,
   rectangle, disk, diamond
7 from google.colab.patches import cv2_imshow
8 from PIL import Image
9
10 #reading the image
11 image = cv2.imread('book1.png')
12
13 # Open an image file
14 input_image_path = '/content/book1.png'
15 output_image_path = '/content/output_image.png'
16 image = Image.open("/content/book1.png")
17
18 new_size = (114*3, 89*3)
19
20 # Resize the image
21 resized_image = image.resize(new_size)
22
23 resized_array = np.array(resized_image)
24
25 cv2_imshow(resized_array)
26 cv2.waitKey(0)
27 cv2.destroyAllWindows()
28
29 gray_image = cv2.cvtColor(resized_array, cv2.COLOR_BGR2GRAY)
30 _, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
31

```

```

32 import numpy as np
33 # Iterate through each element in the array
34 for i in range(binary_image.shape[0]):
35     for j in range(binary_image.shape[1]):
36         # Check if the pixel value is 255
37         if binary_image[i, j] == 255:
38             # Change it to 1
39             binary_image[i, j] = 1
40 # print(binary_image)
41
42
43 def pad_binary_image(image, top_padding, bottom_padding, left_padding,
44                      right_padding):
45     original_height, original_width = image.shape
46
47     #calculating the new dimensions with padding
48     padded_height = original_height + top_padding + bottom_padding
49     padded_width = original_width + left_padding + right_padding
50
51     #creating a new blank image filled with zeros
52     padded_image = np.zeros((padded_height, padded_width), dtype=np.uint8)
53
54     #copying the original binary image to the center of the padded image
55     padded_image[top_padding:top_padding + original_height,
56                  left_padding:left_padding + original_width] = image
57     # print("Padded image is \n")
58     # print(padded_image)
59     return padded_image
60
61 def count_intersection_pixels(image, struc_ele, x_offset, y_offset):
62     #ensuring that the structuring element fits within the image
63     if y_offset + struc_ele.shape[1]-1 > image.shape[1] or x_offset +
64         struc_ele.shape[0]-1 > image.shape[0]:
65         raise ValueError("Structuring element does not fit within the image at
66             the specified coordinates.")
67
68     #creating a region of interest (ROI) in the image
69     roi = image[x_offset : x_offset + struc_ele.shape[0], y_offset : y_offset
70                 + struc_ele.shape[1]]
71
72     #counting the number of intersection pixels (pixels with a value of 1 in
73     #both image and the structuring element)
74     intersection_count = np.sum(roi & struc_ele)
75     #print("Intersection count is \n")
76     #print(intersection_count)
77     return intersection_count
78
79 #updated dilation function
80 def updated_dilation(image, s, struc_ele):
81     struc_height, struc_width = struc_ele.shape
82     struc_x = struc_width//2
83     struc_y = struc_height//2
84     original_height, original_width = image.shape
85
86     resulting_image = np.zeros((original_height, original_width),
87                                dtype=np.uint8)
88
89     padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
90     # we run on the image and check the cardinality of the 1 pixels when
91     # structuring element is placed at each pixel in the image
92     # if this cardinality is greater than or equal to s then we keep it 1 else
93     # 0
94     for x in range(image.shape[0]): # height

```

```

86         for y in range(image.shape[1]): # width
87             intersection_count = count_intersection_pixels(padded_image,
88                 struc_ele, x, y)
89             if intersection_count >= s:
90                 resulting_image[x, y] = 255
91
92     return resulting_image
93
94 #updated erosion function
95 def updated_erosion(image, s, struc_ele):
96     struc_height, struc_width = struc_ele.shape
97     struc_x = struc_width//2
98     struc_y = struc_height//2
99     original_height, original_width = image.shape
100
101     resulting_image = np.zeros((original_height, original_width),
102         dtype=np.uint8)
103
104     padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
105     compliment_image = complemented_image =
106         np.logical_not(padded_image).astype(int)
107     # we run on the image and check the cardinality of the 1 pixels when
108     structuring element is placed at each pixel in the image
109     # if this cardinality is greater than or equal to s then we keep it 1 else
110     0
111     for x in range(image.shape[0]): # height
112         for y in range(image.shape[1]): # width
113             intersection_count = count_intersection_pixels(compliment_image,
114                 struc_ele, x, y)
115             if intersection_count <= s:
116                 resulting_image[x, y] = 255
117
118     return resulting_image
119
120 # normal dilation function
121 def normal_dilation(image, s, struc_ele):
122     struc_height, struc_width = struc_ele.shape
123     struc_x = struc_width//2
124     struc_y = struc_height//2
125     original_height, original_width = image.shape
126
127     resulting_image = np.zeros((original_height, original_width),
128         dtype=np.uint8)
129
130     padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
131     # we run on the image and check the cardinality of the 1 pixels when
132     structuring element is placed at each pixel in the image
133     # if this cardinality is greater than or equal to s then we keep it 1 else
134     0
135     for x in range(image.shape[0]): # height
136         for y in range(image.shape[1]): # width
137             intersection_count = count_intersection_pixels(padded_image,
138                 struc_ele, x, y)
139             if intersection_count >= 1: # taking intersection greater than
140                 equal to 1
141                 resulting_image[x, y] = 255
142
143     return resulting_image
144
145 # normal erosion function
146 def normal_erosion(image, s, struc_ele):
147     struc_height, struc_width = struc_ele.shape
148     struc_x = struc_width//2

```

```

138     struc_y = struc_height//2
139     original_height, original_width = image.shape
140
141     resulting_image = np.zeros((original_height, original_width),
142                                dtype=np.uint8)
143
144     padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
145     # print(padded_image)
146     compliment_image = complemented_image =
147         np.logical_not(padded_image).astype(int)
148     # print(compliment_image)
149     # we run on the image and check the cardinality of the 1 pixels when
150     # structuring element is placed at each pixel in the image
151     # if this cardinality is greater than or equal to s then we keep it 1 else
152     # 0
153     for x in range(image.shape[0]): # height
154         for y in range(image.shape[1]): # width
155             intersection_count = count_intersection_pixels(compliment_image,
156                                                            struc_ele, x, y)
157             if intersection_count == 0:
158                 resulting_image[x, y] = 255
159
160     return resulting_image
161
162 import matplotlib.pyplot as plt
163
164 def display_images(result1, result2, result3, result4, custom_title):
165     # Create a figure with 2 rows and 2 columns
166     fig, axs = plt.subplots(2, 2, figsize=(10, 10))
167
168     # Display images in each subplot with custom titles
169     axs[0, 0].imshow(result1, cmap='gray')
170     axs[0, 0].set_title(f'Updated Dilation - {custom_title}')
171
172     axs[0, 1].imshow(result3, cmap='gray')
173     axs[0, 1].set_title(f'Normal Dilation - {custom_title}')
174
175     axs[1, 0].imshow(result2, cmap='gray')
176     axs[1, 0].set_title(f'Updated Erosion - {custom_title}')
177
178     axs[1, 1].imshow(result4, cmap='gray')
179     axs[1, 1].set_title(f'Normal Erosion - {custom_title}')
180
181     # Hide the axes labels
182     for ax in axs.flat:
183         ax.label_outer()
184
185     # Show the plot
186     plt.show()
187
188 #square structuring element
189 square_struc_ele = square(5)
190 #rectangle structuring element
191 rectangle_struc_ele = rectangle(5, 3)
192 #circle structuring element
193 circle_struc_ele = disk(3)
194 #diamond structuring element
195 diamond_struc_ele = diamond(3)
196 #plus structuring element
197 plus_struc_ele = np.array([[0, 0, 1, 0, 0],
198                             [0, 0, 1, 0, 0],
199                             [1, 1, 1, 1, 1],
200                             [0, 0, 1, 0, 0],
201                             [0, 0, 1, 0, 0]])

```

```

196         [0, 0, 1, 0, 0]], dtype=bool)
197
198
199 result1 = updated_dilation(binary_image, 20, square_struc_ele)
200 result2 = updated_erosion(binary_image, 20, square_struc_ele)
201 result3 = normal_dilation(binary_image, 1, square_struc_ele)
202 result4 = normal_erosion(binary_image, 20, square_struc_ele)
203
204 display_images(result1, result2, result3, result4, "Square Structuring
    Element")
205
206 result1 = updated_dilation(binary_image, 15, rectangle_struc_ele)
207 result2 = updated_erosion(binary_image, 10, rectangle_struc_ele)
208 result3 = normal_dilation(binary_image, 1, rectangle_struc_ele)
209 result4 = normal_erosion(binary_image, 15, rectangle_struc_ele)
210
211 display_images(result1, result2, result3, result4, "Rectangle Structuring
    Element")
212
213 result1 = updated_dilation(binary_image, 20, circle_struc_ele)
214 result2 = updated_erosion(binary_image, 20, circle_struc_ele)
215 result3 = normal_dilation(binary_image, 1, circle_struc_ele)
216 result4 = normal_erosion(binary_image, 40, circle_struc_ele)
217
218
219 display_images(result1, result2, result3, result4, "Circular Structuring
    Element")
220
221 circle_struc_ele.size
222
223 result1 = updated_dilation(binary_image, 20, diamond_struc_ele)
224 result2 = updated_erosion(binary_image, 20, diamond_struc_ele)
225 result3 = normal_dilation(binary_image, 1, diamond_struc_ele)
226 result4 = normal_erosion(binary_image, 40, diamond_struc_ele)
227
228 diamond_struc_ele.size
229
230 display_images(result1, result2, result3, result4, "Diamond Structuring
    Element")
231
232
233 result1 = updated_dilation(binary_image, 7, plus_struc_ele)
234 result2 = updated_erosion(binary_image, 7, plus_struc_ele)
235 result3 = normal_dilation(binary_image, 1, plus_struc_ele)
236 result4 = normal_erosion(binary_image, 25, plus_struc_ele)
237
238 plus_struc_ele.size
239
240 display_images(result1, result2, result3, result4, "Plus Structuring Element")
241
242 def dilate_then_erode(input_image, s1, s2, structuring_element):
243     # Perform dilation
244     dilated_image = updated_dilation(input_image, s1, structuring_element)
245
246     # Perform erosion on the dilated image
247     final_image = updated_erosion(dilated_image, s2, structuring_element)
248
249     return final_image
250
251 result1 = dilate_then_erode(binary_image, 20,20, square_struc_ele)
252 result2 = dilate_then_erode(binary_image, 15,10, rectangle_struc_ele)
253 result3 = dilate_then_erode(binary_image, 20,20, circle_struc_ele)
254 result4 = dilate_then_erode(binary_image, 20, 20, diamond_struc_ele)

```

```

255 result5 = dilate_then_erode(binary_image, 7,7, plus_struc_ele)
256
257
258 # Create a figure with 2 rows and 3 columns
259 fig, axs = plt.subplots(2, 3, figsize=(15, 10))
260
261 # Display images in each subplot with custom titles
262 axs[0, 0].imshow(result1, cmap='gray')
263 axs[0, 0].set_title('Square')
264
265 axs[0, 1].imshow(result2, cmap='gray')
266 axs[0, 1].set_title('Rectangle')
267
268 axs[0, 2].imshow(result3, cmap='gray')
269 axs[0, 2].set_title('Circle')
270
271 axs[1, 0].imshow(result4, cmap='gray')
272 axs[1, 0].set_title('Diamond')
273
274 axs[1, 1].imshow(result5, cmap='gray')
275 axs[1, 1].set_title('Plus')
276
277 # Hide the axes labels
278 for ax in axs.flat:
279     ax.label_outer()
280
281 # Hide the last subplot (if necessary, adjust this based on your layout)
282 axs[1, 2].axis('off')
283
284 # Show the plot
285 plt.show()

```

2.3 Results

We have improved the performance of the algorithm by replacing regular operations with regulated ones and fine-tuning a parameter called "strictness" based on specific criteria. This optimization, achieved by using regulated operations strategically, has upgraded the effectiveness of the morphological algorithm. By swapping out certain parts with regulated operations and adjusting them appropriately, we have obtained better results.


```
SigImage=imread('sig.jpg'); % Load the image file and store it as the variable SigImage.  
figure,imshow(SigImage);
```

A handwritten signature in black ink on a white background. The signature is written in a cursive style and reads "Clara King". The letters are fluidly connected, with a prominent loop at the start of the first name and a long, sweeping tail on the last name.

```
I2=imresize(SigImage,[128 ,128]);  
figure,imshow(I2);
```

A handwritten signature in black ink on a white background. The signature is written in a cursive style and reads "Chandray".

```
I3=rgb2gray(I2);  
I3=im2double(I3);  
I3=imbinarize(I3);  
I3 = bwmorph(~I3, 'thin', inf);  
I3=~I3;  
figure,imshow(I3);
```

%converting image to black and white
%thining the image



```
i=1;
k=1;

while i<=128
    j=1;
    while j<=128
        if I3(i, j)==0
            u(k)=i;
            v(k)=j;
            k=k+1;
            I3(i, j)=1;
        end
        j=j+1;
    end
    i=i+1;
end

C=[u;v];%the curve of the signature
N=k-1;%the number of pixels in the signature
```

```

oub=0;
for i=1:N
    oub= oub+ C(1,i);
end
oub=oub/N; %the original x co-ordinate center of mass of the image
ovb=0;
for i=1:N
    ovb= ovb+ C(2,i);
end
ovb=ovb/N; %the original y co-ordinate center of mass of the image

```

```

%moving the signature to the origin

```

```

for i=1:N
    u(i)=u(i)-oub+1;
    v(i)=v(i)-ovb+1;
end
% the new curve of the signature
C=[u;v];

```

```

ub=sum(C(1,:))/N;
vb=sum(C(2,:))/N;
ubSq=sum((C(1, :)-ub).^2)/N;
vbSq=sum((C(2, :)-vb).^2)/N;

```

```

uvb=0;
for i=1:N
    uvb=uvb+ (u(i)*v(i));
end
uvb=uvb/N;
M=[ubSq uvb;uvb vbSq];
%calculating minimum eigen value of the matrix
minIgen=min(abs(eig(M)));

MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
theta=(atan((-MI(1))/MI(2))*180)/pi;

thetaRad=(theta*pi)/180;
%% rotating the signature and passing the new co-ordinates
for i=1:N
    v(i)=(C(2,i)*cos(thetaRad))-(C(1,i)*sin(thetaRad));
    u(i)=(C(2,i)*sin(thetaRad))+(C(1,i)*cos(thetaRad));
end
C=[u;v];

%moving the signature to its original position

for i=1:N
    u(i)=round(u(i)+oub-1);
    v(i)=round(v(i)+ovb-1);
end

```


vdxsvf4ph

November 10, 2023

```
[223]: pip install Pillow
```

Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (9.4.0)

```
[224]: import cv2
import numpy as np
from skimage import io
from skimage.morphology import binary_dilation, binary_erosion, square,
    rectangle, disk, diamond
from google.colab.patches import cv2_imshow
from PIL import Image
```

```
[225]: #reading the image
image = cv2.imread('book1.png')
```

```
[226]: # Open an image file
input_image_path = '/content/book1.png'
output_image_path = '/content/output_image.png'
image = Image.open("/content/book1.png")

new_size = (114*3, 89*3)

# Resize the image
resized_image = image.resize(new_size)

resized_array = np.array(resized_image)

cv2_imshow(resized_array)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
[227]: gray_image = cv2.cvtColor(resized_array, cv2.COLOR_BGR2GRAY)
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
```

```
[228]: import numpy as np
# Iterate through each element in the array
for i in range(binary_image.shape[0]):
    for j in range(binary_image.shape[1]):
        # Check if the pixel value is 255
        if binary_image[i, j] == 255:
            # Change it to 1
            binary_image[i, j] = 1
# print(binary_image)
```

```
[229]: def pad_binary_image(image, top_padding, bottom_padding, left_padding,
    ↪right_padding):
    original_height, original_width = image.shape

    #calculating the new dimensions with padding
    padded_height = original_height + top_padding + bottom_padding
    padded_width = original_width + left_padding + right_padding

    #creating a new blank image filled with zeros
    padded_image = np.zeros((padded_height, padded_width), dtype=np.uint8)
```

```

    #copying the original binary image to the center of the padded image
    padded_image[top_padding:top_padding + original_height, left_padding:
↳left_padding + original_width] = image
    # print("Padded iamge is \n")
    # print(padded_image)
    return padded_image

```

```

[230]: def count_intersection_pixels(image, struc_ele, x_offset, y_offset):
    #ensuring that the structuring element fits within the image
    if y_offset + struc_ele.shape[1]-1 > image.shape[1] or x_offset + struc_ele.
↳shape[0]-1 > image.shape[0]:
        raise ValueError("Structuring element does not fit within the image at
↳the specified coordinates.")

    #creating a region of interest (ROI) in the image
    roi = image[x_offset : x_offset + struc_ele.shape[0], y_offset : y_offset +
↳struc_ele.shape[1]]

    #counting the number of intersection pixels (pixels with a value of 1 in
↳both image and the structuring element)
    intersection_count = np.sum(roi & struc_ele)
    #print("Intersection count is \n")
    #print(intersection_count)
    return intersection_count

```

```

[231]: #updated dilation function
def updated_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
↳uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when
↳structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,
↳struc_ele, x, y)
            if intersection_count >= s:
                resulting_image[x, y] = 255

```



```
return resulting_image
```

```
[232]: #updated erosion function
def updated_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
    ↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    compliment_image = complemented_image = np.logical_not(padded_image).
    ↪astype(int)
    # we run on the image and check the cardinality of the 1 pixels when
    ↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,
    ↪struc_ele, x, y)
            if intersection_count <= s:
                resulting_image[x, y] = 255

    return resulting_image
```

```
[233]: # normal dilation function
def normal_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
    ↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when
    ↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,
    ↪struc_ele, x, y)
```

```

        if intersection_count >= 1: # taking intersection greater than
↪equal to 1
            resulting_image[x, y] = 255

    return resulting_image

```

```

[234]: # normal erosion function
def normal_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # print(padded_image)
    compliment_image = complemented_image = np.logical_not(padded_image).
↪astype(int)
    # print(compliment_image)
    # we run on the image and check the cardinality of the 1 pixels when
↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,
↪struc_ele, x, y)
            if intersection_count == 0:
                resulting_image[x, y] = 255

    return resulting_image

```

```

[235]: import matplotlib.pyplot as plt

def display_images(result1, result2, result3, result4, custom_title):
    # Create a figure with 2 rows and 2 columns
    fig, axs = plt.subplots(2, 2, figsize=(10, 10))

    # Display images in each subplot with custom titles
    axs[0, 0].imshow(result1, cmap='gray')
    axs[0, 0].set_title(f'Updated Dilation - {custom_title}')

    axs[0, 1].imshow(result3, cmap='gray')
    axs[0, 1].set_title(f'Normal Dilation - {custom_title}')

```

```

    axs[1, 0].imshow(result2, cmap='gray')
    axs[1, 0].set_title(f'Updated Erosion - {custom_title}')

    axs[1, 1].imshow(result4, cmap='gray')
    axs[1, 1].set_title(f'Normal Erosion - {custom_title}')

    # Hide the axes labels
    for ax in axs.flat:
        ax.label_outer()

    # Show the plot
    plt.show()

```

```

[236]: #square structuring element
square_struc_ele = square(5)
#rectangle structuring element
rectangle_struc_ele = rectangle(5, 3)
#circle structuring element
circle_struc_ele = disk(3)
#diamond structuring element
diamond_struc_ele = diamond(3)
#plus structuring element
plus_struc_ele = np.array([[0, 0, 1, 0, 0],
                           [0, 0, 1, 0, 0],
                           [1, 1, 1, 1, 1],
                           [0, 0, 1, 0, 0],
                           [0, 0, 1, 0, 0]], dtype=bool)

```

```

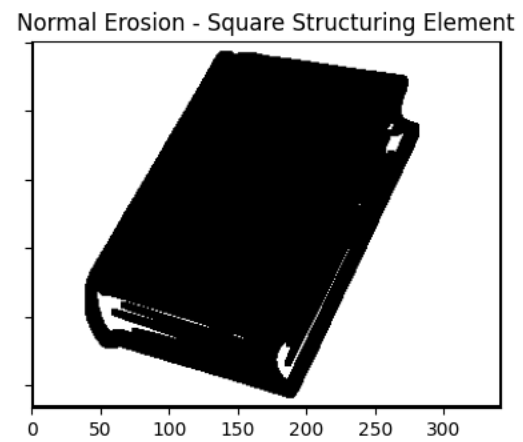
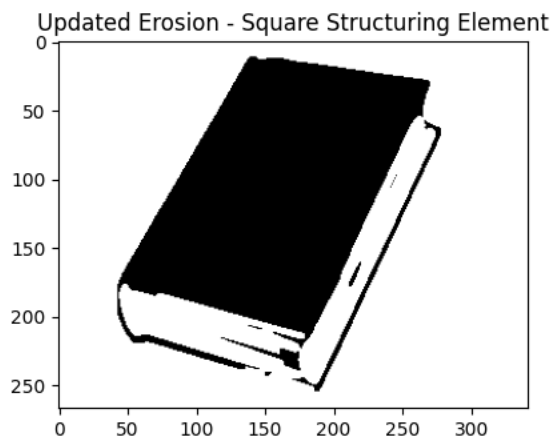
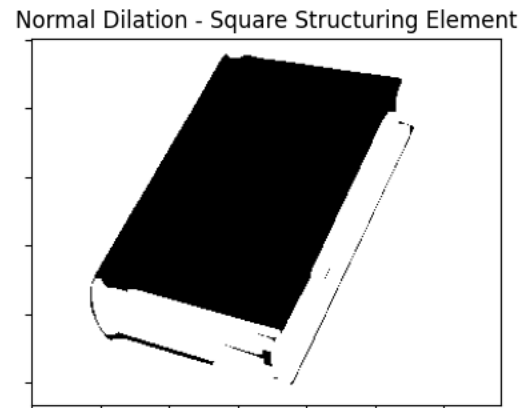
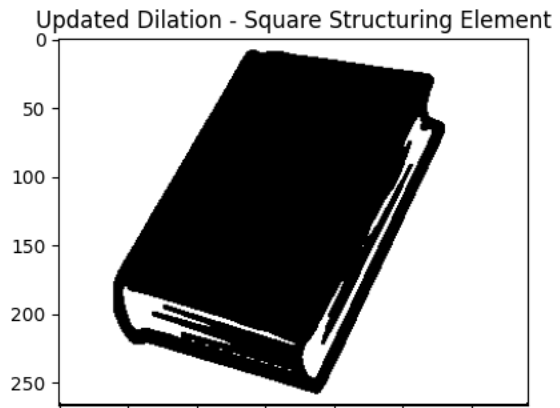
[237]: result1 = updated_dilation(binary_image, 20, square_struc_ele)
result2 = updated_erosion(binary_image, 20, square_struc_ele)
result3 = normal_dilation(binary_image, 1, square_struc_ele)
result4 = normal_erosion(binary_image, 20, square_struc_ele)

```

```

[238]: display_images(result1, result2, result3, result4, "Square Structuring Element")

```



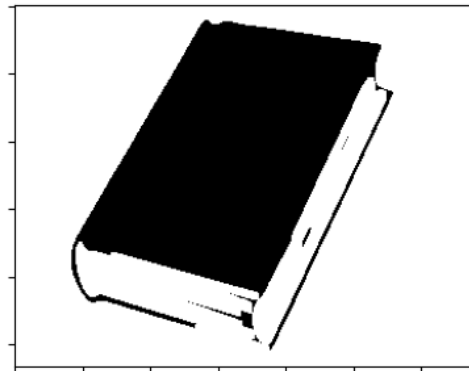
```
[239]: result1 = updated_dilation(binary_image, 15, rectangle_struc_ele)
result2 = updated_erosion(binary_image, 10, rectangle_struc_ele)
result3 = normal_dilation(binary_image, 1, rectangle_struc_ele)
result4 = normal_erosion(binary_image, 15, rectangle_struc_ele)
```

```
[240]: display_images(result1, result2, result3, result4, "Rectangle Structuring_
↪Element")
```

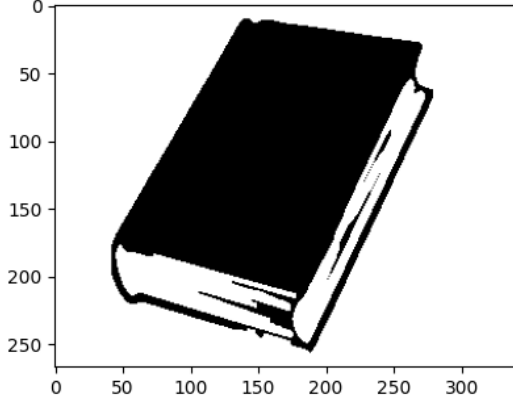
Updated Dilation - Rectangle Structuring Element



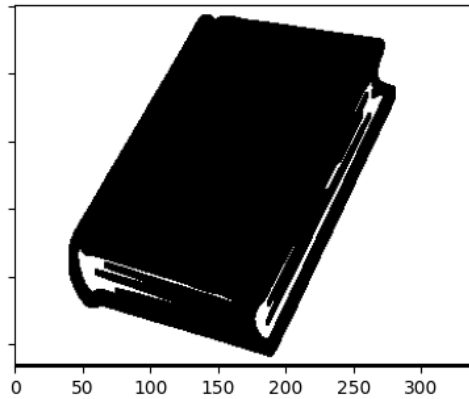
Normal Dilation - Rectangle Structuring Element



Updated Erosion - Rectangle Structuring Element

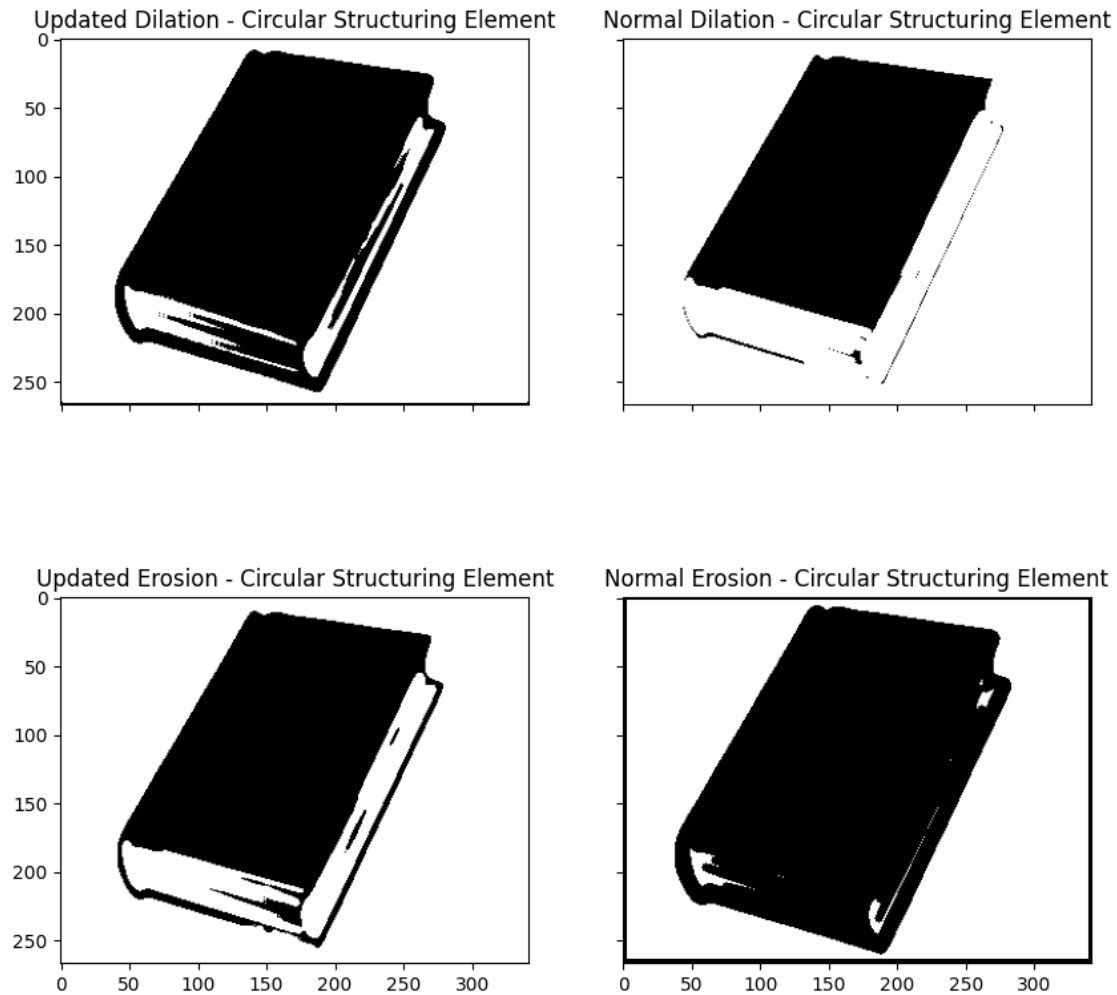


Normal Erosion - Rectangle Structuring Element



```
[241]: result1 = updated_dilation(binary_image, 20, circle_struc_ele)
result2 = updated_erosion(binary_image, 20, circle_struc_ele)
result3 = normal_dilation(binary_image, 1, circle_struc_ele)
result4 = normal_erosion(binary_image, 40, circle_struc_ele)
```

```
[242]: display_images(result1, result2, result3, result4, "Circular Structuring_
      ↪Element")
```



```
[243]: circle_struc_ele.size
```

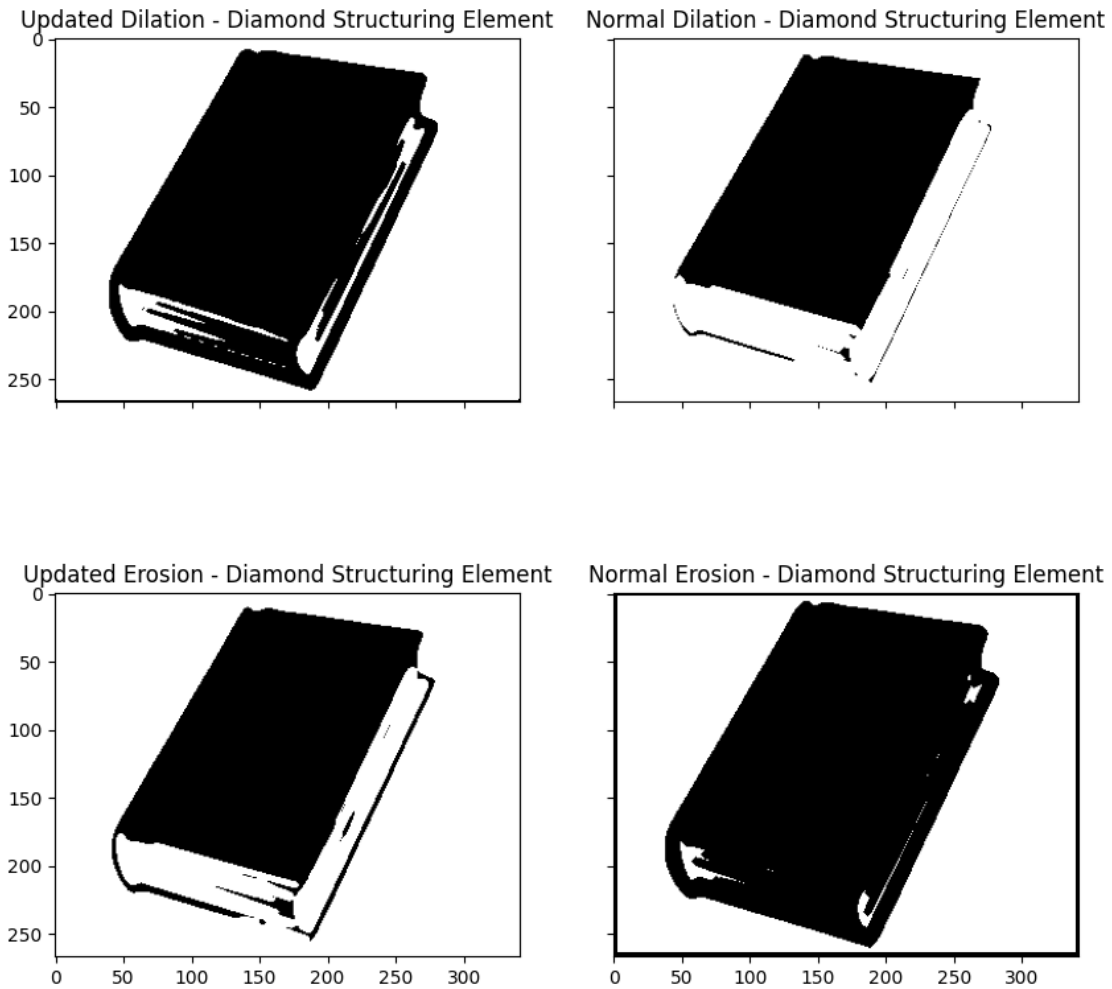
```
[243]: 49
```

```
[244]: result1 = updated_dilation(binary_image, 20, diamond_struc_ele)
result2 = updated_erosion(binary_image, 20, diamond_struc_ele)
result3 = normal_dilation(binary_image, 1, diamond_struc_ele)
result4 = normal_erosion(binary_image, 40, diamond_struc_ele)
```

```
[245]: diamond_struc_ele.size
```

```
[245]: 49
```

```
[246]: display_images(result1, result2, result3, result4, "Diamond Structuring_
↳Element")
```

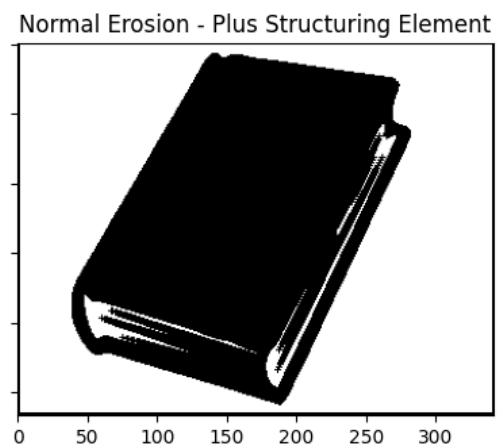
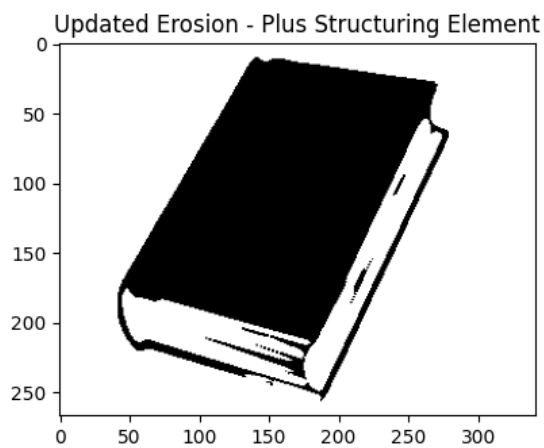
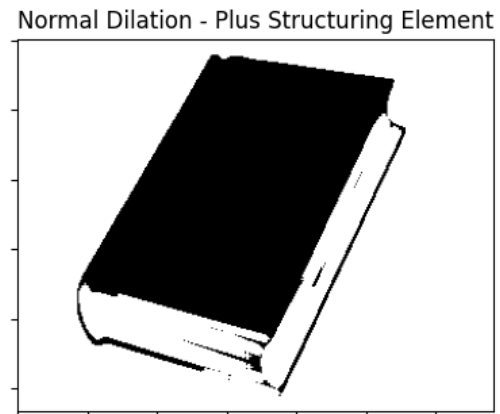
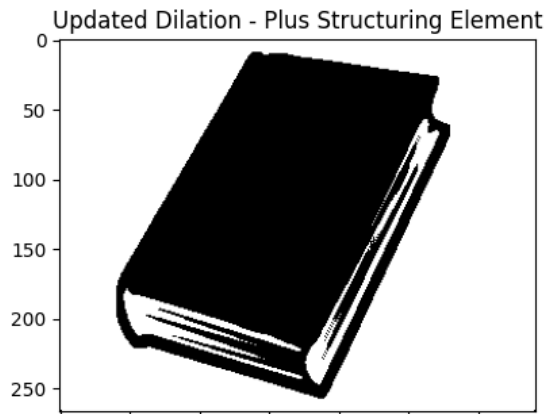


```
[247]: result1 = updated_dilation(binary_image, 7, plus_struc_ele)
result2 = updated_erosion(binary_image, 7, plus_struc_ele)
result3 = normal_dilation(binary_image, 1, plus_struc_ele)
result4 = normal_erosion(binary_image, 25, plus_struc_ele)
```

```
[248]: plus_struc_ele.size
```

```
[248]: 25
```

```
[249]: display_images(result1, result2, result3, result4, "Plus Structuring Element")
```



```
[250]: def dilate_then_erode(input_image, s1, s2, structuring_element):
        # Perform dilation
        dilated_image = updated_dilation(input_image, s1, structuring_element)

        # Perform erosion on the dilated image
        final_image = updated_erosion(dilated_image, s2, structuring_element)

        return final_image
```

```
[251]: result1 = dilate_then_erode(binary_image, 20,20, square_struc_ele)
result2 = dilate_then_erode(binary_image, 15,10, rectangle_struc_ele)
result3 = dilate_then_erode(binary_image, 20,20, circle_struc_ele)
result4 = dilate_then_erode(binary_image, 20, 20, diamond_struc_ele)
result5 = dilate_then_erode(binary_image, 7,7, plus_struc_ele)
```



```

[252]: # Create a figure with 2 rows and 3 columns
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Display images in each subplot with custom titles
axs[0, 0].imshow(result1, cmap='gray')
axs[0, 0].set_title('Square')

axs[0, 1].imshow(result2, cmap='gray')
axs[0, 1].set_title('Rectangle')

axs[0, 2].imshow(result3, cmap='gray')
axs[0, 2].set_title('Circle')

axs[1, 0].imshow(result4, cmap='gray')
axs[1, 0].set_title('Diamond')

axs[1, 1].imshow(result5, cmap='gray')
axs[1, 1].set_title('Plus')

# Hide the axes labels
for ax in axs.flat:
    ax.label_outer()

# Hide the last subplot (if necessary, adjust this based on your layout)
axs[1, 2].axis('off')

# Show the plot
plt.show()

```

