Pattern Recognition Project

Name	***ID***
Bassant Mohamed AbdElmoneam	20221376715
Malak Mahmoud Aref	20221445867
Nureen Ehab Mahmoud	20221465124
Zainab Mohamed Abdullah	20221310251

Optical Character Recognition (OCR) using template matching

To perform Optical Character Recognition using template matching we will use opencv library which provides various functions for image processing and computer vision tasks. Our code reads the input image and searches for a given set of characters in the image using template matching and draws a rectangle around them. If the input image is of size (WxH) and template image is of size (wxh), output image will have a size of (W-w+1, H-h+1).

1. Import the needed libraries.

2. Then we read the input image and convert it to grayscale.

```
# read the input image
img = cv2.imread('alphanumeric.png')

# convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

3. Also, we read the template images, and store the templates in a variable called templates as if it's a dictionary with the character as the key and the template image as the value. Here is an example from the code:

```
# define the template images for each alphanumeric character
templates = {
    # Upper case letters
    'A': cv2.imread('Letters/A.png', 0),
    'B': cv2.imread('Letters/B.png', 0),
    'C': cv2.imread('Letters/C.png', 0),
    'D': cv2.imread('Letters/D.png', 0),
    'E': cv2.imread('Letters/E.png', 0),
```

4. Then we get a string of alphanumeric characters from the user using the input() function.

```
# get the user input string
user_input = input('Enter the characters to detect: ')
```

5. We also define a threshold value for the matching score which determines the minimum correlation coefficient required for a match to be considered valid.

```
# define the threshold value for template matching
threshold = 0.8
```

6. Create a for loop for each character in the user input string, we check if there is a corresponding template image in the dictionary. If there is, we get the width and height of the template image then we perform template matching using the cv2.matchTemplate() function which returns a correlation map between the input image and the template with cv2.TM_CCOEFF_NORMED as the matching method.

```
# loop over all the characters in the input string and perform template matching
for char in user_input:
    if char in templates:
        template = templates[char]
        # get the width and height of the template
        w, h = template.shape[::-1]
        # perform template matching using normalized cross-correlation
        result = cv2.matchTemplate(gray, template, cv2.TM_CCOEFF_NORMED)
        # find the locations where the matching template exceeds the threshold
        locations = np.where(result >= threshold)
        # loop over all the locations and draw a rectangle around the matched characters
        for pt in zip(*locations[::-1]):
            cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)
```

The matchTemplate function calculates for every pixel how well the template matches the image at that location. It returns a 2d array with these values. The method we use is TM_CCOEFF_NORMED, the NORMED means the results get normalized, so the values are mapped between 0 and 1.

cv2.TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

7. Then we find the locations where the correlation score exceeds the threshold using the np.where() function to find the locations in the correlation map where the correlation coefficient exceeds the threshold. We then loop over all the locations and draw a rectangle around the matched characters using the cv2.rectangle() function. The rectangle is drawn using a red color with a thickness of 2 pixels.

   The np.where() returns the indices where the value of res/quality of match is greater or equal than the threshold. The indices correspondent to x and y values of the image. The indices are returned as a tuple of 2 arrays, one for the x, one for y.

8. for pt in zip(*loc[:: -1]) → *loc[:: -1] An * allows an arbitrary number of arguments.
   zip() returns an iterable, an object that can be used to loop over. It creates tuples of the input arguments and using for pt in it returns these one by one. In this case the input is an array of x-values and an array of y-values, so it will return an (x,y) tuple. The tuple is stored in pt.

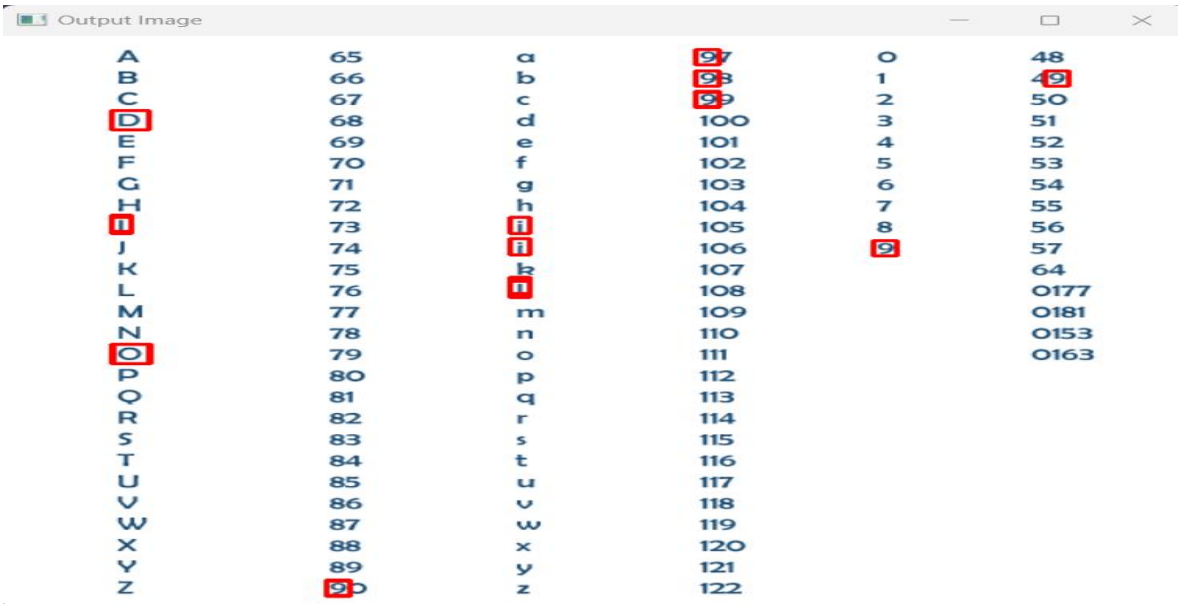
9. Finally, we show the output image using the cv2.imshow() function.

```
# show the output image
cv2.imshow('Output Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Sample Run:

With threshold = 0.8

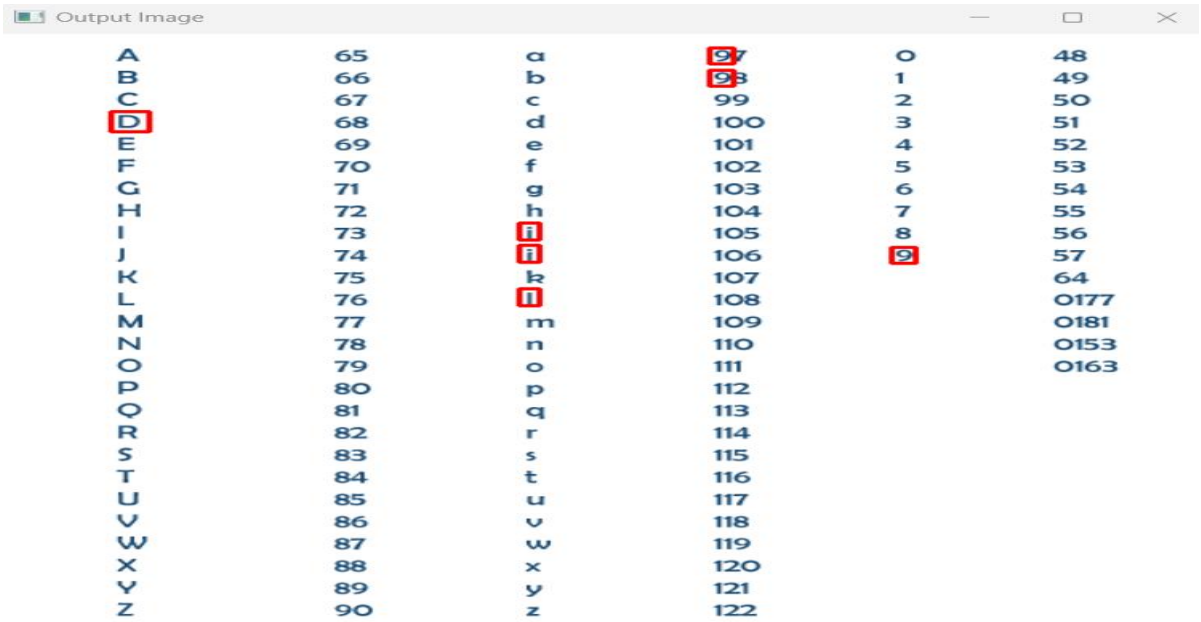
```
Enter the characters to detect: iD9
```



We conclude that the program detects the characters that the user wants to detect but but still not all the characters presented in the image also, it detected characters that the user doesn't want it.

Why this is happened? because of the accuracy, no program will give you 100% accuracy.

If we increase the threshold to 0.9 the output will be more accurate but is still there is character the program doesn't detect it.



Limitations of OCR:

1) The quality of OCR depends on the quality of input image that is provided to it. This means that if there are any imperfections in an image, OCR will have a harder time extracting text from it OCR errors can be even more difficult to fix since they often require the user to correct the OCR errors before re-processing with OCR again.

2) **Sometimes inaccurate:**
   One of the main disadvantages of optical character recognition is that it can be inaccurate. This is because OCR technology is not 100% accurate, and it can sometimes make mistakes when converting images to text. For example, OCR might mistake a lowercase 'I for a "1", or a "b" for an "8". This can cause problems if the text is used for critical purposes, such as in a legal document.

3) **Error prone:**
   OCR can introduce errors, such as incorrectly recognizing a character as a word or line break. A character recognition error is when an OCR engine, in converting text to text, incorrectly recognizes a character as another one. For example, the OCR could recognize "N" and change it to "E." This is common in texts with non-English characters.

4) **Case Sensitivity for Editing:**
   The use of spell checking to correct OCR text will typically not permit the case of the letters to be considered, e.g., cat and CAT will be treated alike but sometimes not.