Problem Statement/Title :

Write a python program to detect and recognise the license plate of a vehicle using openCV.

Motivation

License plate recognition is an important technology used in various security and traffic applications. The motivation behind this problem statement is to try and implement the concepts learnt in Image Processing on a real life application.

Keywords

OpenCV, pytesseract, optical character recognition, edge detection, contours

Abstract(150 words approx.)

This mini project presents the implementation of license plate recognition in python using the OpenCV library. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision and image processing applications. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. In this project, OpenCV is used for detecting the number plate using image processing techniques. After the image processing step is executed, pytesseract library is used to detect the actual contents of the number plate. This step is known as Optical Character Recognition (OCR). Optical character recognition or optical character reader is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image.

Technology Used

OpenCV

Literature Survey

A research paper titled "Real time license plate detection using openCV and tesseract" was published in the 2017 International Conference on Communication and Signal Processing (ICCSP). The paper uses OpenCV for license plate detection and pytesseract for detecting the actual characters. The paper describes various steps required to extract text from any image file (jpeg/png) and create a separate text file consisting of information extracted from image file.

Module wise Scope/Work flow/Block diagram with explanation

Steps involved :

1. License Plate Detection

   - Read the image.
   - Resize the image to the required size. Convert
   - it to grayscale.
   - Remove the unwanted details from the image using gaussian blur.
   - Perform edge detection using canny edge detection.
   - Start looking for contours in the image. A contour could be anything that has a closed surface.
   - Once the counters have been detected we sort them from big to small and consider only the first 10 results.
   - To filter the license plate image among the obtained results, we will loop though all the results and check which has a rectangle shape contour with four sides and closed figure.
   - Mask the entire picture except for the place where the number plate is.

2. Character Segmentation

   - Segment the license plate out of the image by cropping it and saving it as a new image.

3. Character Recognition

   - The Final step is to actually read the number plate information from the segmented image. We
   - will use the pytesseract package to read characters from image.

Functions used:(Explain with example how it works)

1. `cv2.imRead()`

Example

```
img = cv2.imread('example.jpg')  img
 = cv2.imread('example.jpg',0)
```

cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

2. `cv2.resize`

```
scale_percent = 60
width = int(img.shape[1] * scale_percent / 100)
 height = int(img.shape[0] * scale_percent / 100)
 dim = (width, height)
img = cv2.resize(img,dim )
```

The function resize resizes the image src down to or up to the specified size. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are cv.INTER_AREA for shrinking and cv.INTER_CUBIC (slow) & cv.INTER_LINEAR for zooming. By default, the interpolation method cv.INTER_LINEAR is used for all resizing purposes.

3. `cv2.cvtColour`

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

The function converts an input image from one color space to another. Some example color space conversions -

- COLOR_RGB2GRAY
  Python: cv.COLOR_RGB2GRAY
- COLOR_GRAY2BGR
  Python: cv.COLOR_GRAY2BGR
- COLOR_GRAY2RGB
  Python: cv.COLOR_GRAY2RGB

4. `cv2.Gaussianblur`

```
 # remove unwanted noise using gaussian blur
 # here, kernel of 3x3 size is used
gray = cv2.GaussianBlur(gray, (3, 5), 0)
```

OpenCV provides cv2.gaussianblur() function to apply Gaussian Smoothing on the input source image. Following is the syntax of GaussianBlur() function :

```
dst = cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[,
 borderType=BORDER_DEFAULT]]] )
```

Parameter Description

- src input image
- dst output image
- ksize Gaussian Kernel Size. [height width]. height and width should be odd and can have different values. If ksize is set to [0 0], then ksize is computed from sigma values.
- sigmaX Kernel standard deviation along X-axis (horizontal direction).
- sigmaY Kernel standard deviation along Y-axis (vertical direction). If sigmaY=0, then sigmaX value is taken for sigmaY
- borderType Specifies image boundaries while kernel is applied on image borders. Possible values are : cv.BORDER_CONSTANT cv.BORDER_REPLICATE cv.BORDER_REFLECT cv.BORDER_WRAP cv.BORDER_REFLECT_101 cv.BORDER_TRANSPARENT cv.BORDER_REFLECT101 cv.BORDER_DEFAULT cv.BORDER_ISOLATED

5. `cv2.Canny()`
   Canny edge is an edge detection algorithm. The syntax of OpenCV Canny Edge Detection function is

```
edges = cv2.Canny('/path/to/img', minVal, maxVal, apertureSize,
 L2gradient)
```

where

- /path/to/img (Mandatory) File Path of the image minVal
- (Mandatory) Minimum intensity gradient maxVal
- (Mandatory) Maximum intensity gradient apertureSize
- (Optional)
- L2gradient (Optional) (Default Value : false) If true, Canny() uses a much more computationally expensive equation to detect edges, which provides more accuracy at the cost of resources.

Example

```
edged = cv2.Canny(gray, 30,
220)
```

6. `cv2.findContours()` In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black. There are three arguments in cv.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method.

```
# Contours are defined as the line joining
# all the points along the boundary of an image that are having the same
intensity.
```

```
contours = cv2.findContours(edged.copy(), cv2.RETR_TREE,
 cv2.CHAIN_APPROX_SIMPLE)
```

7. `cv2.arcLength()`

   It is also called arc length. It can be found out using cv.arcLength() function. Second argument specify whether shape is a closed contour (if passed True), or just a curve.

   `perimeter = cv.arcLength(cnt,True)`

8. `cv2.approxPolyDP()`

   A built-in function in OpenCV is used to approximate the shape of polygonal curves to the specified precision called approxPolyDP() function.

   `approx = cv2.approxPolyDP(c, 0.018 * peri, True)`

   Syntax - `approxPolyDP(input_curve, epsilon, closed)` where

- input_curve represents the input polygon whose contour must be approximated with specified precision,
- epsilon represents the maximum distance between the approximation of a shape contour of the input polygon and the original input polygon
- closed is a Boolean value whose value is true if the approximated curve is closed or the value is false if the approximated curve is not closed.

9. `cv2.drawContours()`

   To draw the contours, cv.drawContours function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

```
cv2.drawContours(img, [screenCnt], -1, (0, 0, 255), 3)
cv.drawContours(img, contours, -1, (0,255,0), 3)
```

10. `cv2.bitwise_and()`

    Bit-wise conjunction of input array elements. Used for masking.

```
new_image = cv2.bitwise_and(img,img,mask=mask)
 cv2.bitwise_and(source1, source2, destination, mask)
```

- source1: First Input Image array(Single-channel, 8-bit or floating-point) source2:
- Second Input Image array(Single-channel, 8-bit or floating-point) dest: Output
- array (Similar to the dimensions and type of Input image array) mask:
- Operation mask, Input / output 8-bit single-channel mask

11. `pytesseract.image_to_string()`

Returns unmodified output as string from Tesseract OCR processing

```
 # config = '---psm 7' ---> Treat the image as a single text line.
text = pytesseract.image_to_string(Cropped, config='--psm 7')
```

References:

OpenCV documentation https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html

License Plate Recognition using OpenCV Python
https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c

source Code

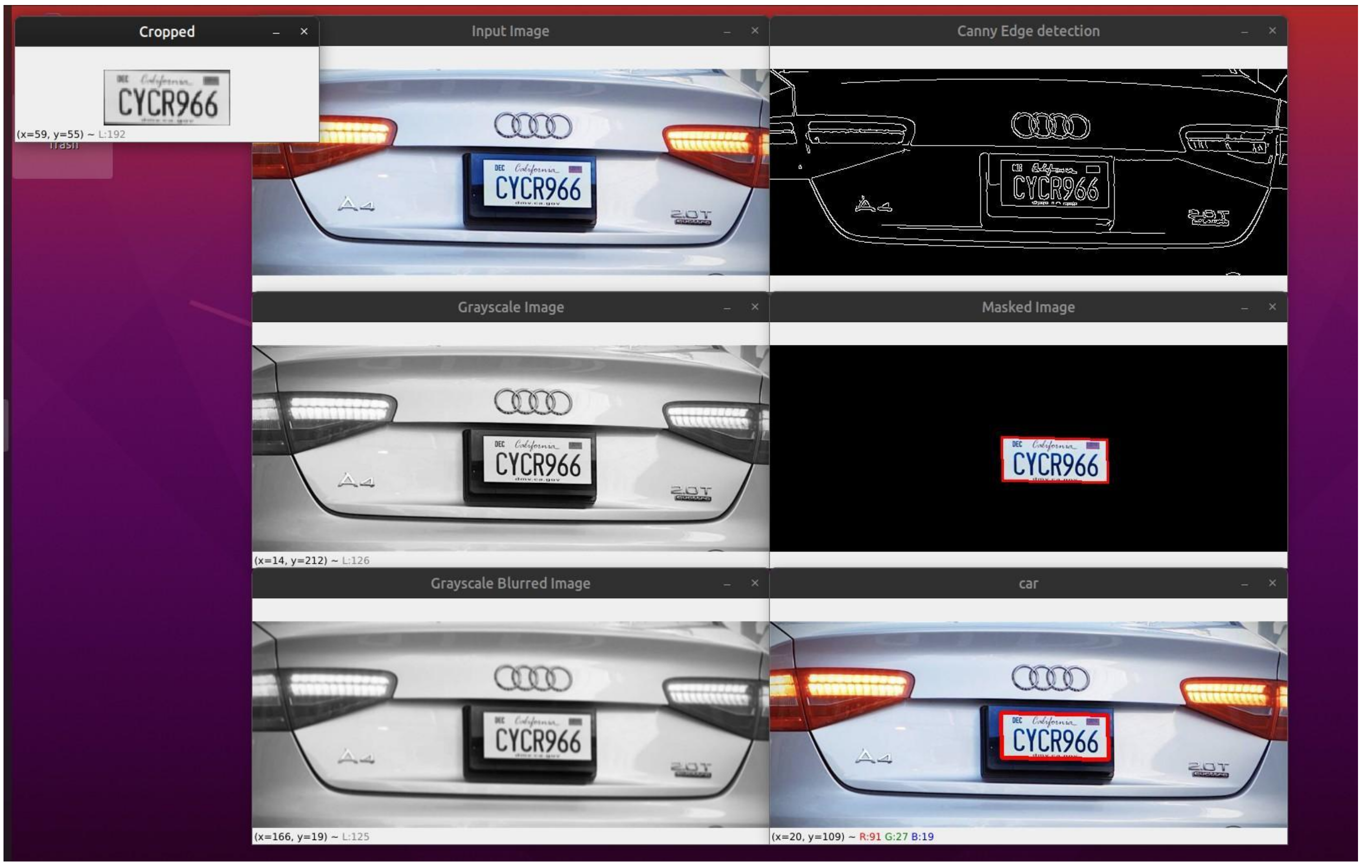
```
# import required libraries
import cv2
import imutils
import numpy as np
import pytesseract

# executing license plate detection on 8 different vehicles
for i in range(1,9):

    name = 'vehicle' + str(i) + '.jpeg'
    # read the image
    img = cv2.imread(name)

    # percent of original size
    scale_percent = 60
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)

    # resize the image
    img = cv2.resize(img,dim )
    # display the image
    cv2.imshow('Input Image',img)
    cv2.waitKey(0)

    # convert colour image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Grayscale Image',gray)
    cv2.waitKey(0)

    # remove unwanted noise using gaussian blur
    # here, kernel of 3x5 size is used
    gray = cv2.GaussianBlur(gray, (3, 5), 0)
    cv2.imshow('Grayscale Blurred Image',gray)
    cv2.waitKey(0)

    # perform canny edge detection #
    minimum threshold value = 30 #
    maximum threshold value = 200
    edged = cv2.Canny(gray, 30, 220)
    cv2.imshow('Canny Edge detection',edged)
    cv2.waitKey(0)

    # find contours in the image
    # Contours are defined as the line joining
    # all the points along the boundary of an image that are having the
same intensity.
    contours = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)


    # Once the counters have been detected we sort them from big to
    # small and consider only the first 10 results ignoring the others
    contours = sorted(contours, key = cv2.contourArea, reverse = True)
[:10]

    # To filter the license plate image among the obtained results, we
will loop
    # though all the results and check which has a rectangle shape contour
    # with four sides and closed figure.
    screenCnt = None

    for c in contours:

        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.018 * peri, True)

        if len(approx) == 4:
            screenCnt = approx
            break

    # Once we have found the right counter we save it in a variable called #
    screenCnt and then draw a rectangle box around it to make sure we
have
```

```
    # detected the license plate correctly.
    if screenCnt is None:
        detected = 0
        print ("No contour detected")
    else:
         detected = 1

    if detected == 1:
        cv2.drawContours(img, [screenCnt], -1, (0, 0, 255), 3)

    # Now that we know where the number plate is, we can proceed with
masking
    # the entire picture except for the place where the number plate is.
    # bitwise and function is used to mask the image
    mask = np.zeros(gray.shape,np.uint8)
    new_image = cv2.drawContours(mask,[screenCnt],0,255,-1,)

    new_image = cv2.bitwise_and(img,img,mask=mask)
    # display the masked image
    cv2.imshow('Masked Image',new_image)
    cv2.waitKey(0)

    # segment the license plate out of the image by cropping it and saving
it as a new image
    (x, y) = np.where(mask == 255) (topx,
    topy) = (np.min(x), np.min(y))
    (bottomx, bottomy) = (np.max(x), np.max(y))
    Cropped = gray[topx:bottomx+1, topy:bottomy+1]

    # read the number plate information from the segmented image
    # config = '---psm 7' ---> Treat the image as a single text line.
    text = pytesseract.image_to_string(Cropped, config='--psm 7')
    print("Vehicle number ",i)
    print("Detected license plate Number is:",text)

    # display final result
    cv2.imshow('car',img)
    cv2.waitKey(0)
    cv2.imshow('Cropped',Cropped)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Sample Output

Vehicle number 5

Detected license plate Number is: CYCR966


Cropped
(x=59, y=55) ~ L:192
Input Image
Canny Edge detection
Grayscale Image
(x=14, y=212) ~ L:126
Masked Image
Grayscale Blurred Image
(x=166, y=19) ~ L:125
car
(x=20, y=109) ~ R:91 G:27 B:19