Detection of word occurrences using an template mechanism

Group 20

Team Members - Group 20

Name	Index Number
PEIRIS T.R.S	16001079
RATHNAYAKE M.S.M	16001206
SENARATH K.L.P.M	16001321
MEDIS Y.D.K	16000897
KARUNARATNE D.M.M.S	16000706

1. INTRODUCTION

- The objective is to determine the position of all instances of a given template in an parent image.
- We have used template matching mechanism to identify the template image inside the parent image.
- Template matching
  - A method for searching and finding the location of a template image in a parent image.
  - It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image.

2. METHODOLOGY

Assumptions

- Given template image is in the same scale of the parent image.
- Otherwise we have to rescale the template image to the same scale of the parent image.

Approach

- We have solved the problem using python openCV image processing library.
- We have used numpy library for faster array calculations.

```
#importing libraries
import cv2
import numpy as np
```

Step 1 – Reading Images

- We import and read parent and template image.

```
#import parent image and template image
parent_image='../assets/page.png'
template_image='../assets/letter.png'

# Read the parent image
mask = cv2.imread(parent_image)
img_parent = cv2.imread(parent_image)
img = cv2.imread(parent_image,0)

# Read the template image
template = cv2.imread(template_image,0)
```

Step 2 - Convert Color

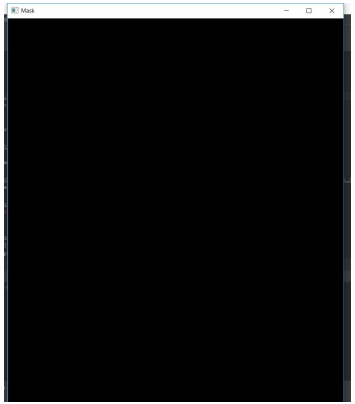
- Converting RGB image to Grayscale

```
#converting RGB to grey image
imageGrey=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
templateGrey=cv2.cvtColor(template,cv2.COLOR_BGR2GRAY)
```

Step 3 - Creating a Filter

- Store the width and height of the parent and template image.

```
#creating black mask for the size of the image
mask=np.zeros(imageGrey.shape,dtype=np.uint8)
```

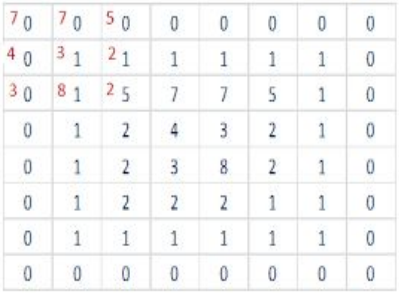


Step 4 – Template Matching

- For template matching we have used openCV function **matchTemplate()** which requires 3 main parameters.
  - Input image
  - Template image
  - Template matching methodology

```
#template matching
result=cv2.matchTemplate(imageGrey,templateGrey,cv2.TM_SQDIFF_NORMED)
```

Step 4 - Template Matching



Input + Template image values.

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

0.8	0.7	...	0.5
0.7	0.4	...	0.7
...	...	...	...
0.4	0.1	...	0.7

Output values

Step 5 - Thresholding

- Since there can be many occurrences of the template image we will define a **Threshold value**.
- When using **normalized coefficient correlation method** every top left corner points that are **less than or equal** to the threshold value will be considered as occurrences of the template image in the input image.We store the location that satisfy the thresholding condition

```
#getting threshold value from the slider
threshold=top.slider.get()

#checking the normalized squared difference value is smaller
#than threshold value so thats its closer to the template or
#equal to the template
location=np.where(result <= threshold)
```

Step 6 -Tagging The Occurrences

- To show the occurrences of the template image we have created a black mask.Mask size will be same as the input image's size.
- We draw white rectangles on the mask where the occurrences appears on

```
#creating black mask for the size of the image
mask=np.zeros(imageGrey.shape,dtype=np.uint8)


#drawing the rectangle on the mask
for point in zip(*location[::-1]):
    cv2.rectangle(mask, point,(point[0]+templateWidth,point[1]+templateHeight),255,-1)
```

Step 7 - Combining The Mask

- After that merge the parent image with mask using bitwise and operator.

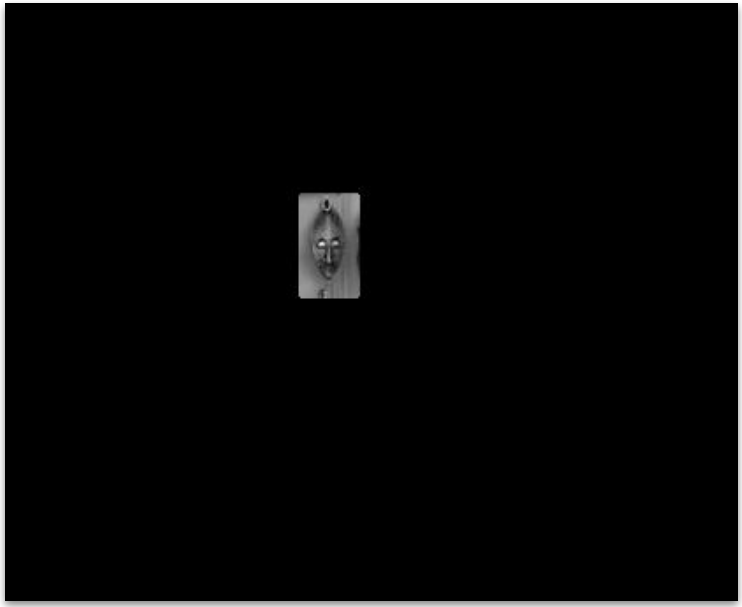
```
#applying the mask ontop of the image
finalimage=cv2.bitwise_and(imageGrey,imageGrey,mask=mask)
```

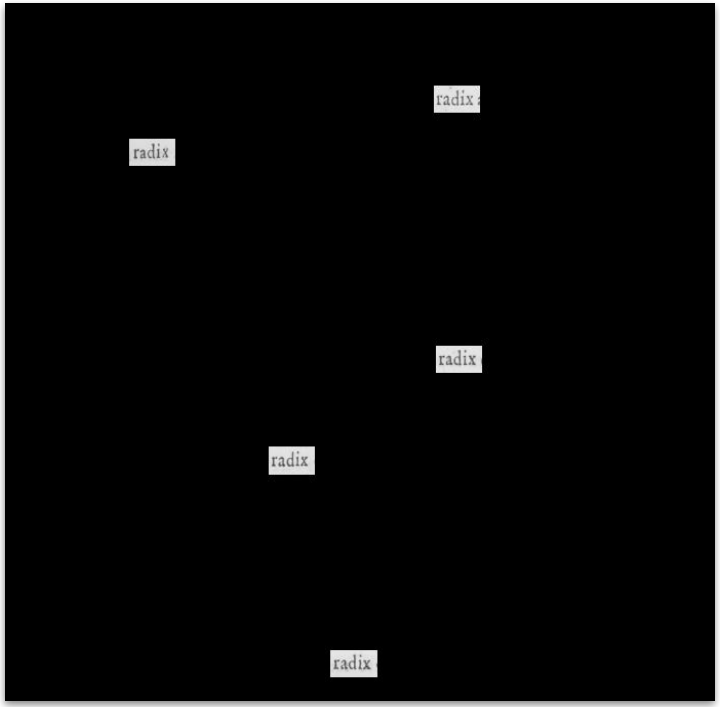
Step 8 – Display Output

- As final step, we used *cv2.imshow* function to display the image.

```
# Show the final image with the matched area.
cv2.imshow('Detected',img_parent)
```

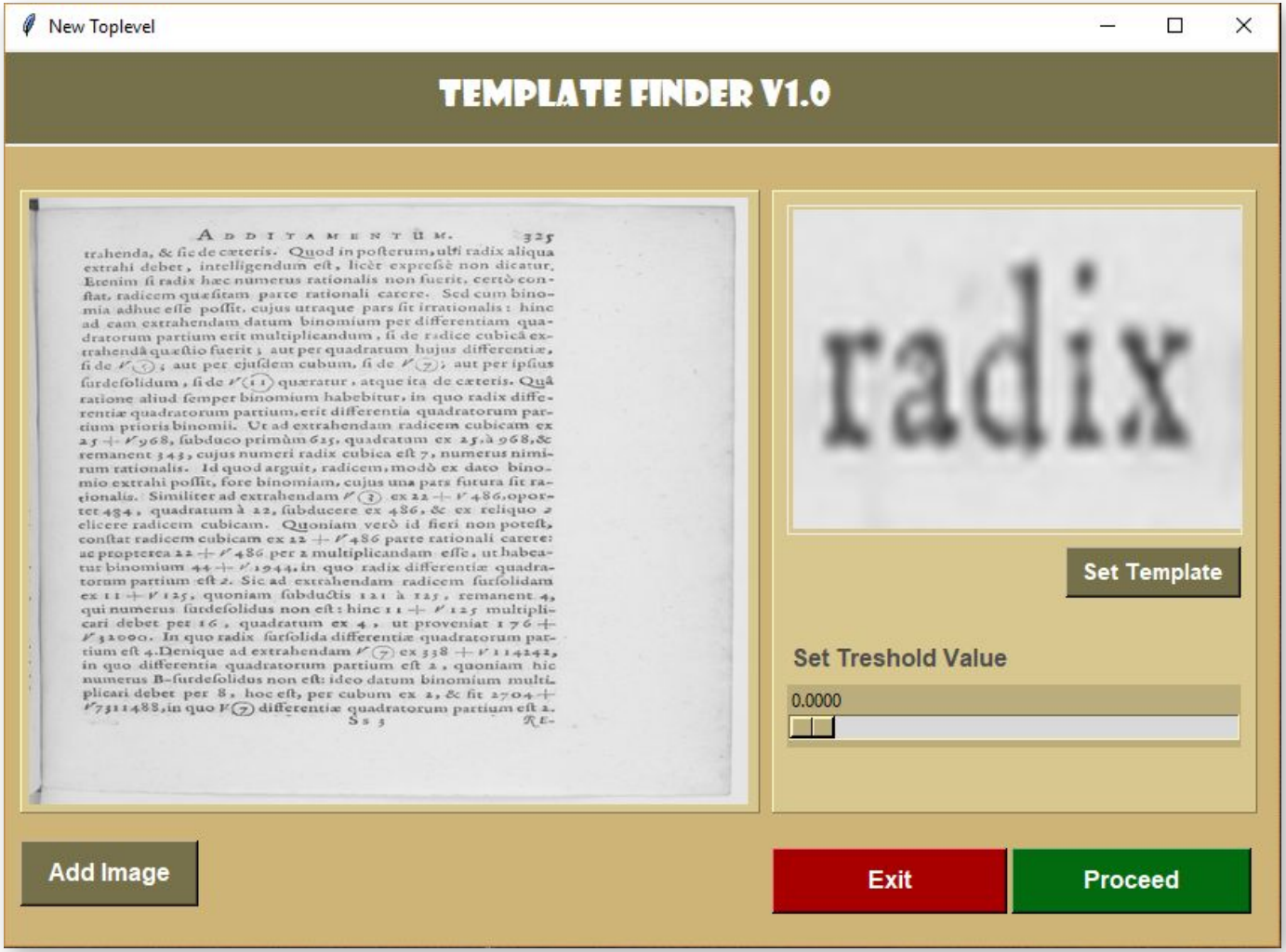
Final Results !!





User Interface

- We have created an interface for the user to perform this task.
- Used Technologies
  - ➢ Tkinter Standard GUI Library
  - ➢ Pillow Library
  - ➢ Page UI building Tool


New Toplevel
TEMPLATE FINDER V1.0
ADDITAMENTUM. 325
trahenda, & sic de cæteris. Quod in posterum, ultr radix aliqua extrahi debet, intelligendum est, licèt expressè non dicatur. Etenim si radix hæc numerus rationalis non fuerit, certò constat, radicem quæsitam parte rationali carere. Sed cum binomia adhuc esse possit, cujus utraque pars sit irrationalis: hinc ad eam extrahendam datum binomium per differentiam quadratorum partium erit multiplicandum, si de radice cubicâ extrahendâ quæstio fuerit; aut per quadratum hujus differentiæ, si de √(3); aut per ejusdem cubum, si de √(7); aut per ipsius surdesolidum, si de √(11) quæratur, atque ita de cæteris. Quâ ratione aliud semper binomium habebitur, in quo radix differentiæ quadratorum partium, erit differentia quadratorum partium prioris binomii. Ut ad extrahendam radicem cubicam ex 25 + √968, subduco primùm 625, quadratum ex 25, à 968, & remanent 343, cujus numeri radix cubica est 7, numerus nimirum rationalis. Id quod arguit, radicem, modò ex dato binomio extrahi possit, fore binomiam, cujus una pars futura sit rationalis. Similiter ad extrahendam √(3) ex 22 + √486, oportet 484, quadratum à 22, subducere ex 486, & ex reliquo 2 elicere radicem cubicam. Quoniam verò id fieri non potest, constat radicem cubicam ex 22 + √486 parte rationali carere: ac propterea 22 + √486 per 2 multiplicandam esse, ut habeatur binomium 44 + √1944, in quo radix differentiæ quadratorum partium est 2. Sic ad extrahendam radicem surdesolidam ex 11 + √125, quoniam subductis 121 à 125, remanent 4, qui numerus surdesolidus non est: hinc 11 + √125 multiplicari debet per 16, quadratum ex 4, ut proveniat 176 + √32000. In quo radix surdesolida differentiæ quadratorum partium est 4. Denique ad extrahendam √(7) ex 338 + √114242, in quo differentia quadratorum partium est 2, quoniam hic numerus B-surdesolidus non est: ideo datum binomium multiplicari debet per 8, hoc est, per cubum ex 2, & fit 2704 + √7311488, in quo √(7) differentiæ quadratorum partium est 2.
Ss 3 RE-
radix
Set Template
Set Treshold Value
0.0000
Add Image
Exit
Proceed

3.
SOURCE CODE

```
def proceed():
    if(parentImage and templateImage):
        #reading images
        image=cv2.imread(parentImage)
        template=cv2.imread(templateImage)

        #converting RGB to grey image
        imageGrey=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        templateGrey=cv2.cvtColor(template,cv2.COLOR_BGR2GRAY)

        #template matching
        result=cv2.matchTemplate(imageGrey,templateGrey,cv2.TM_SQDIFF_NORMED)

        #finding the occurences of the template
        templateHeight,templateWidth=templateGrey.shape

        #getting threshold value from the slider
        threshold=top.slider.get()

        #checking the normalized squared difference value is smaller
        #than threshold value so thats its closer to the template or
        #equal to the template
        location=np.where(result <= threshold)

        #creating black mask for the size of the image
        mask=np.zeros(imageGrey.shape,dtype=np.uint8)
```

```
    #drawing the rectangle on the mask
    for point in zip(*location[::-1]):
        cv2.rectangle(mask, point,(point[0]+templateWidth,point[1]+templateHeight),255,-1)

    #applying the mask ontop of the image
    finalimage=cv2.bitwise_and(imageGrey,imageGrey,mask=mask)

    #Displaying the image

    #cv2.imshow("Template",templateGrey)
    cv2.imshow("Template Detected",finalimage)
    cv2.waitKey(0)
else:
    messagebox.showerror("Error", "Please Select Images to Proceed")
```

THANK YOU!

Q&A