AUTOMATED NUMBER
PLATE DETECTION AND RECOGNITION

A

Project Report

submitted in partial fulfillment of the
requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE

Specialization in

Business Analytics & Optimization

By:

Name	**Roll No**
Rajeshwarpreet S	R103216079
Smarth Galhotra	R103216098
Anisha Gera	R103216126

Under the guidance of

Dr. Hitesh Kumar Sharma
Assistant Professor, PIC
Department of Informatics

Department of Informatics
School of Computer Science
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
Bidholi, Via Prem Nagar, Dehradun, Uttarakhand
2019-20

CANDIDATES DECLARATION

I/We hereby certify that the project work entitled AUTOMATED NUMBER PLATE DETECTION AND RECOGNITION in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science And Engineering with Specialization in Business Analytics and optimization and submitted to the Department of Informatics at School of Computer Science, University of Petroleum And Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **Jan, 2020** to **May, 2020** under the supervision of **Dr. Hitesh Kumar Sharma, Assistant Professor, Department of Informatics**.

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

(Rajeshwarpreet S)	**(Smarth Galhotra)**	**(Anisha Gera)**
Roll No. R103216079	**Roll No. R103216098**	**Roll No. R103216126**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

(Date: 18 April 2020)
(Dr. Hitesh Kumar Sharma)
Project Guide

(Dr. T.P Singh)
Head
Department of Informatics School of Computer Science
University of Petroleum and Energy Studies
Dehradun - 248001 (Uttarakhand)

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Dr. Hitesh Kumar Sharma**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank , **Dr. T.P Singh**, Head of Department SoCS, for his great support.

We are also grateful to **Dr. Manish Prateek Professor,** Director SoCS, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

Name	**Rajeshwarpreet S**	**Smarth Galhotra**	**Anisha Gera**
Roll No.	**R103216079**	**R103216098**	**R103216126**

ABSTRACT

Automatic recognition of car number plate is vital since the number of vehicles is escalating and it's impossible to manage or monitor such system manually, to resolve problems like traffic monitoring, tracking stolen cars, managing parking toll, rules violation, distinguish registered or guest vehicles. In India this problem is considered as challenging due to irregular shape, diverse formats of plates, different scales or blurred captured images. The aim is to build a system to overcome the above problems using machine learning.

Keywords: Recognition, Machine Learning

TABLE OF CONTENTS

Contents

1 Introduction 7

2 Literature Review 7

3 Problem Statement 8

4 Objective 8

5 Design Methodology 8

6 Implementation 9

6.1 Pseudocode
6.2 Output Screen
6.3 Result Analysis

7 Conclusion and Future Scope 12

LIST OF FIGURES

List of Figures

1. Process Flow 9
2. Architecture of Number Plate Recognition System..9
3. Original Image...11
4. Binary Image...11
5. Threshold Image..11
6. License Plate Character..11

1. Introduction

The purpose of Number Plate Recognition is automatic vehicle identification by applying advanced image processing technologies. A study on vehicles, estimates that more than half a billion cars are on the roads worldwide.

A number plate is a unique alphanumerical registration ID on a metal board that represents the issue of legal license by the competent authority for the use of public roads. Number Plate Recognition is an important component in all Intelligent Transportation Systems (ITS) applications like Border Control, Identifying Stolen car, Electronic Toll Systems, Surveillance, and Intelligent Traffic Control System etc.

There are several solutions available for License Plate Recognition. However, most of them work only under restricted conditions, such as fixed illumination, limited vehicle speed, designated routes, and stationary backgrounds.

This project focuses on designing a solution for Number Plate Recognition which will work in dynamic conditions.

2. Literature Review

Searching for license plate recognition is still a challenge. It involves three major steps. They specify number pad space, character segmentation, and character recognition. Each step suggested different ways to improve efficiency. [1] Automatic recognition of car license plate number got to be an indispensable part in our day by day life. This paper mainly explains an Automatic Number Plate Recognition System (ANPR) using Morphological operations, Histogram manipulation and Edge discovery Techniques for plate localization and characters segmentation. Artificial Neural Networks are used for Character classification and recognition.[2] Tella Pavani used the adaptive threshold to highlight the characters and suppress the background. In order to remove unwanted image spaces, a component algorithm is first applied to the converted binary image from the original panel. A special algorithm called Image Scissoring is used to divide the Optical Character Recognition engine called tesseract, which returns ASCII to the license number. The entire system has been implemented using OpenCV [3] Another way in which character areas are selected is through the binarization, connected component analysis. The Point Analysis method removes unwanted points and combines split points and split points.

3. Problem Statement

Most of the Number Plate Recognition (NPR) solutions work only under restricted conditions, such as fixed illumination, limited vehicle speed and stationary backgrounds. However, for a Number Plate Recognition system to work in today’s world should have the capability to identify Number Plates in dynamic conditions.

4. Objective

The goal is to promote the idea of smart city, automate systems to provide ease to humans, study and implement different object detection methods, evaluate and execute various character recognition techniques, recognize number plates efficiently in dynamic conditions. The prime objective is to build the more accurate model to enhance its applications.

5. Design Methodology

In this process initially, data collected is in the form of video that’s recorded using surveillance camera. Then frames are fetched from the video using OpenCV library. Then image is processed through pre-processing phase where image is transformed into gaussian, threshold and morphological image which is further used to find contours in image which help in detecting possible plate. Then the possible plates are validated to get the desired plate. Now we have to segment our plate number. The input is the image of the plate, we will have to be able to extract the alpha numeric character from images. Segmentation is one of the most important processes for the automatic identification of license plates, because any other step is based on it. If the segmentation fails, recognition phase will not be correct. The detected plate is further is reshaped and character written on it are recognised using model trained by machine learning. The recognition phase is the last step in the development of the automatic number plate recognition system. The recognition must make from the images characters obtained at the end of the segmentation phase. The learning model that will be used for this recognition must be able to read an image and recognise the corresponding character. When the number plate is detected we move forward to test the accuracy using test dataset.

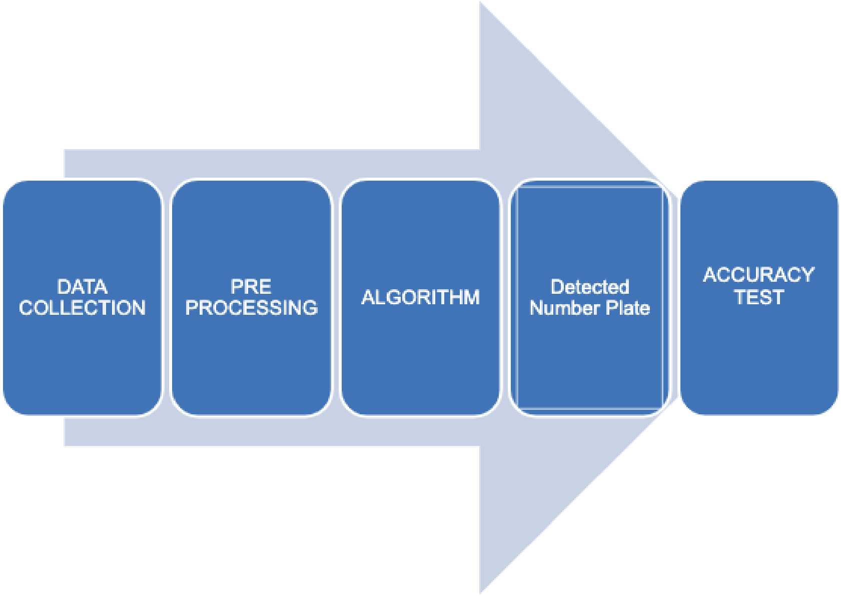


Figure-1(PROCESS FLOW)

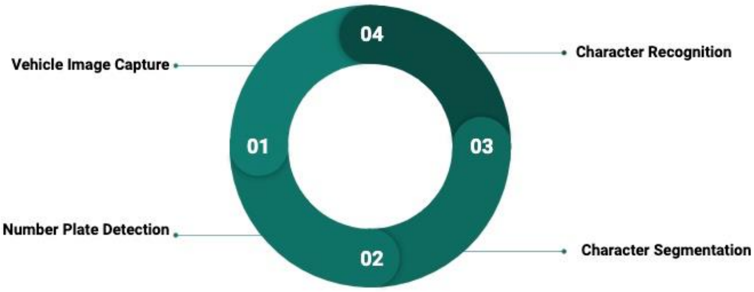


Figure-2(ARCHITECTURE OF NUMBER PLATE RECOGINITION SYSTEM)

6. Implementation

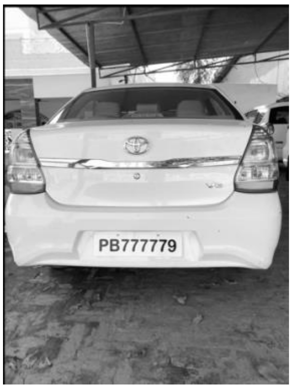
The model is implemented to detect number plate from images which is done by preprocessing image to find contours and validate the contour whether it can be possible plate or not. Further recognize the character from the extracted plate using trained model.

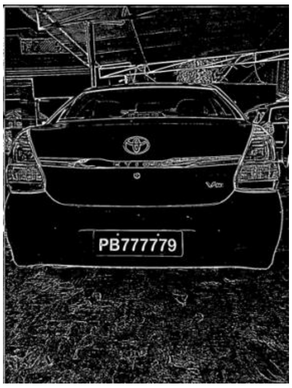
1. Pseudocode

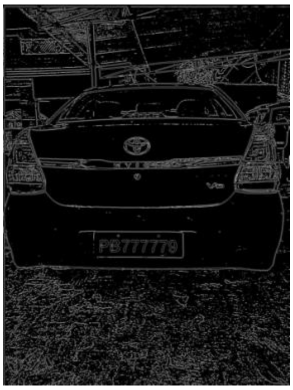
```
a) Import libraries
b) Data = videoCapture( )
c) Processing = preprocess(img)
d) Listofpossibleplate = detectplateinimage(img)
e) Img_reshape = img.resize( )
f) Listofchar = detectcharinplate(Listofpossibleplate)
g) Licplate = listofchar
```

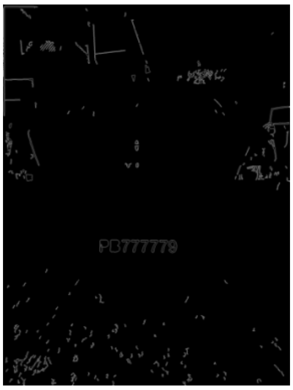
2. Output Screen

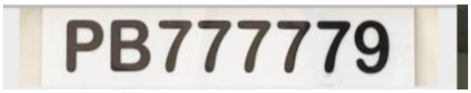
- **Functions Effect on Image:**

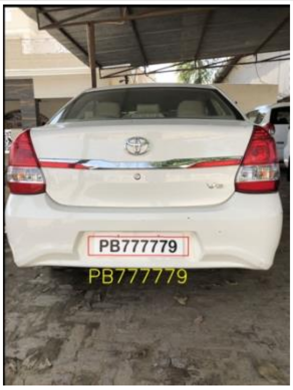












- **Final Result:**

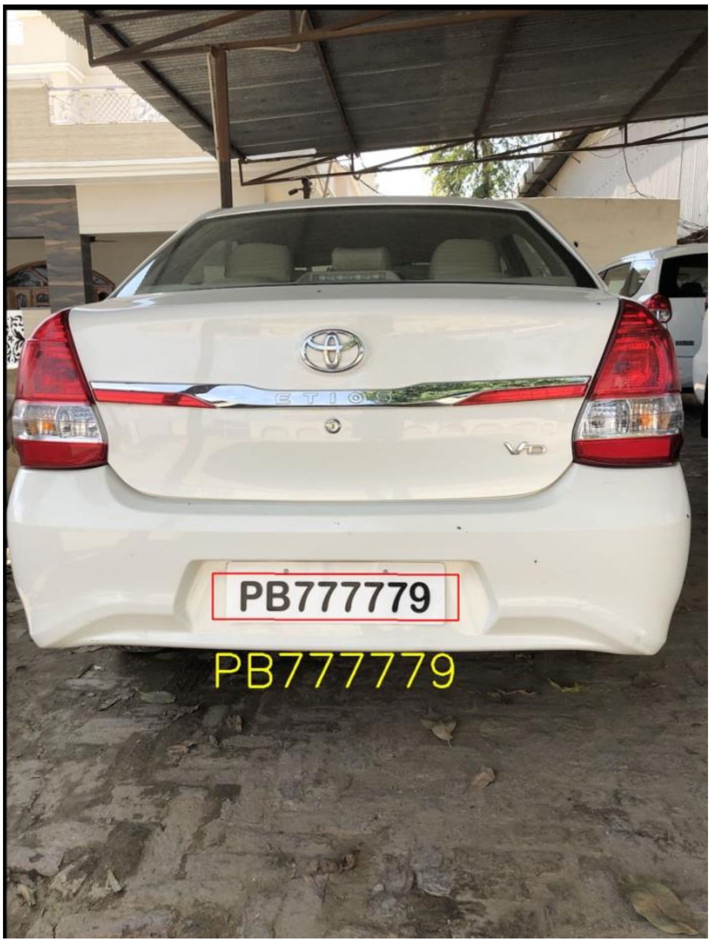


Figure 6.2.1: Original Image

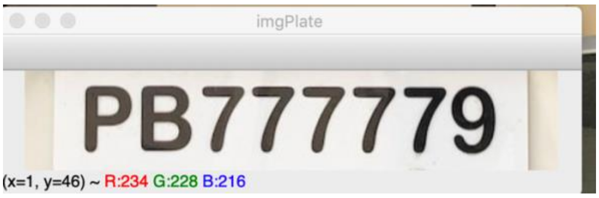


Figure 6.2.2: Plate Image

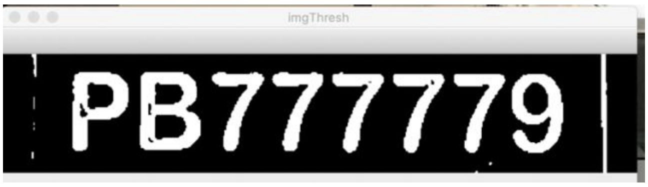
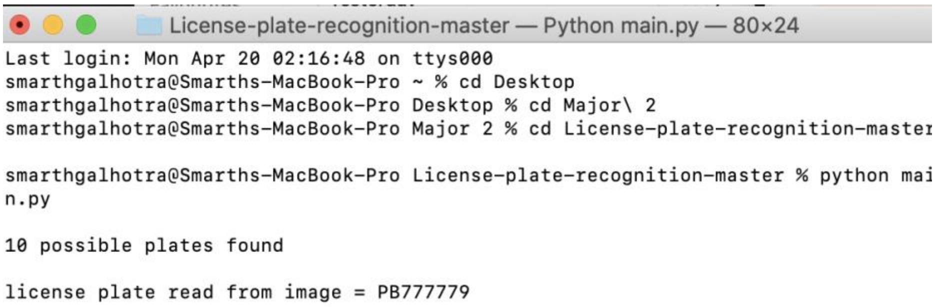


Figure 6.2.3: Plate Threshold Image



```
Last login: Mon Apr 20 02:16:48 on ttys000
smarthgalhotra@Smarths-MacBook-Pro ~ % cd Desktop
smarthgalhotra@Smarths-MacBook-Pro Desktop % cd Major\ 2
smarthgalhotra@Smarths-MacBook-Pro Major 2 % cd License-plate-recognition-master

smarthgalhotra@Smarths-MacBook-Pro License-plate-recognition-master % python mai
n.py

10 possible plates found

license plate read from image = PB777779
```

Figure 6.2.4: License Plate Characters

3. Result Analysis

The model successfully detects license plate as well as recognize its alphabets and numbers from the original data. The algorithm sometimes results in misidentification of characters due to small dataset used, which can be improved by training the algorithm with more dataset.

7. Conclusion and Future Scope

The yearning to learn more about image processing led to this project. We successfully achieved most of the objectives set in the start.

In future, the model can be used for identifying the stolen vehicles, toll tax collection, car registration applications, traffic control, parking management, and for security purposes in society or office campuses.

References

[1] "Automatic Number Plate Recognition System" by Amr Badr etal Mohamed M. Abdelwahab.

[2] "Number Plate Recognition by using open CV-Python" International Research Journal of Engineering and Technology (IRJET)e-ISSN: 2395-0056 by Tella pavani etal DVR Mohan

[3] "Automated Car Number Plate Detection System to detect far number plates" by Anisha Goyal OSR Journal of Computer Engineering (IOSR-JCE)

[4] Blog article "Automatic License Plate Detection & Recognition using deep learning" by Achraf Kharzi

[5] Automatic Number Plate Recognition System (ANPR): A Survey by Chirag Patel etal Dipti Shah International Journal of Computer Applications (0975 – 8887)

[6] Blog - "YOLO — You only look once, real time object detection explained" by Manish Chablani

[7] "Real time license plate recognition using deep learning" Internation Journal of Information Retrieval Research by Saquib Nadeem Hashmi etal Kaushtu

Annexure

```
# Main.py

import cv2
import numpy as np
import os

import DetectChars
import DetectPlates
import PossiblePlate

# Module wide variables
SCALAR_BLACK = (0.0, 0.0, 0.0)
SCALAR_WHITE = (255.0, 255.0, 255.0)
SCALAR_YELLOW = (0.0, 255.0, 255.0)
SCALAR_GREEN = (0.0, 255.0, 0.0)
SCALAR_RED = (0.0, 0.0, 255.0)

showSteps = False


def main():

    KNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()

    if KNNTrainingSuccessful == False:                          # if KNN training was not successful
        print("\nError: KNN traning was not successful\n")
        return


    imgOriginalScene  = cv2.imread("LicPlateImages/car26.jpg")              # open image

    if imgOriginalScene is None:
        print("\nError: image not read from file \n\n")
        os.system("pause")
        return


    listOfPossiblePlates = DetectPlates.detectPlatesInScene(imgOriginalScene)

    listOfPossiblePlates = DetectChars.detectCharsInPlates(listOfPossiblePlates)

    cv2.imshow("imgOriginalScene", imgOriginalScene)

    if len(listOfPossiblePlates) == 0:
        print("\nNo license plates were detected\n")
    else:



        listOfPossiblePlates.sort(key = lambda possiblePlate: len(possiblePlate.strChars), reverse = True)


        licPlate = listOfPossiblePlates[0]

        cv2.imshow("imgPlate", licPlate.imgPlate)
        cv2.imshow("imgThresh", licPlate.imgThresh)
        if len(licPlate.strChars) == 0:
            print("\nNo characters were detected\n\n")
            return


        drawRedRectangleAroundPlate(imgOriginalScene, licPlate)             # draw red rectangle around
plate

        print("\nLicense plate read from image = " + licPlate.strChars + "\n")  # write license plate text to std
out
        print("----------------------------------------")

        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)           # write license plate text on the
image

        cv2.imshow("imgOriginalScene", imgOriginalScene)

        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)           # write image out to file


    cv2.waitKey(0)                                  # hold windows open until user presses a key

    return



def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):

    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)            # get 4 vertices of rotated
rect

    cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR_RED, 2)
# draw 4 red lines
    cv2.line(imgOriginalScene, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), SCALAR_RED, 2)
    cv2.line(imgOriginalScene, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), SCALAR_RED, 2)
    cv2.line(imgOriginalScene, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), SCALAR_RED, 2)



def writeLicensePlateCharsOnImage(imgOriginalScene, licPlate):
    ptCenterOfTextAreaX = 0
    ptCenterOfTextAreaY = 0

    ptLowerLeftTextOriginX = 0
    ptLowerLeftTextOriginY = 0

    sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape
    plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape

    intFontFace = cv2.FONT_HERSHEY_SIMPLEX
    fltFontScale = float(plateHeight) / 30.0
    intFontThickness = int(round(fltFontScale * 1.5))

    textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale, intFontThickness)

    ( (intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight), fltCorrectionAngleInDeg ) =
licPlate.rrLocationOfPlateInScene
```

```
    intPlateCenterX = int(intPlateCenterX)
    intPlateCenterY = int(intPlateCenterY)

    ptCenterOfTextAreaX = int(intPlateCenterX)
    if intPlateCenterY < (sceneHeight * 0.75):
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight * 1.6))
    else:
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight * 1.6))
    # end if

    textSizeWidth, textSizeHeight = textSize

    ptLowerLeftTextOriginX = int(ptCenterOfTextAreaX - (textSizeWidth / 2))
    ptLowerLeftTextOriginY = int(ptCenterOfTextAreaY + (textSizeHeight / 2))

    cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX,
ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR_YELLOW, intFontThickness)


if __name__ == "__main__":
    main()
```

# Preprocess.py

```
import cv2
import numpy as np
import math


GAUSSIAN_SMOOTH_FILTER_SIZE = (5, 5)
ADAPTIVE_THRESH_BLOCK_SIZE = 19
ADAPTIVE_THRESH_WEIGHT = 9


def preprocess(imgOriginal):
    imgGrayscale = extractValue(imgOriginal)

    imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)

    height, width = imgGrayscale.shape

    imgBlurred = np.zeros((height, width, 1), np.uint8)

    imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN_SMOOTH_FILTER_SIZE,
0)

    imgThresh = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, ADAPTIVE_THRESH_BLOCK_SIZE,
ADAPTIVE_THRESH_WEIGHT)

    return imgGrayscale, imgThresh



def extractValue(imgOriginal):
    height, width, numChannels = imgOriginal.shape

    imgHSV = np.zeros((height, width, 3), np.uint8)
    imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)

    imgHue, imgSaturation, imgValue = cv2.split(imgHSV)

    return imgValue



def maximizeContrast(imgGrayscale):

    height, width = imgGrayscale.shape

    imgTopHat = np.zeros((height, width, 1), np.uint8)
    imgBlackHat = np.zeros((height, width, 1), np.uint8)

    structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

    imgTopHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_TOPHAT, structuringElement)
    imgBlackHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_BLACKHAT, structuringElement)

    imgGrayscalePlusTopHat = cv2.add(imgGrayscale, imgTopHat)
    imgGrayscalePlusTopHatMinusBlackHat = cv2.subtract(imgGrayscalePlusTopHat, imgBlackHat)

    return imgGrayscalePlusTopHatMinusBlackHat
```

# **DetectChars.py**

```
import os

import cv2
import numpy as np
import math
import random

import Main
import Preprocess
import PossibleChar


kNearest = cv2.ml.KNearest_create()

MIN_PIXEL_WIDTH = 2
MIN_PIXEL_HEIGHT = 8

MIN_ASPECT_RATIO = 0.25
MAX_ASPECT_RATIO = 1.0

MIN_PIXEL_AREA = 80

MIN_DIAG_SIZE_MULTIPLE_AWAY = 0.3
MAX_DIAG_SIZE_MULTIPLE_AWAY = 5.0

MAX_CHANGE_IN_AREA = 0.5

MAX_CHANGE_IN_WIDTH = 0.8
MAX_CHANGE_IN_HEIGHT = 0.2

MAX_ANGLE_BETWEEN_CHARS = 12.0
```

```
MIN_NUMBER_OF_MATCHING_CHARS = 3

RESIZED_CHAR_IMAGE_WIDTH = 20
RESIZED_CHAR_IMAGE_HEIGHT = 30

MIN_CONTOUR_AREA = 100


def loadKNNDataAndTrainKNN():
    allContoursWithData = []                # declare empty lists,
    validContoursWithData = []

    try:
        npaClassifications = np.loadtxt("classifications.txt", np.float32)
    except:
        print("error, unable to open classifications.txt, exiting program\n")
        os.system("pause")
        return False
    # end try

    try:
        npaFlattenedImages = np.loadtxt("flattened_images.txt", np.float32)
    except:
        print("error, unable to open flattened_images.txt, exiting program\n")
        os.system("pause")
        return False

    npaClassifications = npaClassifications.reshape((npaClassifications.size, 1))

    kNearest.setDefaultK(1)

    kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE, npaClassifications)

    return True


def detectCharsInPlates(listOfPossiblePlates):
    intPlateCounter = 0
    imgContours = None
    contours = []

    if len(listOfPossiblePlates) == 0:
        return listOfPossiblePlates



    for possiblePlate in listOfPossiblePlates:

        possiblePlate.imgGrayscale, possiblePlate.imgThresh = 
Preprocess.preprocess(possiblePlate.imgPlate)

        if Main.showSteps == True: # show steps
            cv2.imshow("5a", possiblePlate.imgPlate)
            cv2.imshow("5b", possiblePlate.imgGrayscale)
            cv2.imshow("5c", possiblePlate.imgThresh)

                # increase size of plate image for easier viewing and char detection
        possiblePlate.imgThresh = cv2.resize(possiblePlate.imgThresh, (0, 0), fx = 1.6, fy = 1.6)
```

```
                # threshold again to eliminate any gray areas
        thresholdValue, possiblePlate.imgThresh = cv2.threshold(possiblePlate.imgThresh, 0.0, 255.0, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

        if Main.showSteps == True: # show steps
            cv2.imshow("5d", possiblePlate.imgThresh)

                # this function first finds all contours, then only includes contours that could be chars (without comparison to other chars yet)
        listOfPossibleCharsInPlate = findPossibleCharsInPlate(possiblePlate.imgGrayscale, possiblePlate.imgThresh)

        if Main.showSteps == True: # show steps
            height, width, numChannels = possiblePlate.imgPlate.shape
            imgContours = np.zeros((height, width, 3), np.uint8)
            del contours[:]                                 # clear the contours list

            for possibleChar in listOfPossibleCharsInPlate:
                contours.append(possibleChar.contour)

            cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)

            cv2.imshow("6", imgContours)

                # given a list of all possible chars, find groups of matching chars within the plate
        listOfListsOfMatchingCharsInPlate = findListOfListsOfMatchingChars(listOfPossibleCharsInPlate)

        if Main.showSteps == True: # show steps
            imgContours = np.zeros((height, width, 3), np.uint8)
            del contours[:]

            for listOfMatchingChars in listOfListsOfMatchingCharsInPlate:
                intRandomBlue = random.randint(0, 255)
                intRandomGreen = random.randint(0, 255)
                intRandomRed = random.randint(0, 255)

                for matchingChar in listOfMatchingChars:
                    contours.append(matchingChar.contour)
                cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen, intRandomRed))
            cv2.imshow("7", imgContours)

        if (len(listOfListsOfMatchingCharsInPlate) == 0):

            if Main.showSteps == True: # show steps
                print("chars found in plate number " + str(
                    intPlateCounter) + " = (none), click on any image and press a key to continue . . .")
                intPlateCounter = intPlateCounter + 1
                cv2.destroyWindow("8")
                cv2.destroyWindow("9")
                cv2.destroyWindow("10")
                cv2.waitKey(0)

            possiblePlate.strChars = ""
            continue                                                # go back to top of for loop

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):
            listOfListsOfMatchingCharsInPlate[i].sort(key = lambda matchingChar: matchingChar.intCenterX)
            listOfListsOfMatchingCharsInPlate[i] = 
removeInnerOverlappingChars(listOfListsOfMatchingCharsInPlate[i])


        if Main.showSteps == True: # show steps
            imgContours = np.zeros((height, width, 3), np.uint8)

            for listOfMatchingChars in listOfListsOfMatchingCharsInPlate:
                intRandomBlue = random.randint(0, 255)
                intRandomGreen = random.randint(0, 255)
                intRandomRed = random.randint(0, 255)

                del contours[:]

                for matchingChar in listOfMatchingChars:
                    contours.append(matchingChar.contour)

                cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen, 
intRandomRed))
            cv2.imshow("8", imgContours)

        intLenOfLongestListOfChars = 0
        intIndexOfLongestListOfChars = 0

        for i in range(0, len(listOfListsOfMatchingCharsInPlate)):
            if len(listOfListsOfMatchingCharsInPlate[i]) > intLenOfLongestListOfChars:
                intLenOfLongestListOfChars = len(listOfListsOfMatchingCharsInPlate[i])
                intIndexOfLongestListOfChars = i

        longestListOfMatchingCharsInPlate = 
listOfListsOfMatchingCharsInPlate[intIndexOfLongestListOfChars]

        if Main.showSteps == True: # show steps
            imgContours = np.zeros((height, width, 3), np.uint8)
            del contours[:]

            for matchingChar in longestListOfMatchingCharsInPlate:
                contours.append(matchingChar.contour)

            cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)

            cv2.imshow("9", imgContours)

        possiblePlate.strChars = recognizeCharsInPlate(possiblePlate.imgThresh, 
longestListOfMatchingCharsInPlate)

        if Main.showSteps == True: # show steps
            print("chars found in plate number " + str(
                intPlateCounter) + " = " + possiblePlate.strChars + ", click on any image and press a key to 
continue . . .")
            intPlateCounter = intPlateCounter + 1
            cv2.waitKey(0)


    if Main.showSteps == True:
        print("\nchar detection complete, click on any image and press a key to continue . . .\n")
        cv2.waitKey(0)

    return listOfPossiblePlates
```

```
def findPossibleCharsInPlate(imgGrayscale, imgThresh):
    listOfPossibleChars = []
    contours = []
    imgThreshCopy = imgThresh.copy()


    imgContours, contours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:                        # for each contour
        possibleChar = PossibleChar.PossibleChar(contour)

        if checkIfPossibleChar(possibleChar):
            listOfPossibleChars.append(possibleChar)

    return listOfPossibleChars


def checkIfPossibleChar(possibleChar):
    if (possibleChar.intBoundingRectArea > MIN_PIXEL_AREA and
        possibleChar.intBoundingRectWidth > MIN_PIXEL_WIDTH and
possibleChar.intBoundingRectHeight > MIN_PIXEL_HEIGHT and
        MIN_ASPECT_RATIO < possibleChar.fltAspectRatio and possibleChar.fltAspectRatio <
MAX_ASPECT_RATIO):
        return True
    else:
        return False


def findListOfListsOfMatchingChars(listOfPossibleChars):
    listOfListsOfMatchingChars = []

    for possibleChar in listOfPossibleChars:
        listOfMatchingChars = findListOfMatchingChars(possibleChar, listOfPossibleChars)

        listOfMatchingChars.append(possibleChar)

        if len(listOfMatchingChars) < MIN_NUMBER_OF_MATCHING_CHARS:
            continue

        listOfListsOfMatchingChars.append(listOfMatchingChars)

        listOfPossibleCharsWithCurrentMatchesRemoved = []

        listOfPossibleCharsWithCurrentMatchesRemoved = list(set(listOfPossibleChars) -
set(listOfMatchingChars))

        recursiveListOfListsOfMatchingChars =
findListOfListsOfMatchingChars(listOfPossibleCharsWithCurrentMatchesRemoved)

        for recursiveListOfMatchingChars in recursiveListOfListsOfMatchingChars:
            listOfListsOfMatchingChars.append(recursiveListOfMatchingChars)

        break
    return listOfListsOfMatchingChars
# end function


def findListOfMatchingChars(possibleChar, listOfChars):
    listOfMatchingChars = []

    for possibleMatchingChar in listOfChars:
        if possibleMatchingChar == possibleChar:

            continue
        fltDistanceBetweenChars = distanceBetweenChars(possibleChar, possibleMatchingChar)

        fltAngleBetweenChars = angleBetweenChars(possibleChar, possibleMatchingChar)

        fltChangeInArea = float(abs(possibleMatchingChar.intBoundingRectArea - 
possibleChar.intBoundingRectArea)) / float(possibleChar.intBoundingRectArea)

        fltChangeInWidth = float(abs(possibleMatchingChar.intBoundingRectWidth - 
possibleChar.intBoundingRectWidth)) / float(possibleChar.intBoundingRectWidth)
        fltChangeInHeight = float(abs(possibleMatchingChar.intBoundingRectHeight - 
possibleChar.intBoundingRectHeight)) / float(possibleChar.intBoundingRectHeight)

                # check if chars match
        if (fltDistanceBetweenChars < (possibleChar.fltDiagonalSize * 
MAX_DIAG_SIZE_MULTIPLE_AWAY) and
            fltAngleBetweenChars < MAX_ANGLE_BETWEEN_CHARS and
            fltChangeInArea < MAX_CHANGE_IN_AREA and
            fltChangeInWidth < MAX_CHANGE_IN_WIDTH and
            fltChangeInHeight < MAX_CHANGE_IN_HEIGHT):

            listOfMatchingChars.append(possibleMatchingChar)

    return listOfMatchingChars                  # return result


# Use Pythagoras Theorem to calculate distance between two chars
def distanceBetweenChars(firstChar, secondChar):
    intX = abs(firstChar.intCenterX - secondChar.intCenterX)
    intY = abs(firstChar.intCenterY - secondChar.intCenterY)

    return math.sqrt((intX ** 2) + (intY ** 2))


# Use basic trigonometry (SOH CAH TOA) to calculate angle between chars
def angleBetweenChars(firstChar, secondChar):
    fltAdj = float(abs(firstChar.intCenterX - secondChar.intCenterX))
    fltOpp = float(abs(firstChar.intCenterY - secondChar.intCenterY))

    if fltAdj != 0.0:
        fltAngleInRad = math.atan(fltOpp / fltAdj)
    else:
        fltAngleInRad = 1.5708

    fltAngleInDeg = fltAngleInRad * (180.0 / math.pi)

    return fltAngleInDeg
```

```
def removeInnerOverlappingChars(listOfMatchingChars):
    listOfMatchingCharsWithInnerCharRemoved = list(listOfMatchingChars)
    for currentChar in listOfMatchingChars:
        for otherChar in listOfMatchingChars:
            if currentChar != otherChar:

                if distanceBetweenChars(currentChar, otherChar) < (currentChar.fltDiagonalSize *
MIN_DIAG_SIZE_MULTIPLE_AWAY):
                    if currentChar.intBoundingRectArea < otherChar.intBoundingRectArea:
                        if currentChar in listOfMatchingCharsWithInnerCharRemoved:
                            listOfMatchingCharsWithInnerCharRemoved.remove(currentChar)
                    else:
                        if otherChar in listOfMatchingCharsWithInnerCharRemoved:
                            listOfMatchingCharsWithInnerCharRemoved.remove(otherChar)

    return listOfMatchingCharsWithInnerCharRemoved


def recognizeCharsInPlate(imgThresh, listOfMatchingChars):
    strChars = ""               # this will be the return value, the chars in the lic plate

    height, width = imgThresh.shape

    imgThreshColor = np.zeros((height, width, 3), np.uint8)

    listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)

    cv2.cvtColor(imgThresh, cv2.COLOR_GRAY2BGR, imgThreshColor)

    for currentChar in listOfMatchingChars:
        pt1 = (currentChar.intBoundingRectX, currentChar.intBoundingRectY)
        pt2 = ((currentChar.intBoundingRectX + currentChar.intBoundingRectWidth),
(currentChar.intBoundingRectY + currentChar.intBoundingRectHeight))

        cv2.rectangle(imgThreshColor, pt1, pt2, Main.SCALAR_GREEN, 2)

                # crop char out of threshold image
        imgROI = imgThresh[currentChar.intBoundingRectY : currentChar.intBoundingRectY +
currentChar.intBoundingRectHeight,
                           currentChar.intBoundingRectX : currentChar.intBoundingRectX +
currentChar.intBoundingRectWidth]

        imgROIResized = cv2.resize(imgROI, (RESIZED_CHAR_IMAGE_WIDTH,
RESIZED_CHAR_IMAGE_HEIGHT))

        npaROIResized = imgROIResized.reshape((1, RESIZED_CHAR_IMAGE_WIDTH *
RESIZED_CHAR_IMAGE_HEIGHT))

        npaROIResized = np.float32(npaROIResized)

        retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k = 1)

        strCurrentChar = str(chr(int(npaResults[0][0])))

        strChars = strChars + strCurrentChar                        # append current char to full string


    if Main.showSteps == True: # show steps ####
        cv2.imshow("10", imgThreshColor)
    return strChars
# DetectPlates.py

import cv2
import numpy as np
import math
import Main
import random

import Preprocess
import DetectChars
import PossiblePlate
import PossibleChar

PLATE_WIDTH_PADDING_FACTOR = 1.3
PLATE_HEIGHT_PADDING_FACTOR = 1.5


def detectPlatesInScene(imgOriginalScene):
    listOfPossiblePlates = []                  # this will be the return value

    height, width, numChannels = imgOriginalScene.shape

    imgGrayscaleScene = np.zeros((height, width, 1), np.uint8)
    imgThreshScene = np.zeros((height, width, 1), np.uint8)
    imgContours = np.zeros((height, width, 3), np.uint8)

    cv2.destroyAllWindows()

    if Main.showSteps == True: # show steps
        cv2.imshow("0", imgOriginalScene)

    imgGrayscaleScene, imgThreshScene = Preprocess.preprocess(imgOriginalScene)

    if Main.showSteps == True: # show steps
        cv2.imshow("1a", imgGrayscaleScene)
        cv2.imshow("1b", imgThreshScene)

    listOfPossibleCharsInScene = findPossibleCharsInScene(imgThreshScene)

    if Main.showSteps == True: # show steps
        print("step 2 - len(listOfPossibleCharsInScene) = " + str(
            len(listOfPossibleCharsInScene)))

        imgContours = np.zeros((height, width, 3), np.uint8)

        contours = []

        for possibleChar in listOfPossibleCharsInScene:
            contours.append(possibleChar.contour)

        cv2.drawContours(imgContours, contours, -1, Main.SCALAR_WHITE)
        cv2.imshow("2b", imgContours)

    listOfListsOfMatchingCharsInScene =
DetectChars.findListOfListsOfMatchingChars(listOfPossibleCharsInScene)

    if Main.showSteps == True: # show steps
```

```
        print("step 3 - listOfListsOfMatchingCharsInScene.Count = " + str(
            len(listOfListsOfMatchingCharsInScene)))  # 13 with MCLRNF1 image

        imgContours = np.zeros((height, width, 3), np.uint8)

        for listOfMatchingChars in listOfListsOfMatchingCharsInScene:
            intRandomBlue = random.randint(0, 255)
            intRandomGreen = random.randint(0, 255)
            intRandomRed = random.randint(0, 255)

            contours = []

            for matchingChar in listOfMatchingChars:
                contours.append(matchingChar.contour)

            cv2.drawContours(imgContours, contours, -1, (intRandomBlue, intRandomGreen,
intRandomRed))

        cv2.imshow("3", imgContours)

    for listOfMatchingChars in listOfListsOfMatchingCharsInScene:
        possiblePlate = extractPlate(imgOriginalScene, listOfMatchingChars)

        if possiblePlate.imgPlate is not None:
            listOfPossiblePlates.append(possiblePlate)

    print("\n" + str(len(listOfPossiblePlates)) + " possible plates found")

    if Main.showSteps == True: # show steps
        print("\n")
        cv2.imshow("4a", imgContours)

        for i in range(0, len(listOfPossiblePlates)):
            p2fRectPoints = cv2.boxPoints(listOfPossiblePlates[i].rrLocationOfPlateInScene)

            cv2.line(imgContours, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), Main.SCALAR_RED, 2)
            cv2.line(imgContours, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), Main.SCALAR_RED, 2)
            cv2.line(imgContours, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), Main.SCALAR_RED, 2)
            cv2.line(imgContours, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), Main.SCALAR_RED, 2)

            cv2.imshow("4a", imgContours)

            print("possible plate " + str(i) + ", click on any image and press a key to continue . . .")

            cv2.imshow("4b", listOfPossiblePlates[i].imgPlate)
            cv2.waitKey(0)

        print("\nplate detection complete, click on any image and press a key to begin char recognition . .
.\n")
        cv2.waitKey(0)

    return listOfPossiblePlates


def findPossibleCharsInScene(imgThresh):
    listOfPossibleChars = []

    intCountOfPossibleChars = 0
    imgThreshCopy = imgThresh.copy()

    imgContours, contours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    height, width = imgThresh.shape
    imgContours = np.zeros((height, width, 3), np.uint8)

    for i in range(0, len(contours)):                        # for each contour

        if Main.showSteps == True: # show steps #######
            cv2.drawContours(imgContours, contours, i, Main.SCALAR_WHITE)

        possibleChar = PossibleChar.PossibleChar(contours[i])

        if DetectChars.checkIfPossibleChar(possibleChar):
            intCountOfPossibleChars = intCountOfPossibleChars + 1
            listOfPossibleChars.append(possibleChar)

    if Main.showSteps == True: # show steps
        print("\nstep 2 - len(contours) = " + str(len(contours)))
        print("step 2 - intCountOfPossibleChars = " + str(intCountOfPossibleChars))
        cv2.imshow("2a", imgContours)

    return listOfPossibleChars



def extractPlate(imgOriginal, listOfMatchingChars):
    possiblePlate = PossiblePlate.PossiblePlate()

    listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)

    fltPlateCenterX = (listOfMatchingChars[0].intCenterX +
listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterX) / 2.0
    fltPlateCenterY = (listOfMatchingChars[0].intCenterY +
listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY) / 2.0

    ptPlateCenter = fltPlateCenterX, fltPlateCenterY

    intPlateWidth = int((listOfMatchingChars[len(listOfMatchingChars) - 1].intBoundingRectX +
listOfMatchingChars[len(listOfMatchingChars) - 1].intBoundingRectWidth -
listOfMatchingChars[0].intBoundingRectX) * PLATE_WIDTH_PADDING_FACTOR)

    intTotalOfCharHeights = 0

    for matchingChar in listOfMatchingChars:
        intTotalOfCharHeights = intTotalOfCharHeights + matchingChar.intBoundingRectHeight

    fltAverageCharHeight = intTotalOfCharHeights / len(listOfMatchingChars)

    intPlateHeight = int(fltAverageCharHeight * PLATE_HEIGHT_PADDING_FACTOR)

    fltOpposite = listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY -
listOfMatchingChars[0].intCenterY
    fltHypotenuse = DetectChars.distanceBetweenChars(listOfMatchingChars[0],
listOfMatchingChars[len(listOfMatchingChars) - 1])
    fltCorrectionAngleInRad = math.asin(fltOpposite / fltHypotenuse)
    fltCorrectionAngleInDeg = fltCorrectionAngleInRad * (180.0 / math.pi)
    possiblePlate.rrLocationOfPlateInScene = ( tuple(ptPlateCenter), (intPlateWidth, intPlateHeight), fltCorrectionAngleInDeg )

    rotationMatrix = cv2.getRotationMatrix2D(tuple(ptPlateCenter), fltCorrectionAngleInDeg, 1.0)

    height, width, numChannels = imgOriginal.shape

    imgRotated = cv2.warpAffine(imgOriginal, rotationMatrix, (width, height))

    imgCropped = cv2.getRectSubPix(imgRotated, (intPlateWidth, intPlateHeight), tuple(ptPlateCenter))

    possiblePlate.imgPlate = imgCropped         # copy the cropped plate image into the applicable member variable of the possible plate

    return possiblePlate

# PossiblePlate.py

import cv2
import numpy as np


class PossiblePlate:

    def __init__(self):
        self.imgPlate = None
        self.imgGrayscale = None
        self.imgThresh = None

        self.rrLocationOfPlateInScene = None

        self.strChars = ""

# PossibleChar.py

import cv2
import numpy as np
import math


class PossibleChar:


    def __init__(self, _contour):
        self.contour = _contour

        self.boundingRect = cv2.boundingRect(self.contour)

        [intX, intY, intWidth, intHeight] = self.boundingRect

        self.intBoundingRectX = intX
        self.intBoundingRectY = intY
        self.intBoundingRectWidth = intWidth
        self.intBoundingRectHeight = intHeight

        self.intBoundingRectArea = self.intBoundingRectWidth * self.intBoundingRectHeight

        self.intCenterX = (self.intBoundingRectX + self.intBoundingRectX + self.intBoundingRectWidth) /
```

```
2
        self.intCenterY = (self.intBoundingRectY + self.intBoundingRectY + self.intBoundingRectHeight) /
2

        self.fltDiagonalSize = math.sqrt((self.intBoundingRectWidth ** 2) + (self.intBoundingRectHeight
** 2))

        self.fltAspectRatio = float(self.intBoundingRectWidth) / float(self.intBoundingRectHeight)
```