

קורס עיבוד תמונה

פרוייקט סוף

מגישים:

שם: דן אברין ת.ז. 204358394

שם: דניאל קוצ'ין ת.ז. 314256827

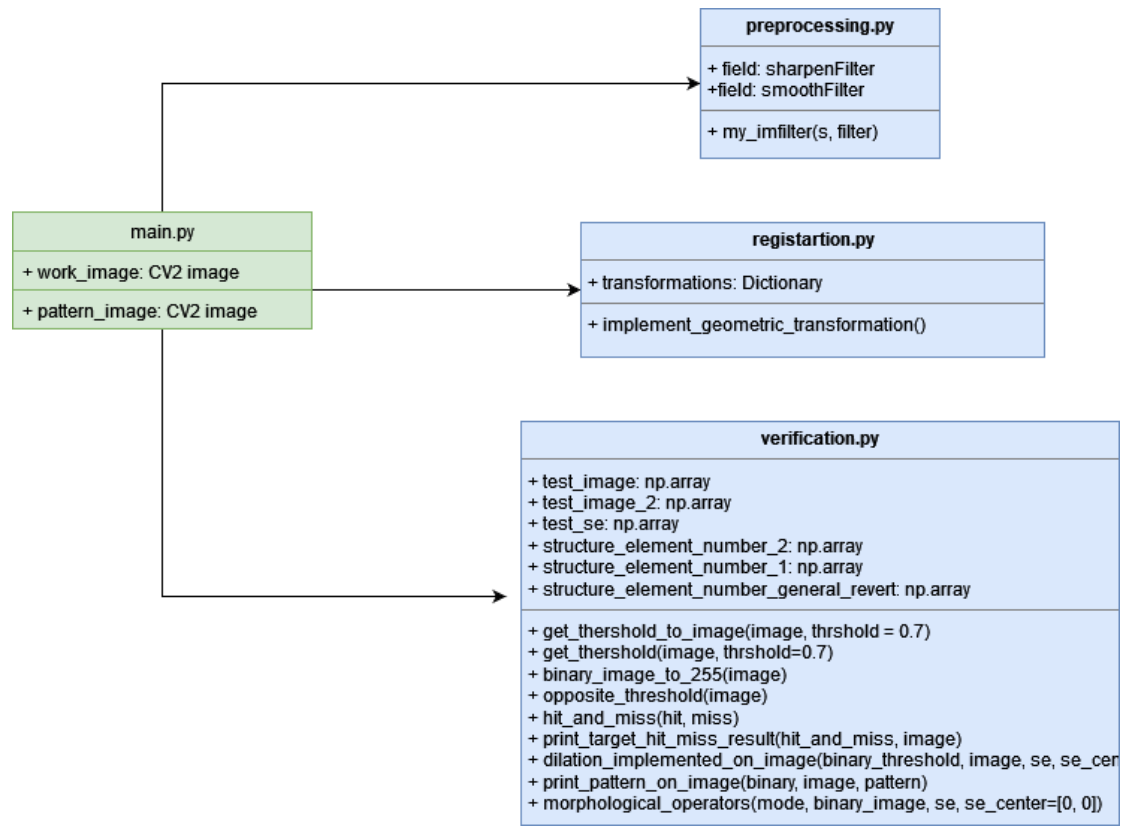
בפרוייקט זה בחרנו לעבוד על תמונה מקורית שלנו:



אותה תמונה בשחור לבן:



BLOCK DIAGRAM



:Image Loading .0

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
```

:Pre-Processing .1

בשלב ה pre 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]])])

    # determining 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

```

:Geometric Operators .2

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():
    transformation_result =
registration.implement_geometric_transformation(work_image,
registration.transformations[transformation], transformation)

cv2.imwrite('image_output/geometric_operators/transformation_{ }_{ }.png'.format(transformation, time.time()), transformation_result)
```

להלן רשימת האופרטורים השונים שבחרנו לבצע בהם שימוש על התמונה:

Transformation Name	Transformation
transformation_identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_scale_filter_2x	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_scale_filter_halfx	$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_scale_horizontal_shear	$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_scale_vertical_shear	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_translation	$\begin{bmatrix} 1 & 0 & 100 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix}$
transformation_rotation	$\begin{bmatrix} 0.9 & -0.7 & 0 \\ 0.7 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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 טרנספורמציה 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 1x3 matrix
            imageCanvas[int(result_multiplication[0]),
int(result_multiplication[1])] = current_pixel_info
return imageCanvas
```

2.1 טרנספורמציות זהות:

Transformation Name	Transformation
transformation identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

תוצאה:

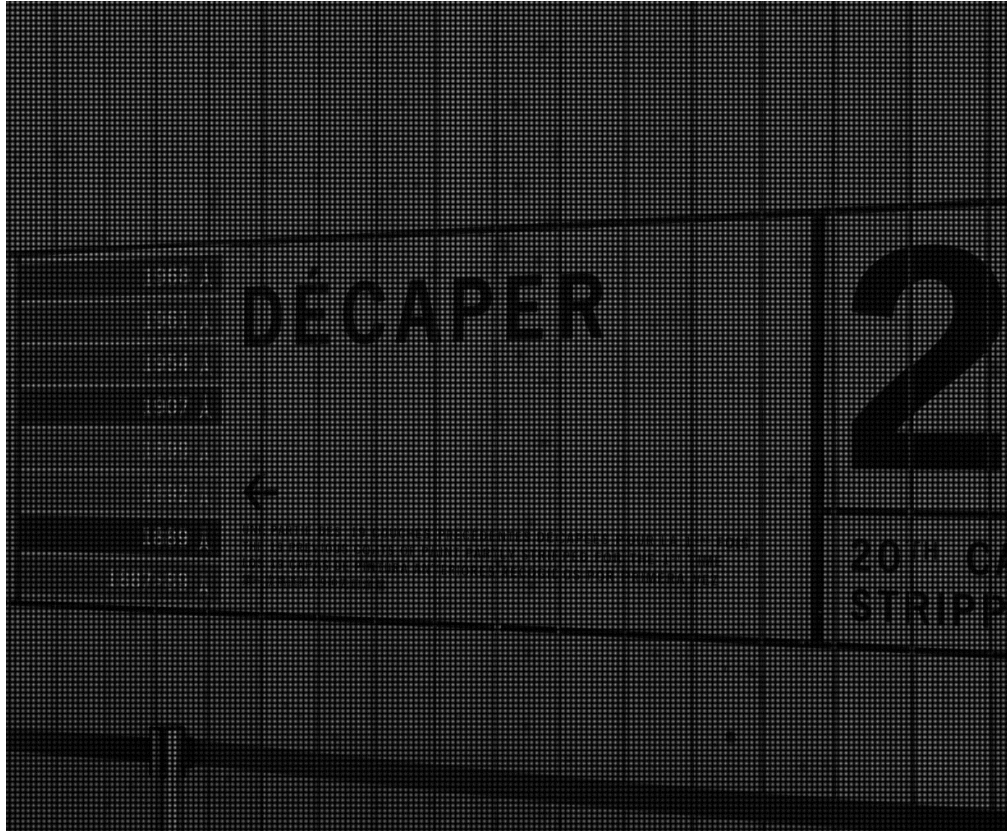


בתמונה זו היה שימוש באופרטור המורפולוגי של זהות, ולכן קיבלנו אותה תמונת מקור שאיתה עבדנו.

2.2 טרנפורמציית scaling - במקרה זה הגדלה פי 2:

Transformation Name	Transformation
transformation scale filter 2x	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

תוצאה:



בתמונה הנ"ל קיבלנו תמונה כביכול יותר כהה, אך בפועל מדובר בתמונה גדולה X2 מהתמונה המקורית, ועקב מרווח הפיקסלים במתיחה התמונה נראית יותר כהה, **ניתן לראות זאת בעזרת ZOOM IN לחלק מן התמונה:**



2.3 טרנפורמציות scaling - במקרה זה הקטנה פי 0.5:

Transformation Name	Transformation
transformation scale filter halfx	$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

תוצאה:



קשה להבחין כאן בהבדל ולכן נעשה ZOOM IN נוכל לראות את ההבדלים, מימין, התמונה המקורית, משמאל התמונה שעברה הקטנה פי 0.5:



2.4 טרנפורמציית Horizontal Shear - מתיחה לרוחב:

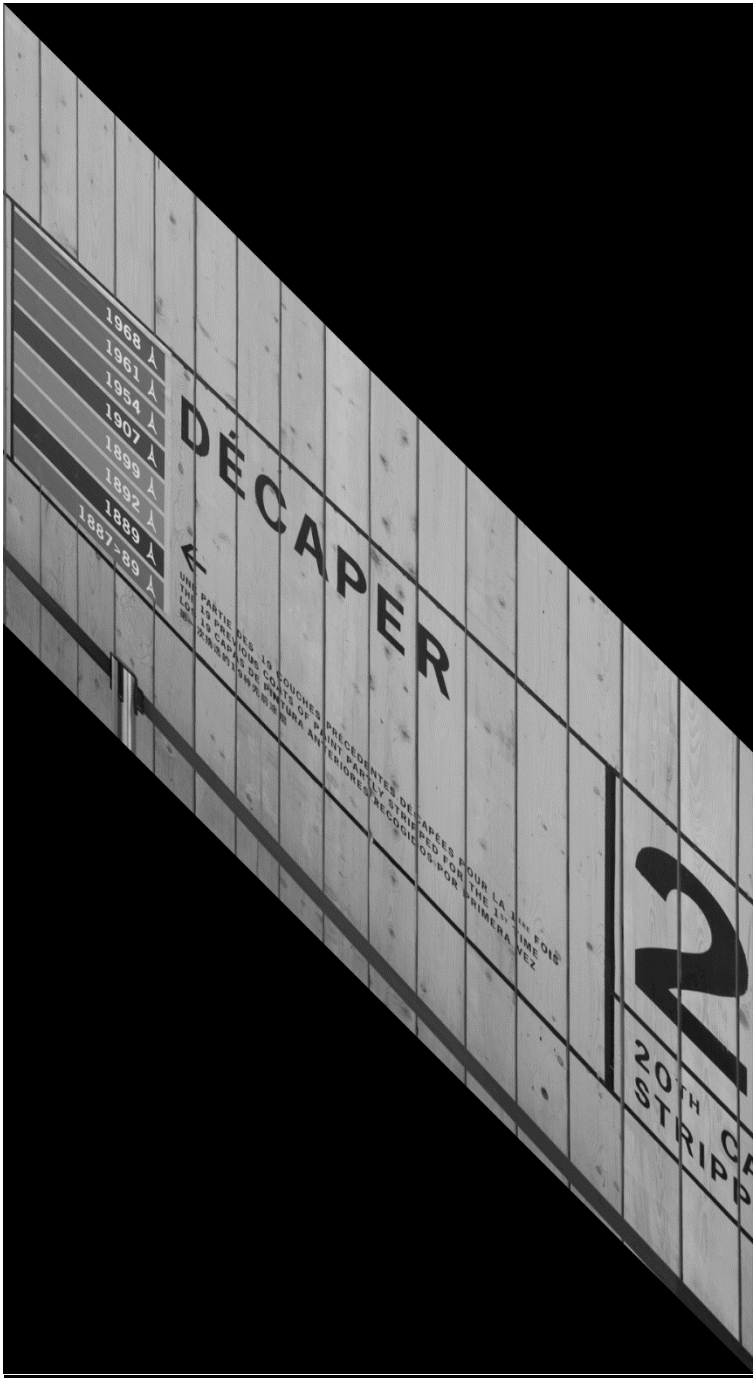
Transformation Name	Transformation n
transformation_scale_horizontal_shear	$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

התוצאה:



2.5 טרנספורמציות - Vertical Shear - מתיחה לאורך:

Transformation Name	Transformation
transformation_scale_vertical_shear	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



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

Transformation Name	Transformation
transformation translation	$\begin{bmatrix} 1 & 0 & 100 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix}$

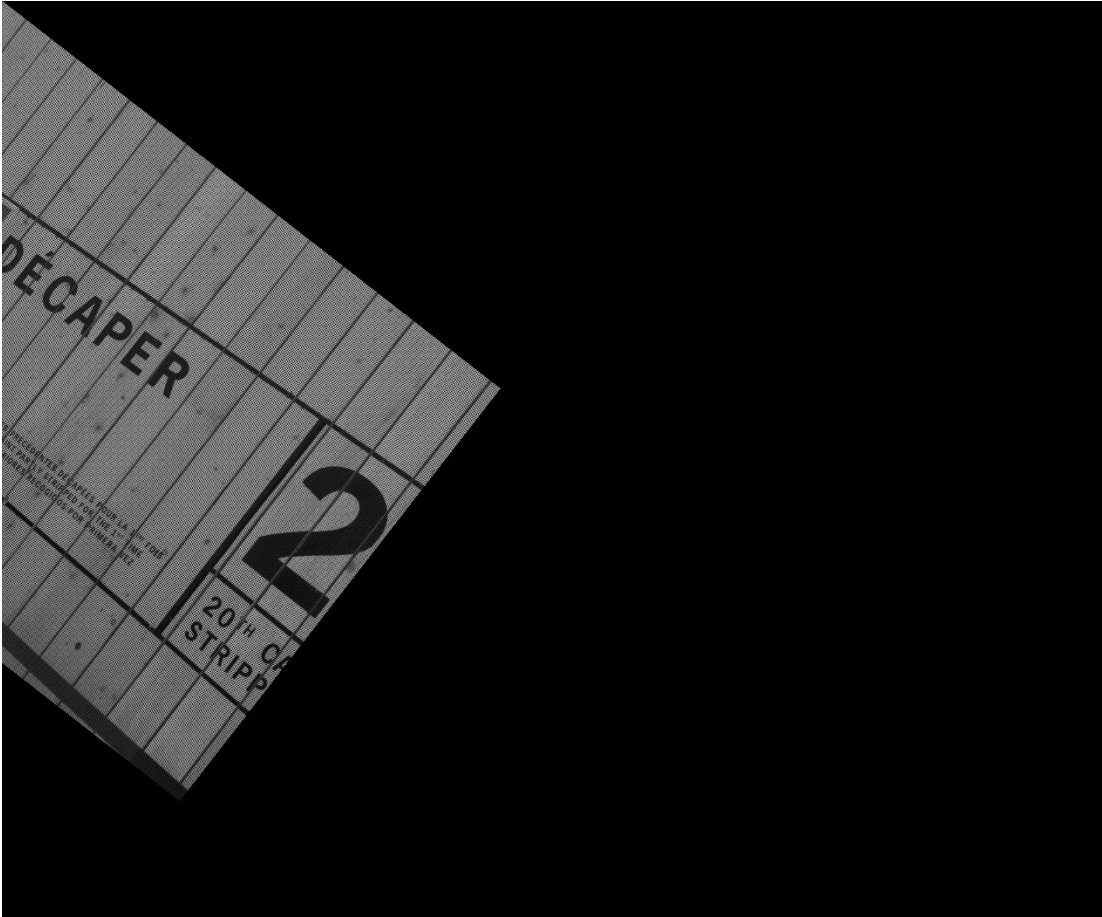
תוצאה:



2.7 טרנפורמציית Rotation – סיבוב:

Transformation Name	Transformation
transformation rotation	$\begin{bmatrix} 0.9 & -0.7 & 0 \\ 0.7 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

תוצאה:



3. Morphological Operators

בחלק זה של הפרוייקט מימשנו אופרטורים מורפולוגיים שונים על התמונה,

להלן הקוד והלוגיקה שמבצעת את עיקר העבודה מול אופרטורים מורפולוגיים שונים:

הפונקציה מקבלת mode (סוג אופרטור מורפולוגי), תמונת שבוצע עליה threshold ששמה binary image, se מסמן Structure Element ו se_center מחזיק מערך עבור נקודת אמצע של ה Structure Element.

```
def morphological_operators(mode, binary_image, se, se_center=[0, 0]):  
    """  
    This function executes the morphological_operators according to  
    the passed Structure Element se.  
    :param mode: hit_and_miss or dilation  
    :param binary_image: passed binary image (thresholded)  
    :param se: Structure Element  
    :param se_center: [x_pos, y_pos] coordinates of the Structure  
    Element  
    :return: result - binary map after execution of Structure Element  
    """  
    rows, cols = binary_image.shape  
    se_rows, se_cols = se.shape  
    result = np.zeros([rows, cols], dtype=np.uint8) # result with  
    pad  
  
    se_forward_distance_row = ((se_rows - 1) - se_center[0])  
    se_forward_distance_col = ((se_cols - 1) - se_center[1])  
  
    for i in range(rows):  
        for j in range(cols):
```

בחלק הבא של הפונקציה מתבצעת עיקר העבודה עבור ביצוע לוגיקה של hit and miss, ריצה וסריקה של מטריצת האופרטור המורפולוגי שהועבר אל מול התמונה הבינארית, כולל כיסוי של מקרי קצה – פינות המטריצה, וצלעות המטריצה.

```
if mode == 'hit_and_miss':  
    # edge cases:  
    # 1. 4 edges  
    if (i-se_center[0] < 0) and (j-se_center[1] < 0): # top left  
    corner v v  
        se_col_diff = abs(j - se_center[1])  
        se_row_diff = abs(i - se_center[0])  
        sliced_se = se[se_row_diff:, se_col_diff:]  
        sliced_binary = binary_image[:se_rows - se_row_diff, :se_cols  
- se_col_diff]  
  
        elif (i+se_forward_distance_row >= rows) and  
(j+se_forward_distance_col >= cols): # bottom right corner v v  
            se_row_diff = i + se_forward_distance_row - (rows-1) # row  
se pixels out of border  
            se_col_diff = j + se_forward_distance_col - (cols-1) # col  
se pixels out of border  
            sliced_se = se[:se_rows - se_row_diff, :se_cols -  
se_col_diff]  
            sliced_binary = binary_image[i - se_center[0]:, j -  
se_center[1]:]
```

```

        elif (i-se_center[0] < 0) and (j + se_forward_distance_col >=
cols): # top right corner v v
            se_col_diff = j + se_forward_distance_col - (cols-1) # col
se pixels out of border
            se_row_diff = abs(i-se_center[0]) # exact amount of out se
pixels
            real_forward_aviable_pixels = se_cols - se_col_diff - 1 #
amount of pixels until out of border
            sliced_se = se[se_row_diff:, :real_forward_aviable_pixels if
real_forward_aviable_pixels > 0 else 1]
            sliced_binary = binary_image[:se_rows - se_row_diff, j-
se_center[1]:]

        elif (i+se_forward_distance_row >= rows) and (j-se_center[1] <
0): # bottom left corner v v
            se_col_diff = abs(j-se_center[1]) # exact amount of out se
pixels
            # se_row_diff = abs(i-se_center[0])
            se_row_diff = i + se_forward_distance_row - (rows - 1) # row
se pixels out of border
            sliced_se = se[:se_rows - se_row_diff, se_col_diff:]
            sliced_binary = binary_image[i-se_center[0]:, :se_cols -
se_col_diff]

        # 2. just borders
        elif i-se_center[0] < 0: # only rows upper part
            se_row_diff = abs(i - se_center[0]) # exact amount of out se
pixels
            sliced_se = se[se_row_diff:, :]
            sliced_binary = binary_image[:se_rows - se_row_diff, j -
se_center[1]:j + se_forward_distance_col+1]

        elif j-se_center[1] < 0: # only cols left part
            se_col_diff = abs(j - se_center[1])
            sliced_se = se[:, se_col_diff:]
            sliced_binary = binary_image[i - se_center[0]:i +
se_forward_distance_row + 1, :se_cols - se_col_diff]

        elif i+se_forward_distance_row >= rows: # only rows bottom part
            se_row_diff = i + se_forward_distance_row - (rows-1) # row
se pixels out of border
            sliced_se = se[:se_rows - se_row_diff, :]
            sliced_binary = binary_image[i - se_center[0]:, j -
se_center[1]:j + se_forward_distance_col+1]

        elif j + se_forward_distance_col >= cols: # only cols right part
            se_col_diff = j + se_forward_distance_col - (cols-1) # col
se pixels out of border
            sliced_se = se[:, :se_cols - se_col_diff]
            sliced_binary = binary_image[i - se_center[0]:i +
se_forward_distance_row + 1, j-se_center[1]:]

        else: # middle zone in the matrix
            sliced_se = se[:, :]
            sliced_binary = binary_image[i - se_center[0]:i +
se_forward_distance_row + 1, j - se_center[1]:j +
se_forward_distance_col+1]

        # run over the 2 matrix and check match:
        sliced_se_rows, sliced_se_cols = sliced_se.shape

```

```

matchFlag = True # indicates if SE matches current scanned zone
for se_sliced_row in range(sliced_se_rows):
    for se_sliced_col in range(sliced_se_cols):
        if sliced_se[se_sliced_row, se_sliced_col] == 1 and
sliced_binary[se_sliced_row, se_sliced_col] != 1:
            matchFlag = False
            if not matchFlag:
                break

if matchFlag:
    result[i, j] = 1 # match
else:
    result[i, j] = 0 # no match

```

בחלק הבא מתבצעת התייחסות במידה ואנחנו מבצעים אופרטור מורפולוגי של dilation:

במידה ומצאנו במסיכת ה threshold ערך 1, נמרח מסביב ע"פ האופרטור שהעובר, כמובן שמתבצע חיפוש אינדקסים מסביב על מנת לדעת היכן במדויק לצייר את הפיקסלים:

```

elif mode == 'dilation':
    # print in the result 1's as the SE:
    if binary_image[i, j] == 1:
        for r in range(se_rows):
            for c in range(se_cols):
                if se[r, c] == 1: # need to put 1
                    try: # won't execute if out of bound
                        # find i:
                        if r < se_center[0]:
                            current_i = i - se_center[0]

                        elif r > se_center[0]:
                            current_i = i + se_center[0]

                        else: # in center
                            current_i = i
                    # find j:
                    if c < se_center[1]:
                        current_j = j - se_center[1]
                        pass
                    elif c > se_center[1]:
                        current_j = j + se_center[1]
                        pass
                    else:
                        current_j = j

                    result[current_i, current_j] = 1
            except Exception as e:
                pass

return result

```

HIT AND MISS 3.1

```
### 3. ----- Verification -----  
##### 3.1 - find number 1 in sign:  
work_image_sliced = work_image[450:600, 75:300] # slice image  
cv2.imwrite('image_output/morphological_operators/hit_and_miss/work_image_sli  
ced {}.png'.format(time.time()), work_image_sliced)
```

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



3.1.1 מציאת הסיפרה 1:

צרכנו Structure Element עם הצורה המאונכת הבאה עבור hit:

[illegible]

ההשתמשנו ב Structure Element הבא על מנת לבודד אותיות, בתהליך בביצוע miss:

[illegible]

תחילה ביצענו תהליך Threshold עם סף 0.689:

בחרנו בסף זה כי בתוצאה רואים בבירור הפרדה של הטקסט משאר התמונה ויכולנו למפות את האותיות בצורה נוחה.

```
work_image_threshold = verification.get_threshold(work_image_sliced,
threshold=0.689) # extract to image file
work_image_threshold_to_255 =
verification.binary_image_to_255(work_image_threshold) # thresh to 255
cv2.imwrite('image_output/morphological_operators/hit_and_miss/work_image_thr
eshold_to_255_{}.png'.format(time.time()), work_image_threshold_to_255) #
to file
```

וקיבלנו את התוצאה הבאה:



לאחר מכן בוצעה הרצה של תהליך hit עם sen הנ"ל:

```
work_hit = verification.morphological_operators("hit_and_miss",
work_image_threshold,
verification.structure_element_number_1,
[10, 5]) # search hit
work_hit_to_255 = verification.binary_image_to_255(work_hit) # hit to 255
cv2.imwrite('image_output/morphological_operators/hit_and_miss/work_hit_to_25
5_{}.png'.format(time.time()), work_hit_to_255) # hit thresh to file
```

בחרנו את אמצע ה Structure Element כנקודת המיקוד – [5, 10].

וקיבלנו את התוצאה הבאה:



בלבן מסומנות "התאמות" שתואמות את sen שהעברנו.

בוצע היפוך בינארי עבור ה threshold שיצרנו:

```
work_miss = verification.morphological_operators("hit_and_miss",
verification.opposite_threshold(work_image_threshold),
verification.structure_element_number_general_revert,
[10, 5]) # search miss
work_miss_to_255 = verification.binary_image_to_255(work_miss) # miss
thresh to 255
cv2.imwrite('image_output/morpological_operators/hit_and_miss/work_miss_to_2
55_{}.png'.format(time.time()), work_miss_to_255) # to file
```

פונקציית עזר שבה נעזרנו על מנת להפוך את התמונה הבינארית:

```
def opposite_threshold(image):
    """
    This function gets threshold image and reverts it - 0 instead of
    1 and vice versa
    :param image:
    :return: return_image - binary opposite of given image
    """
    rows, cols = image.shape
    return_image = np.zeros(shape=(rows, cols), dtype=np.uint8)
    for row in range(rows):
        for col in range(cols):
            if image[row,col] == 1:
                return_image[row, col] = 0
            elif image[row, col] == 0:
                return_image[row, col] = 1
    return return_image
```

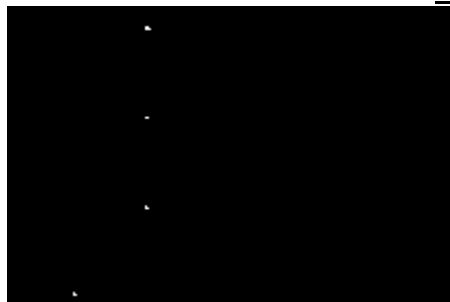
לאחר התהליך קיבלנו את התוצאה הבאה:



ביצענו פעולת AND בין ה miss לבין הhit:

```
work_hit_and_miss = verification.hit_and_miss(work_hit, work_miss) # AND
operation between hit and miss
```

וקיבלנו את החיתוך הבא:



שלחנו את התמונה המקורית ואת תוצאת פעולת ה and לפונקציה ייעודית שמסמנת את המטרות שמצאנו:

```
final_result_image =  
verification.print_target_hit_miss_result(work_hit_and_miss,  
work_image_sliced) # print target result on original image
```

הפונקציה הבאה מקבלת מיפוי בינארי hit and miss ותמונת מקור image ומבצעת מיסגור ברדיוס 5 פיקסלים מסביב למטרות שנמצאו:

```
def print_target_hit_miss_result(hit_and_miss, image):  
    """  
    This function gets hit and miss map result and original image,  
    and prints borders around targets  
    Returns image copy with targets  
    :param hit_and_miss: binary hit and miss map  
    :param image: given image 0 to 255 values  
    :return: image_copy - image with targets around targets  
    """  
    rows, cols = hit_and_miss.shape  
    image_rows, image_cols = image.shape  
    image_copy = image.copy()  
    # hit_and_miss_result = np.zeros(shape=(image_rows, image_cols),  
    dtype=np.uint8)  
  
    for row in range(rows):  
        for col in range(cols):  
            if hit_and_miss[row,col] == 1:  
  
                # print border around target in white  
                try: # ignore close to border exceptions  
                    for i in range(5):  
                        image_copy[row+i, col+5] = 255  
                        image_copy[row-i, col-5] = 255  
                        image_copy[row+5, col+i] = 255  
                        image_copy[row-5, col-i] = 255  
                except Exception as e:  
                    pass  
  
    return image_copy
```

להלן תוצאת ההרצה:



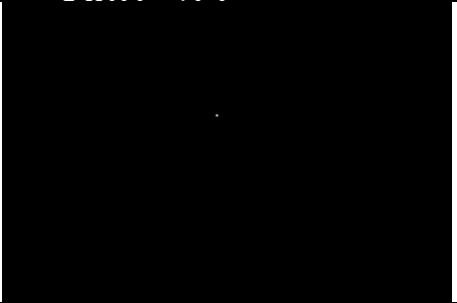





ניתן בבירור לראות כי ספרות 1 נמצאו וסומנו בתמונה לאחר ביצוע הלוגיקה.

3.1.2 דוגמא נוספת - מציאת ספרה 2:

חזרנו על התהליך אך הפעם שלחנו את sen 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, 0, 0, 0]
[0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

תוצאה	תיאור פעולה
	תמונת המקור החתוכה
	לאחר ביצוע $\text{threshold} = 0.689$
	מציאת Hit

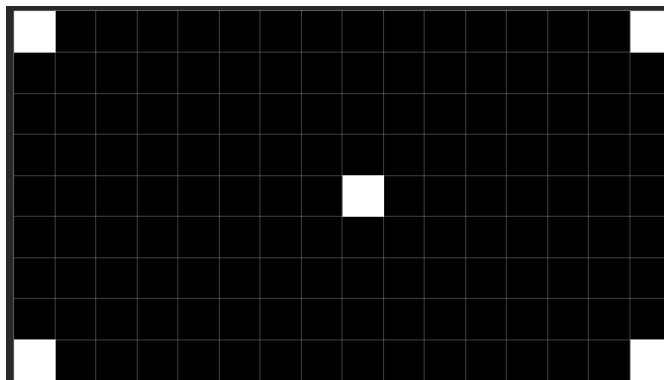
	<p>היפוך threshold וביצוע חיפוש על ההיפוך</p>
	<p>תוצאת פעולת AND עבור שני thresholdים לאחר ביצוע Hit and miss</p>
	<p>מיסגור וסימון המטרות לאחר מציאתן על התמונה המקורית (ניתן לראות ריבוע מסומן מעל הספרה '2')</p>

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



בוצע תהליך Dilation באמצעות ה Structure Element הבא:

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

קטע הקוד הבא מבצע את התהליך, כאשר שלחנו את se הנ"ל עם נקודת המרכז

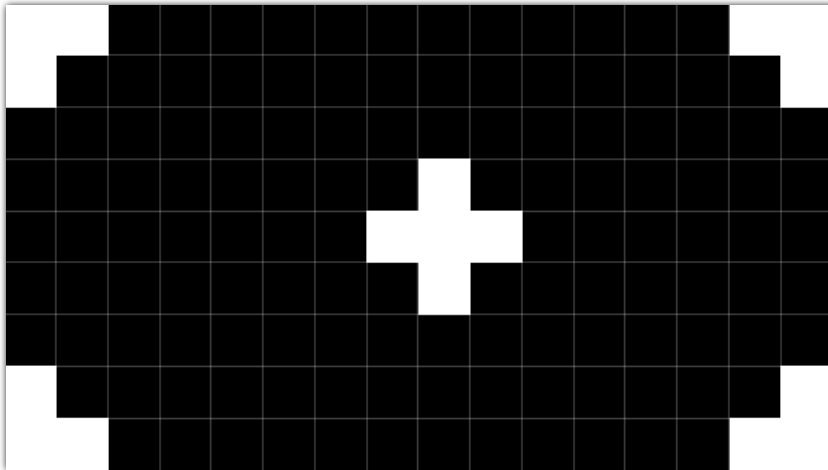
: [1,1]

```
##### 3.3 - Dilation - test image
test_dilation = verification.test_image_2 # test_dilation image
dilation_test_to_255 = verification.binary_image_to_255(test_dilation) # to
255
cv2.imwrite('image_output/morpological_operators/Dilation/dilation_test_imag
e_{}.png'.format(time.time()), dilation_test_to_255) # to file

dilation = verification.morphological_operators("dilation",
                                                verification.test_image_2,
                                                verification.test_se,
                                                [1, 1]) # get resulted
dilation

dilation_test_to_255 = verification.binary_image_to_255(dilation) # to 255
cv2.imwrite('image_output/morpological_operators/Dilation/dilation_test_resu
lt_{}.png'.format(time.time()), dilation_test_to_255) # to file
```

תוצאה לאחר ביצוע התהליך:



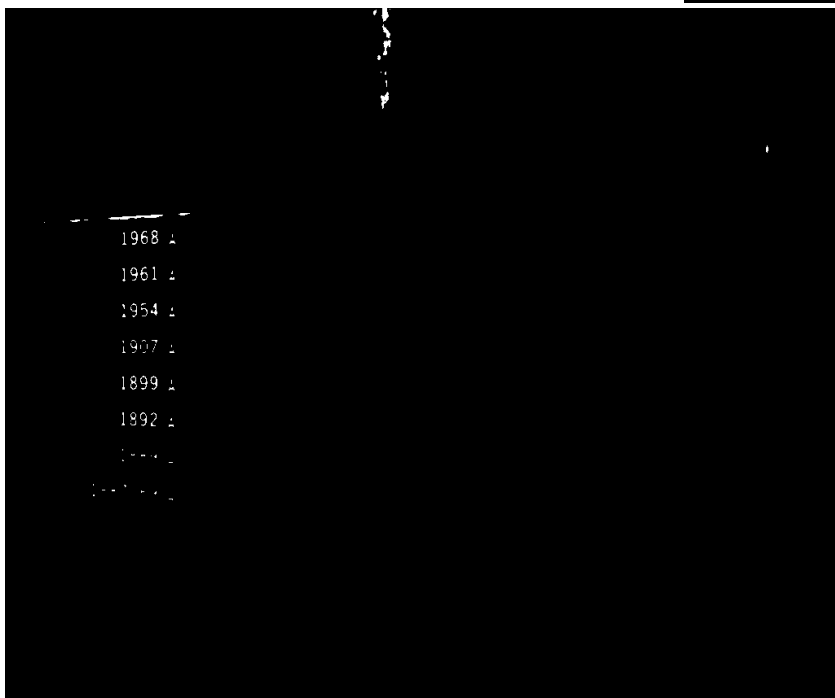
ניתן לראות כי התהליך הצליח ובכל מקום בו הופיע 1 (לבן) בתמונה המקורית, נצבעו מסביב פיקסלים בצורת ה Structure Element שנשלח אליו.

3.3 שימוש בDilation להרחבת הטקסט בתמונה שבחרנו:

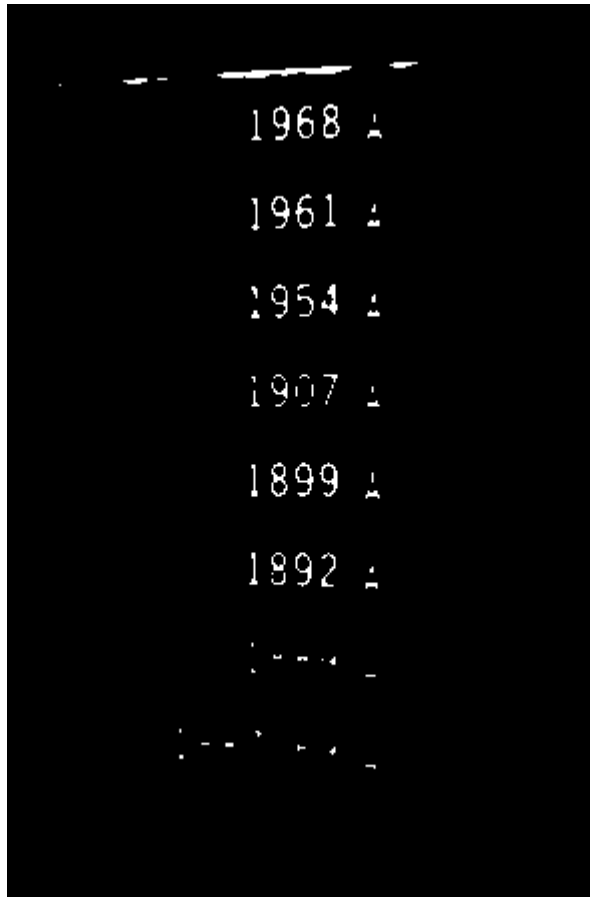
תחילה ביצענו threshold חדש עבור התמונה עם ערך 0.755 על מנת לבודד את הטקסט באופן המיטבי, להלן הקוד:

```
##### 3.4 - Dilation - over a picture - make text in picture Bolder:
work_image_threshold = verification.get_threshold(work_image,
threshold=0.755) # threshold
work_image_threshold_to_255 =
verification.binary_image_to_255(work_image_threshold) # threshold to 255
cv2.imwrite('image_output/morphological_operators/Dilation/work_image_dilation_threshold_{}.png'.format(time.time()), work_image_threshold_to_255) # to file
```

תוצאת threshold:



ZOOM IN לאיזור הטקסט להשוואה בהמשך:



בוצע תהליך Dilation באמצעות ה Structure Element הבא:

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

שלחנו את sen עם נקודת המרכז [1,1]:

```
result_dilation_mask = verification.morphological_operators("dilation",
work_image_threshold,
verification.test_se,
[1, 1]) #
dilation mask result
result_dilation_mask_to_255 =
verification.binary_image_to_255(result_dilation_mask) # dilation mask to
255
cv2.imwrite('image_output/morpological_operators/Dilation/result_dilation_ma
sk{}.png'.format(time.time()), result_dilation_mask_to_255) # to file
```

להלן התוצאה:



השוואה בין 2 התהליכים, ניתן לראות כי הטקסט הודגש במסיכה:

Threshold Before Dilation	Threshold After Dilation

לאחר מכן ביצענו הרחבת הטקסט בתמונה המקורית בעזרת פונקציה ייעודית: הפונקציה מקבלת threshold, ותמונת מקור image, Structor Element ששמו se, ומרכז עבור Sen. הפונקציה מבצעת מימוש Structor Element שראינו קודם לכן להרחבת הטקסט:

```
def dilation_implemented_on_image(binary_threshold, image, se,
se_center):
    """
    This function smears pixels as the structure element according to
    binary threshold over the image
    :param binary_threshold: threshold of give image.
    :param image: give image
    :param se: structure element
    :param se_center: structure element center coordinates
    :return:
    """
    image_row, image_col = image.shape
```

```

se_row, se_col = se.shape

image_copy = image.copy()

for row in range(image_row):
    for col in range(image_col):
        current_value = image[row,col] # save value for smearing

        if binary_threshold[row, col] == 1: # smear here
            for current_se_row in range(se_row):
                for current_se_col in range(se_col):
                    try:
                        # find i:
                        if current_se_row < se_center[0]:
                            current_i = row - se_center[0]
                        elif current_se_row > se_center[0]:
                            current_i = row + se_center[0]
                        else: # in center
                            current_i = row
                        # find j:
                        if current_se_col < se_center[1]:
                            current_j = col - se_center[1]
                        elif current_se_col > se_center[1]:
                            current_j = col + se_center[1]
                        else:
                            current_j = col
                        image_copy[current_i, current_j] =
current_value

                    except Exception as e: # out of borders
                        pass

return image_copy

```

להלן קטע הקוד הקורא לפונקציה הנ"ל:

```

smeared_image =
verification.dilation_implemented_on_image(work_image_threshold,
work_image,

verification.test_se,

[1, 1]) # smear
text over image

```

להלן תוצאת הרחבת הטקסט:



ניתן להשוות את איזור הטקסט בין תמונת המקור לתמונה הנל ונראה את ההבדלים באופן

ברור:

Before	After

3.4 הדפסת תבנית בתוך איזור:

יצרנו פונקציה ייעודית שמדפיסה pattern נתון על איזור התמונה, אנו מקבלים מיפוי בינארי binary (כל threshold יכול להתאים גם), Image תמונת מקור, pattern שאותו נרצה להטביע על התמונה:

```
def print_pattern_on_image(binary, image, pattern):  
    """  
    This function gets binary map, image 0 to 255, and pattern 0 to 255,  
    and prints pattern over the image at positions where map is 1  
    :param binary: binary map 0 or 1  
    :param image: given image 0 to 255  
    :param pattern: given pattern 0 to 255  
    :return: image_copy - image with pattern  
    """  
    image_row, image_col = image.shape  
    pattern_row, pattern_col = pattern.shape  
    image_copy = image.copy()  
  
    for row in range(image_row):  
        for col in range(image_col):  
            if binary[row, col] == 1: # print here  
                image_copy[row, col] = pattern[row % pattern_row, col % pattern_col]  
  
    return image_copy
```

הפונקציה מוצאת את האיזורים המסומנים במיפוי Binary ומבצעת הדפסה מחזורית של pattern באמצעות פעולת מודולו מתמטית.


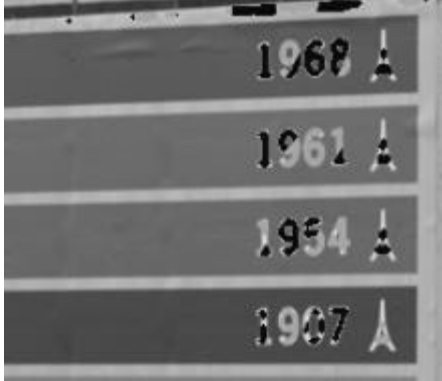
קראנו לפונקציה בעזרת מיפוי הdilation שביצענו לטקסט בסעיף הקודם:

```
resulted_image_with_pattern =  
verification.print_pattern_on_image(result_dilation_mask,  
work_image,  
pattern_image) # print pattern inside text:  
cv2.imwrite('image_output/morphological_operators/pattern_printing/wit  
h_pattern_{}.png'.format(time.time()), resulted_image_with_pattern)  
# to file
```

לאחר הדפסת pattern בתוכו קיבלנו את התוצאה הבאה:



ניתן לראות כי הטקסט קיבל את pattern שהגדרנו בתחילת הדו"ח:

Pattern	Text With Pattern
	

ביצענו threshold נוסף על התמונה המקורית על מנת לבודד את הרקע בצורה ברורה:
בחרנו לשם כך בערך 0.6 threshold, על מנת להבהיר הרקע:

```
work_image threshold = verification.get thershold(work_image,
threshold=0.6) # different threshold mask
work_image_threshold_to_255 =
verification.binary_image_to_255(work_image_threshold) # threshold
to 255
cv2.imwrite('image_output/morpological_operators/pattern_printing/wor
k_image_threshold_{}.png'.format(time.time()),
work image threshold to 255) # to file
```


שלחנו את threshold הנ"ל לפונקצייה שהצגנו להדפסת תבניות:

```
resulted_image_with_pattern =  
verification.print_pattern_on_image(work_image_threshold,  
  
work_image,  
  
pattern_image) # image with pattern result  
cv2.imwrite('image_output/morpological_operators/pattern_printing/wit  
h_pattern_{}.png'.format(time.time()), resulted_image_with_pattern)  
# to file
```

וקיבלנו את התוצאה הבאה:

