RAMAIAH INSTITUTE OF TECHNOLOGY

Digital Image Processing

Text Extraction from image/Mini Project

2017

NAME	USN
Chandan Singh	1MS15IS029
Shubham Sharda	1MS15IS149
Aashish Agrawal	1MS15IS002
Balaji k.	1MS16IS403

Submitted to:-

Dr. Megha. P. Arakeri

Introduction

Image processing is any form of signal processing for which the input is an image, the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it.

The basic concept used here is of Contour. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

Contour is only detected when there is a difference in the intensity of the Object boundary between the background and foreground image.

So the efficiency of contour also depends on the initial processing of the image. How can we make binirize the image using the different threas-holding techniques.

We know that a text or a letter is a closed area or a boundary, so we have taken benefit of this property of the text. Due to this reason, a contour is formed and the text can be easily extracted.

The project is in its initial stage which will be really helpful in different field like:

Banking (To read Credit Card)

• Libraries (To convert Scanned Page to Image)

• Govt. Sector (Form Processing)

• Used in Car Number Plate Recognition System

• Undesirable Text removal from images once it is completed or taken to the next label.

Problem Statement

1. For the initial processing of the image we have used different threas-holding techniques.

2. For the text extraction, connected component technique is used applying the contours in OpenCV.

Objective

The main aim and Objective of the project is to extract the text written in the image and used in the different field mentioned above.

Requirement

The different requirements for the project are

1. Python 2* or 3*

   For the project Python should be installed on your system. The version of the python can be either 2.7.* or 3.* according your choice.

   Installation Guide:

   Ubuntu:

   ```
   $ sudo apt-get update
   $sudo apt-get install python3.6
   ```

   Windows:

   For Windows you can download python from the following link and continue with the normal installation process.

   https://www.python.org/downloads/

2. OpenCV

Since this project is being done on OpenCv, it should be installed on the system along-side python.

You can visit the following site for the installation guide.

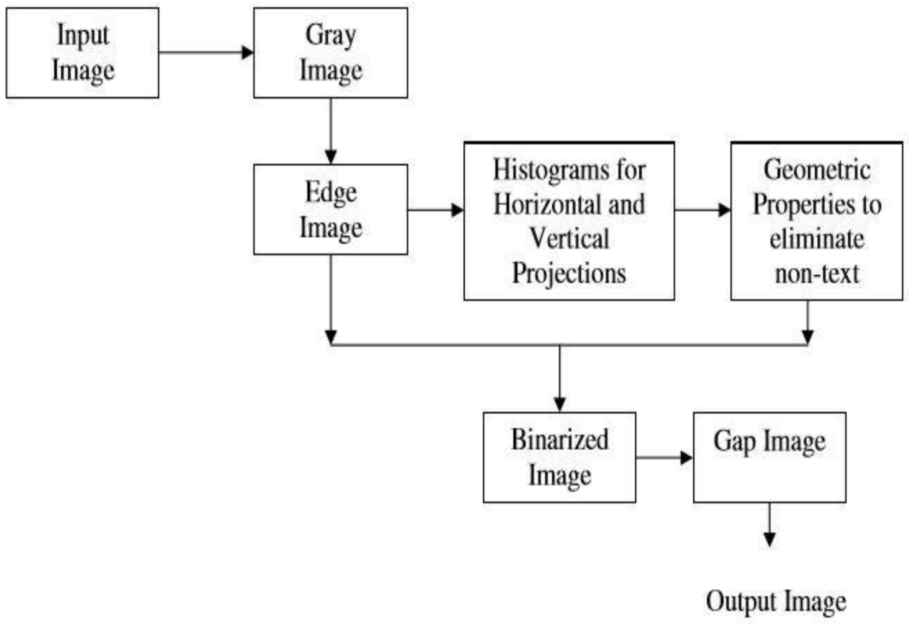
https://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_install/windows_install.html

3. *Numpy*

   Following command will install the numpy on windows.
   Run the following command in command prompt.
   -*pip install numpy*

Methodology

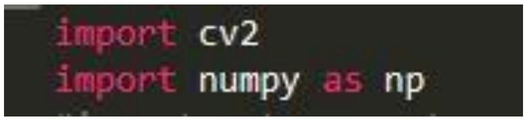
Block-Diagram:


Input Image
Gray Image
Edge Image
Histograms for Horizontal and Vertical Projections
Geometric Properties to eliminate non-text
Binarized Image
Gap Image
Output Image

The Above block Diagram of the project.

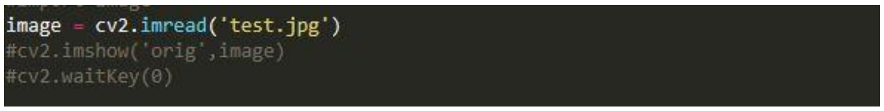
1. First of all the input image is given to the Program.
2. Then the image is converted to the Gray Image.
3. Then After the image is converted to gray scale image , it's edges are found using appropriate Algorithm.
4. Then for the removal for horizontal and vertical non text region we use horizontal and vertical projection.
5. The Output image is obtained which is given for further processing using Counters.

Explanation of Code With each steps Involved



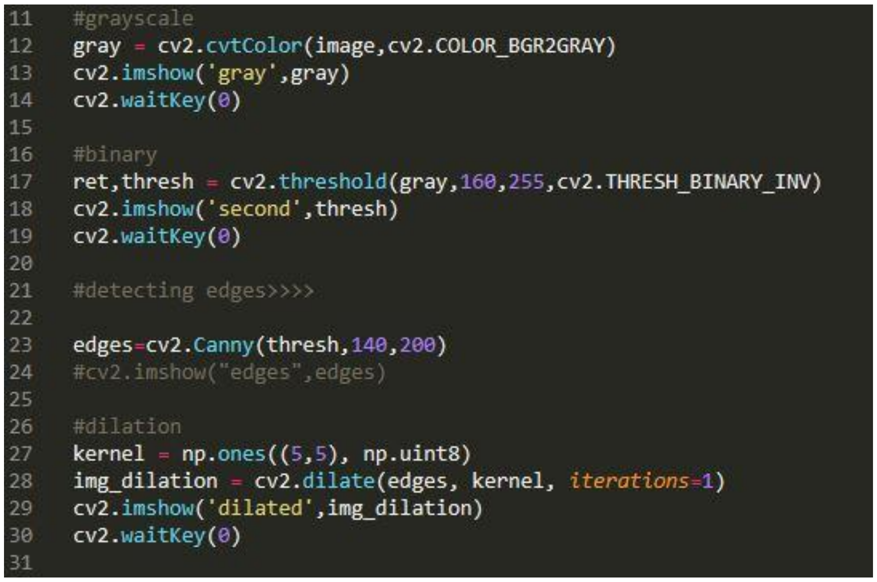
```
import cv2
import numpy as np
```

The Above Line of code shows cv2 and numpy is imported using the import function in the program which is the crutial part of the program.



```
image = cv2.imread('test.jpg')
#cv2.imshow('orig',image)
#cv2.waitKey(0)
```

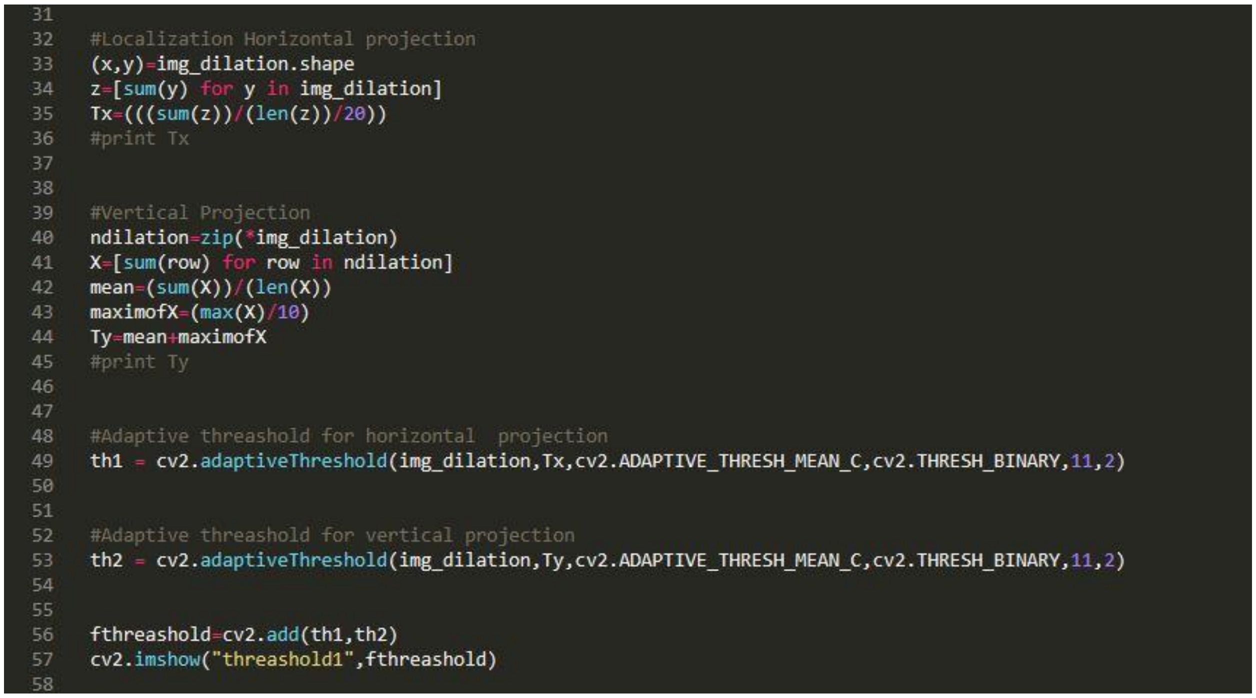
Image is taken as the input with the build-in function in OpenCV i.e. *imread .*



```
#grayscale
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
cv2.imshow('gray',gray)
cv2.waitKey(0)

#binary
ret,thresh = cv2.threshold(gray,160,255,cv2.THRESH_BINARY_INV)
cv2.imshow('second',thresh)
cv2.waitKey(0)

#detecting edges>>>>

edges=cv2.Canny(thresh,140,200)
#cv2.imshow("edges",edges)

#dilation
kernel = np.ones((5,5), np.uint8)
img_dilation = cv2.dilate(edges, kernel, iterations=1)
cv2.imshow('dilated',img_dilation)
cv2.waitKey(0)

```

The Above Code Shows the Initial Processing of the image.

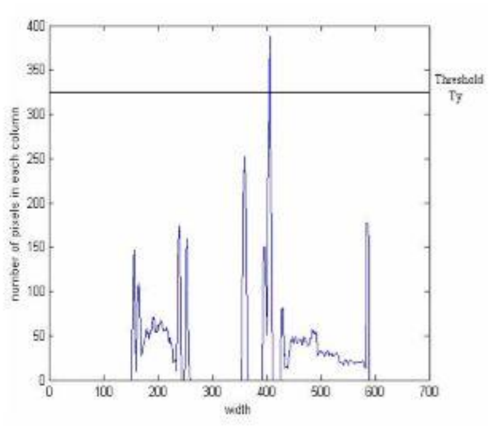
Line (12) shows the image is converted to the grayscale using the function cv2.cvtColor.Then, the edge of the image is detected using Canny-Edge Detection Algorithm, and then the dilation is applied in the image(Dilation adds pixel to the boundary of the image.)



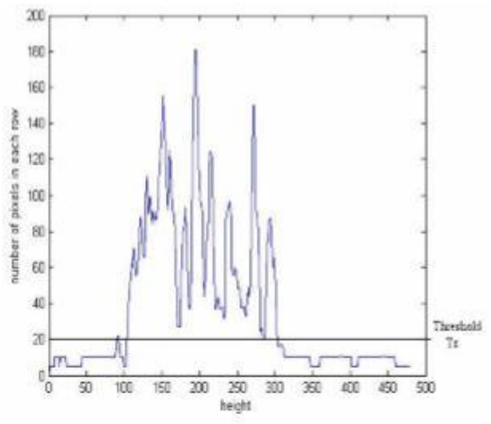
```
#Localization Horizontal projection
(x,y)=img_dilation.shape
z=[sum(y) for y in img_dilation]
Tx=(((sum(z))/(len(z))/20))
#print Tx


#Vertical Projection
ndilation=zip(*img_dilation)
X=[sum(row) for row in ndilation]
mean=(sum(X))/(len(X))
maximofX=(max(X)/10)
Ty=mean+maximofX
#print Ty


#Adaptive threashold for horizontal  projection
th1 = cv2.adaptiveThreshold(img_dilation,Tx,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)


#Adaptive threashold for vertical projection
th2 = cv2.adaptiveThreshold(img_dilation,Ty,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)


fthreashold=cv2.add(th1,th2)
cv2.imshow("threashold1",fthreashold)
```

Now to remove the non-text region what we have done is applied histogram Vertical and Horizontal Projection.

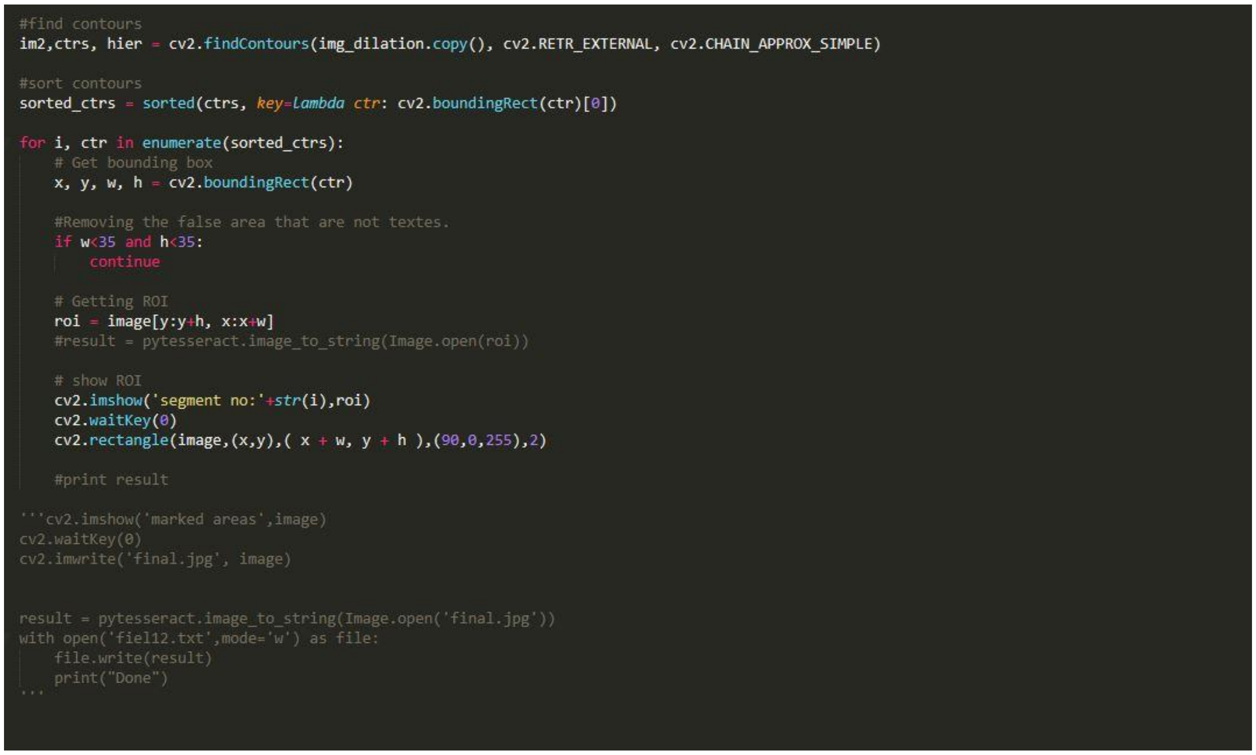
The sharpened edge image is considered as the input intensity image for computing the projection profiles, with white candidate text regions against a black background. The vertical projection profile shows the sum of pixels present in each column of the intensity or the sharpened image. Similarly, the horizontal projection profile shows the sum of pixels present in each row of the intensity image. These projection profiles are essentially histograms where each bin is a count of the total number of pixels present in each row or column.



(a)



(b)



```
#find contours
im2,ctrs, hier = cv2.findContours(img_dilation.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

#sort contours
sorted_ctrs = sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])

for i, ctr in enumerate(sorted_ctrs):
    # Get bounding box
    x, y, w, h = cv2.boundingRect(ctr)

    #Removing the false area that are not textes.
    if w<35 and h<35:
        continue

    # Getting ROI
    roi = image[y:y+h, x:x+w]
    #result = pytesseract.image_to_string(Image.open(roi))

    # show ROI
    cv2.imshow('segment no:'+str(i),roi)
    cv2.waitKey(0)
    cv2.rectangle(image,(x,y),( x + w, y + h ),(90,0,255),2)

    #print result

'''cv2.imshow('marked areas',image)
cv2.waitKey(0)
cv2.imwrite('final.jpg', image)


result = pytesseract.image_to_string(Image.open('final.jpg'))
with open('fiel12.txt',mode='w') as file:
    file.write(result)
    print("Done")
'''
```

The main concept used in this project is **contours.**

See, there are three arguments in cv2.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. Contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

Then the Countours obtained is sorted . And this sorted out countours is made to iterate making it List of Tuples with the function enumerate in python.

Then the Reason of interest is taken from the image, which is the text in the image for us.

This segment is the displayed. Which is our final result.

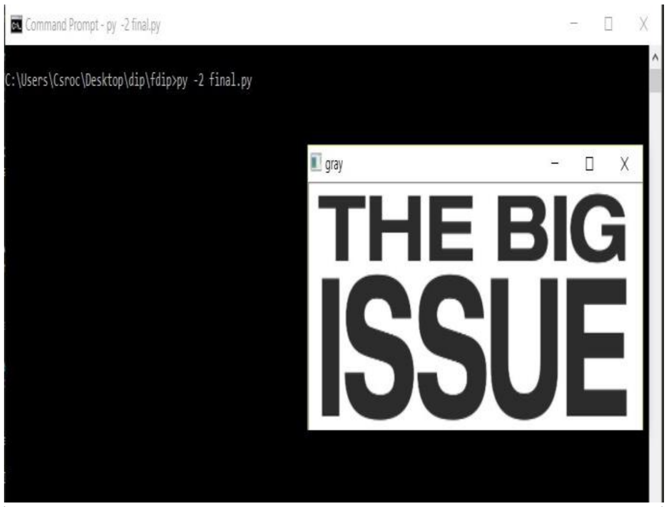
Conclusion

This project is just a short version of the project we are presenting. This project can be taken further, and then can be used for different purposes.

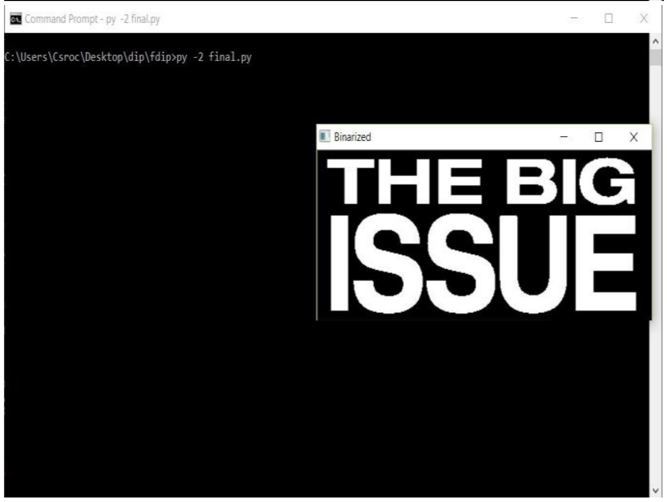
Recently most of the text extraction algorithm uses a direct technique where the whole image is feed to the machine and tries to extract the text from that which is not efficient because it need separate image segmentation for most of the different images.

But, what we are trying to do is first of all extract each text segment using connected component concept i.e Contours and then feed each segment to the separate algorithm, which will treat each segment of text as a separate image. Doing this we will get efficient output, because applying initial processing on a small segment of the image which is already processed is more accurate than applying it on a whole image.

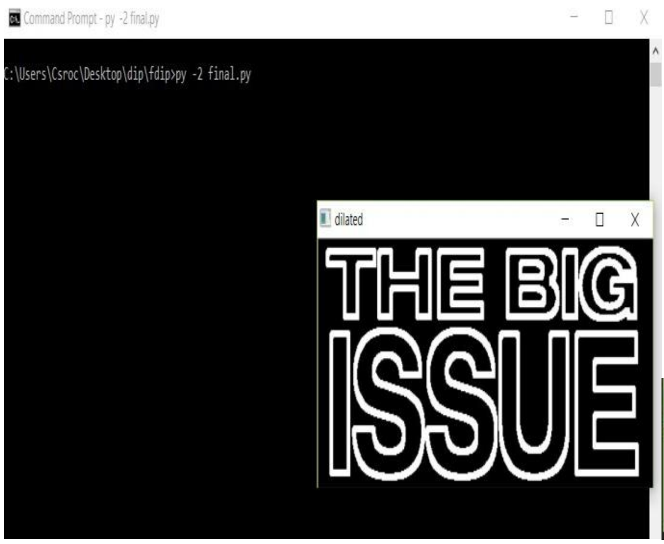
Snippets of the program running.



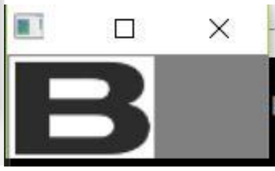
```
Command Prompt - py -2 final.py

C:\Users\Csroc\Desktop\dip\fdip>py -2 final.py
```

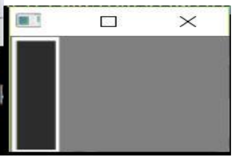


```
Command Prompt - py -2 final.py

C:\Users\Csroc\Desktop\dip\fdip>py -2 final.py
```



```
Command Prompt - py -2 final.py

C:\Users\Csroc\Desktop\dip\fdip>py -2 final.py
```





References

- *http://opencvpython.blogspot.in/2012/06/hi-this-article-is-tutorial-which-try.html*
- *https://docs.opencv.org/3.3.1/dd/d49/tutorial_py_contour_features.html*
- *An Approach To Extract Text Regions From Scene Image*
- *Extraction of text regions in natural images, Sneha Sharma*