**קורס עיבוד תמונה פרוייקט סוף**

**מגישים:**

**שם:** דן אברין **ת.ז.** 204358394

**שם:** דניאל קוצ'ין **ת.ז.** 314256827

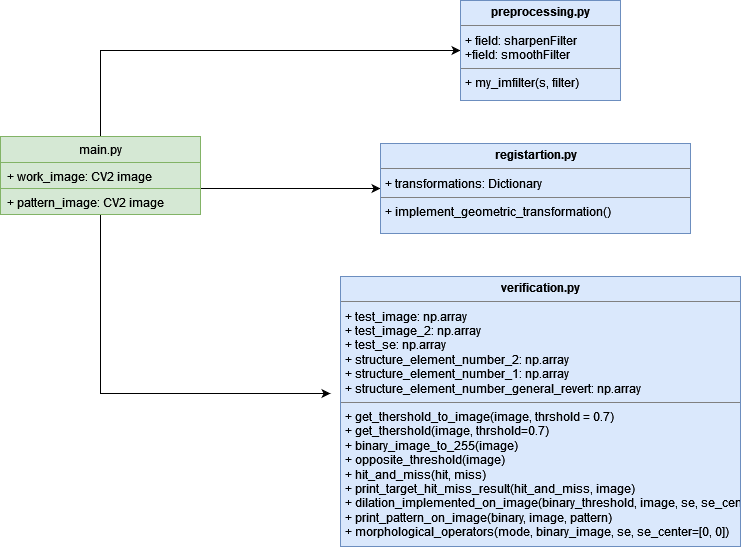
**בפרוייקט זה בחרנו לעבוד על תמונה מקורית שלנו:**

****

אותה תמונה בשחור לבן:



**BLOCK DIAGRAM**

****

1. **Image Loading:**

תחילה נטען ונפתח את תמונת העבודה וpattern שבו נשתמש בהמשך:



### 0. -------- Load Image --------  
work\_image = cv2.imread('image\_resources/sample.png', 0) # open as gray picture  
pattern\_image = cv2.imread('image\_resources/pattern.jpg', 0) # open as gray picture  
cv2.imwrite('image\_output/original\_bw.png', work\_image) # write bw image to file

1. **Pre-Processing:**

בשלב הpre processing העברנו על התמונה על פי הצורך פילטרים של חידוד או החלקה:

### 1. -------- PreProcessing --------  
result\_sharp\_work\_image = preprocessing.my\_imfilter(work\_image, preprocessing.shapreningFilter) # sharpen image  
cv2.imwrite('image\_output/PreProcessing/sharpenResult\_{}.png'.format(time.time()), result\_sharp\_work\_image)  
result\_smooth\_work\_image = preprocessing.my\_imfilter(work\_image, preprocessing.smoothFilter) # smooth image  
cv2.imwrite('image\_output/PreProcessing/smoothResult\_{}.png'.format(time.time()), result\_smooth\_work\_image)

תוצאת פילטר חידוד:

shapreningFilter = np.array([[-1, -1,-1],[-1,8,-1],[-1,-1,-1]]) # Laplacian



תוצאת פילטר החלקה:

smoothFilter = np.array([[1,1,1],[1,1,1],[1,1,1]]) # Average of the around 1/9 \* sums



להלן הקוד שמבצע את פעולות החידוד וההחלקה:

**הפונקציה מקבלת תמונה s ופילטר filter ומבצעת את פעולת הפילטר, לבסוף מחזירה תמונת תוצאה לאחר מימוש הפילטר.**

def my\_imfilter (s, filter):  
 *"""  
 This function gets GRAYSCALE image s and filter, and returns image after filter implementation* ***:param*** *s: given image* ***:param*** *filter: given filter* ***:return****: resultImage or resultFilter, according to given filter  
 """* filterSizeRows, filterSizeCols = filter.shape  
 imageSizeRows, imageSizeCols = s.shape  
  
 paddingDiff=filterSizeRows-1  
 resultFilter = np.zeros((imageSizeRows, imageSizeCols), dtype=np.int32) # image to be returned at the end of process  
 resultImage = np.zeros((imageSizeRows, imageSizeCols), dtype=np.int32) # image to be returned at the end of process  
 filterType = None # will determine logic later  
 filterDictionary = {'sharpening':[np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]),  
 np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])], 'smoothing':[np.array([[1,1,1],[1,1,1],[1,1,1]])]}  
  
 # determing current filter, for logic to apply later  
 for filterFamily in filterDictionary.keys():  
 for currentFilter in filterDictionary[filterFamily]:  
 if np.array\_equal(currentFilter, filter, equal\_nan=False):  
 filterType = filterFamily  
 break  
  
 # Add padding  
 imageWithPadding = np.zeros([imageSizeRows + paddingDiff, imageSizeCols + paddingDiff], dtype=np.uint8)  
 for i in range(imageSizeRows):  
 for j in range(imageSizeCols):  
 imageWithPadding[i+int(paddingDiff/2), j+int(paddingDiff/2)] = s[i,j]  
  
 # implement the filter:  
 for i in range(imageSizeRows):  
 for j in range(imageSizeCols):  
 pixelCalculation = 0 # for filter calculation later  
 # calculate each pixel in the image after filter  
 for filterRow in range(filterSizeRows):  
 for filterCol in range(filterSizeCols):  
 pixelCalculation += imageWithPadding[i+filterRow, j+filterCol] \  
 \* filter[filterRow, filterCol]  
  
 if filterType == 'smoothing':  
 resultFilter[i, j] = int(pixelCalculation / (filterSizeRows \* filterSizeCols))  
 elif filterType == 'sharpening':  
 resultFilter[i, j] = pixelCalculation  
  
 if filterType == 'smoothing':  
 resultFilter=resultFilter.astype(dtype=np.uint8)  
 return resultFilter # end logic for smoothing filter, image is ready  
  
 if filterType == 'sharpening':  
 resultFilter = resultFilter \* 100.0 / (np.amax(resultFilter)-np.amin(resultFilter)) # normalize  
 resultFilter = resultFilter.astype(np.int32)  
 resultImage = s + resultFilter # add filter to original image  
 resultImage = abs(resultImage) # drop negative values  
 resultImage = resultImage\* 255.0 / np.amax(resultImage) # normalize back to 256 scale  
 resultImage = resultImage.astype(np.uint8)  
 return resultImage

1. **Geometric Operators:**

הקוד הבא רץ על כל האופרטורים הגאומטריים שיש בפרוייקט ומבצע עיבוד לתמונת המקור, בהמשך הדו"ח נראה את התוצאות עבור כל אופרטור:

### 2. -------- Registration - Geometric operators --------  
for transformation in registration.transformations.keys():  
 transformatiown\_result = registration.implement\_geometric\_transformation(work\_image, registration.transformations[transformation], transformation)  
 cv2.imwrite('image\_output/geometric\_operators/transformation\_{}\_{}.png'.format(transformation, time.time()), transformatiown\_result)

להלן רשימת האופרטורים השונים שבחרנו לבצע בהם שימוש על התמונה:

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [1, 0, 0] [0, 1, 0] [0, 0, 1] | transformation\_identity |
| [2, 0, 0] [0, 2, 0] [0, 0, 1] | transformation\_scale\_filter\_2x |
| [0.5, 0, 0] [0, 0.5, 0] [0, 0, 1] | transformation\_scale\_filter\_halfx |
| [1, 1, 0] [0, 1, 0] [0, 0, 1] | transformation\_scale\_horizontal\_shear |
| [1, 0, 0] [1, 1, 0] [0, 0, 1] | transformation\_scale\_vertical\_shear |
| [1, 0, 100] [0, 1, 100] [0, 0, 1 ] | transformation\_translation |
| [0.9, -0.7, 0] [0.7, 0.9, 0] [0, 0, 1] | transformation\_rotation |

**פירוט הקוד המבצע את לוגיקת האופרטורים הגאומטריים:**

הפונקציה מקבלת תמונה Image, טרנספורמציה transformation, ותגית עבור הטרנספורמציה transformation\_name ומבצע לוגיקה עבור כל טרנספורמציה.

תחילה נקבעת גודל תמונת היעד על בסיס הטרנספורמציה שתבוצע על התמונה, לכל טרנספומציה תהיה תוצאה בגודל שונה:

def implement\_geometric\_transformation(image, transformation, transformation\_name):  
 *"""  
 This function implements geometric transformation over a given image* ***:param*** *image: given image to execute over the geometric transformation* ***:param*** *transformation: given geometric transformation* ***:param*** *transformation\_name: passed transformation according to transformations dictionary in this module* ***:return****: imageCanvas - image after geometric transformation  
 """* original\_image\_height, original\_image\_width = image.shape # save original image width and height  
 new\_image\_height, new\_image\_width = image.shape # size for the destination canvas  
  
 # determine size:  
 if transformation\_name == "transformation\_translation":  
 new\_image\_height = int(original\_image\_height + transformation[0, 2])  
 new\_image\_width = int(original\_image\_width + transformation[1, 2])  
 elif transformation\_name == "transformation\_identity":  
 new\_image\_height = original\_image\_height  
 new\_image\_width = new\_image\_width  
 elif transformation\_name == "transformation\_scale\_filter\_2x" or transformation\_name == "transformation\_scale\_filter\_halfx":  
 new\_image\_height = int(original\_image\_height \* transformation[0, 0])+1  
 new\_image\_width = int(original\_image\_width \* transformation[1, 1])+1  
 elif transformation\_name == "transformation\_scale\_horizontal\_shear":  
 new\_image\_height = original\_image\_height  
 new\_image\_width = int(original\_image\_width+transformation[0, 1]\*original\_image\_height) + 1  
 elif transformation\_name == "transformation\_scale\_vertical\_shear":  
 new\_image\_height = int(original\_image\_height+transformation[1, 0]\*original\_image\_width) + 1  
 new\_image\_width = original\_image\_width  
 elif transformation\_name == "transformation\_rotation":  
 new\_image\_height = 2\*new\_image\_height  
 new\_image\_width = 2\*new\_image\_width  
 else:  
 new\_image\_height = 4\*new\_image\_height  
 new\_image\_width = 4\*new\_image\_width  
  
 imageCanvas = np.zeros([int(new\_image\_height), int(new\_image\_width)], dtype=np.uint8) # new canvas

לאחר מכן תבוצע לוגיקה והכפלת מטריצות עבור כל x ו y בתמונת המקור ונקבל x' ו y' חדשים לאחר ביצוע הטרנספורמציה, והערך של התמונה המקורית יעבור לתמונה החדשה:

# implement the filter:  
for i in range(original\_image\_height):  
 for j in range(original\_image\_width):  
 current\_pixel\_info = image[i, j]  
 current\_coordinates = np.array([i, j, 0])  
 if transformation\_name == 'transformation\_translation':  
 imageCanvas[int(i + transformation[0, 2]), int(j + transformation[1,2])] = current\_pixel\_info  
 elif transformation\_name == "transformation\_rotation":  
 result\_multiplication = current\_coordinates @ transformation # result 3x3 matrix  
 if result\_multiplication[0] < 0 or result\_multiplication[1] < 0: # out of bounds  
 continue  
 else:  
 imageCanvas[int(result\_multiplication[0]), int(result\_multiplication[1])] = current\_pixel\_info  
 else:  
 result\_multiplication = current\_coordinates @ transformation # result 3x3 matrix  
 imageCanvas[int(result\_multiplication[0]), int(result\_multiplication[1])] = current\_pixel\_info  
  
return imageCanvas

**2.1 טרנפורמציית הזהות:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [1, 0, 0] [0, 1, 0] [0, 0, 1] | transformation\_identity |

**תוצאה:**

****

בתמונה זו היה שימוש באופרטור המורפולוגי של זהות, ולכן קיבלנו אותה תמונת מקור שאיתה עבדנו.

**2.2 טרנפורמציית scaling - במקרה זה הגדלה פי 2:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [2, 0, 0] [0, 2, 0] [0, 0, 1] | transformation\_scale\_filter\_2x |

**תוצאה:**

****

בתמונה הנ"ל קיבלנו תמונה כביכול יותר כהה, אך בפועל מדובר בתמונה גדולה 2X מהתמונה המקורית, ועקב מרווח הפיקסלים במתיחה התמונה נראית יותר כהה,

**ניתן לראות זאות בעזרת ZOOM IN לחלק מן התמונה:**



**2.3 טרנפורמציית scaling - במקרה זה הקטנה פי 0.5:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [0.5, 0, 0] [0, 0.5, 0] [0, 0, 1] | transformation\_scale\_filter\_halfx |

**תוצאה:**



קשה להבין, אך אם נעשה ZOOM IN נוכל לראות את ההבדלים,

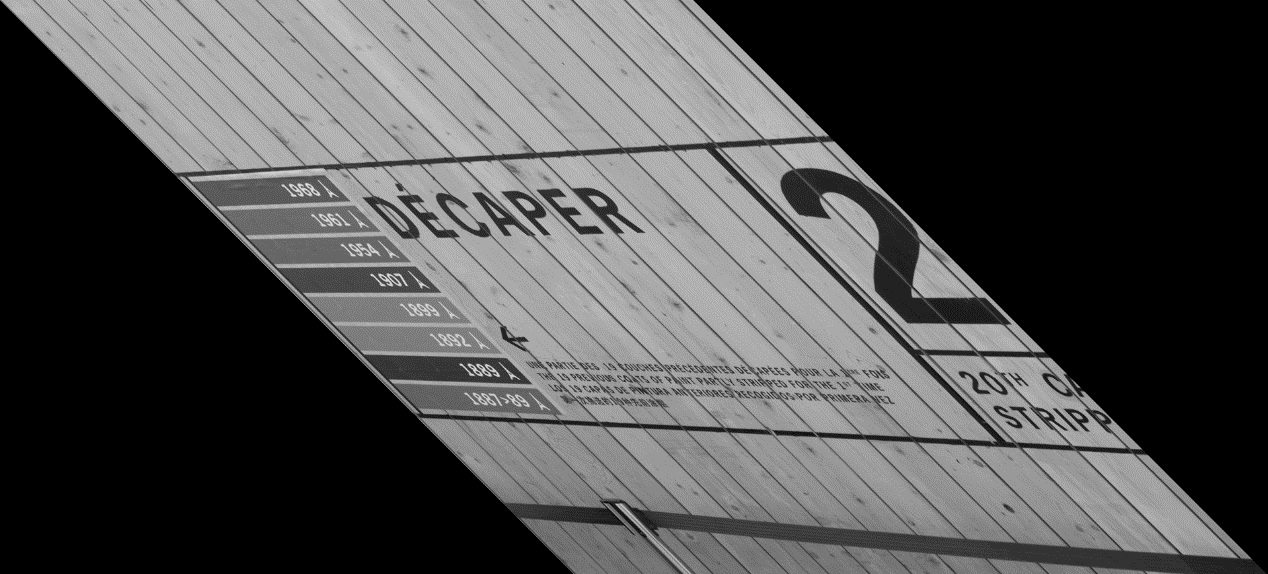
**מימין, התמונה המקורית, משמאל התמונה שעברה הקטנה פי 0.5:**



**2.4 טרנפורמציית Horizontal Shear - מתיחה לרוחב:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [1, 1, 0] [0, 1, 0] [0, 0, 1] | transformation\_scale\_horizontal\_shear |

**התוצאה:**

****

**2.5 טרנפורמציית Vertical Shear - מתיחה לאורך:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [1, 0, 0] [1, 1, 0] [0, 0, 1] | transformation\_scale\_vertical\_shear |

****

**2.6 טרנפורמציית Translation - הזזה – במקרה זה הזזה ימינה ומטה ב100 פיקסלים:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [1, 0, 100] [0, 1, 100] [0, 0, 1 ] | transformation\_translation |

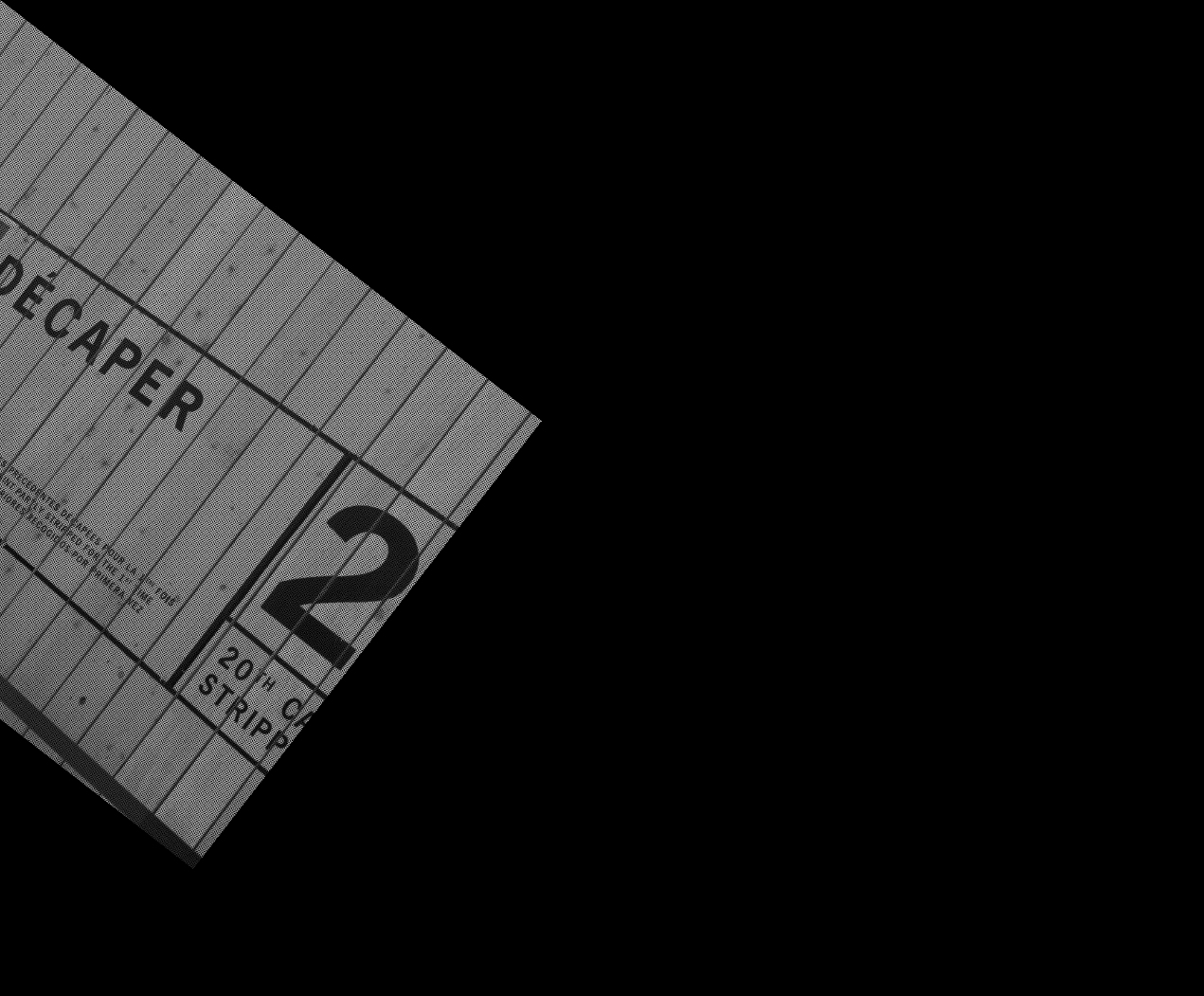
**תוצאה:**



**2.7 טרנפורמציית Rotation – סיבוב – במקרה זה הזזה ימינה ומטה ב100 פיקסלים:**

|  |  |
| --- | --- |
| Transformation | Transformation Name |
| [0.9, -0.7, 0] [0.7, 0.9, 0] [0, 0, 1] | transformation\_rotation |

**תוצאה:**

****

1. **Morpological Operators:**

**3.1 מימוש HIT AND MISS:**

ביצענו תהליך HIT AND MISS כאשר עבדנו על חתיכה מן התמונה שהצגנו למעלה:



* + 1. **מציאת הסיפרה 1:**

יצרנו Structure Element עם הצורה המאונכת הבאה עבור hit:

[0, 0, 0, 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, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 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]

והשתמשנו בStructure Eelement הבא על מנת לבודד אותיות, בתהליך בביצוע miss:

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

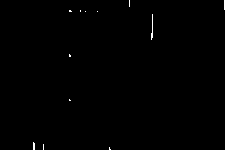
תחילה ביצענו תהליך Threshold עם סף 0.689:

work\_image\_threshold = verification.get\_thershold(work\_image\_sliced, thrshold=0.689) # extract to image file

וקיבלנו את התוצאה הבאה:

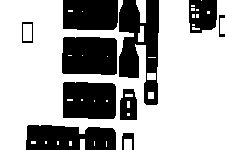


לאחר מכן בוצעה הרצה של תהליך hit עם הse הנ"ל, וקיבלנו את התוצאה הבאה:

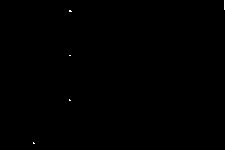


בלבן מסומנות "התאמות" שתואמות את הse שהעברנו.

בוצע היפוך עבור הthreshold שיצרנו וקיבלנו את התוצאה הבאה:



ביצענו סריקה נוספת בעזרת הse השני, וקיבלנו את התוצאה הבאה(Miss):



ביצענו פעולה של AND בין התוצאה של ההרצה הראשונה לתוצאה של ההרצה השניה, והפלט שקיבלנו הוא:



אנו רואים כי ספרות 1 נמצאו וסומנו בתמונה.

* + 1. **דוגמא נוספת - מציאת ספרה 2:**

חזרנו על התהליך אך הפעם שלחנו את הse הבא, המייצג ספרה 2:

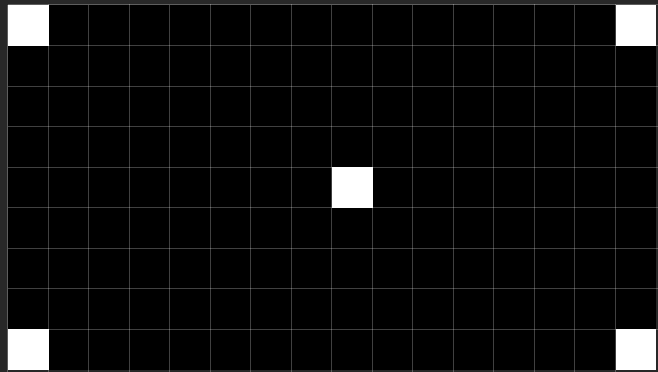
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0]  
[0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0]  
[0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0]  
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0]  
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]  
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]  
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]  
[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]  
[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]  
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]  
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]  
[0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]  
[0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]  
[0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]  
[0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

|  |  |
| --- | --- |
| תיאור פעולה | תוצאה |
| תמונת המקור החתוכה | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\work_image_sliced_1657731893.8458323.png |
| לאחר ביצוע thresold = 0.689 | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\work_image_threshold_to_255_1657731893.8847291.png |
| מציאת Hit | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\work_hit_to_255_1657731894.162883.png |
| היפוך threshold | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\work_miss_to_255_1657731898.1794555.png |
| מציאת Miss | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\work_hit_and_miss_to_255_1657731898.265734.png |
| Miss and hit | C:\Users\dan\Documents\imageProcessingProject\image_output\morpological_operators\final_result_image_1657731898.3106146.png |

**3.2 מימוש Dilation:**

מימשנו אופרטור מורפולוגי של Dilation , להלן דוגמאת הרצה על binary test image הבא:

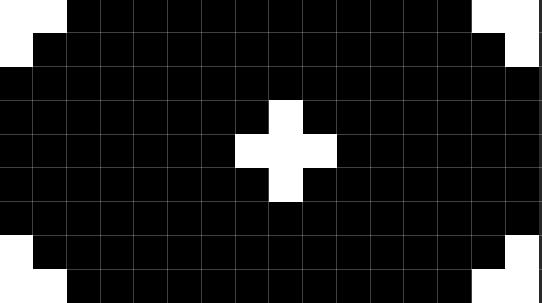
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]



בוצע תהליך Dilation באמצעות הStructure Element הבא:

[0, 1, 0]  
[1, 1, 1]  
[0, 1, 0]

תוצאה לאחר ביצוע התהליך:



ניתן לראות כי בכל מקום בו הופיע 1 (לבן) בתמונה המקורית, נצבעו מסביב פיקסלים בצורת הStrcuture Element שנשלח אליו.