INTRODUCTION TO PYTHON

A THESIS

Number Plate Recognition System

Submitted by

K.NANDINI	**CB.EN.U4AIE22030**
SUBASHREE.M	**CB.EN.U4AIE22048**
VALLETI MAHI VIGNESH	**CB.EN.U4AIE22056**
T.LAKSHMAN	**CB.EN.U4AIE22067**

in partial fulfillment for the award of the degree of

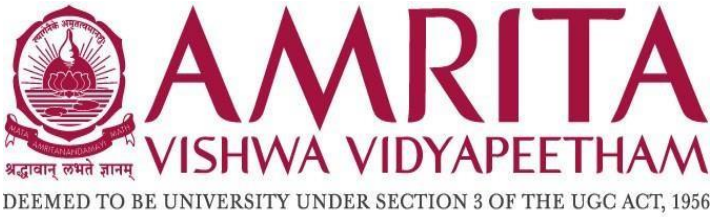
BACHELOR OF TECHNOLOGY
IN
Artificial Intelligence Engineering

Centre for Computational Engineering and Networking, AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAMCOIMBATORE - 641 112 (INDIA)

JULY - 2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112



BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "Indian-Number-Plate-Recognition- System" submitted by **K.NANDINI-CB.EN.U4AIE22030, SUBASHREE.M-CB.EN.U4AIE22048, VALLETI MAHI VIGNESH - CB.EN.U4AIE22056, T.LAKSHMAN- CB.EN.U4AIE22067**, for the award of the Degree of Bachelor ofTechnology in the "CSE(AI) " is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Mrs. Sreelakshmi

Project Guide

Dr. K.P.Soman
Professor and Head CEN

Submitted for the university examination held on _____

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

We **K.NANDINI-CB.EN.U4AIE22030,SUBASHREE.M-CB.EN.U4AIE22048,VALLETI MAHI VIGNESH-CB.EN.U4AIE22056 T.LAKSHMAN- CB.EN.U4AIE22067** hereby declare that this thesis entitled "**Number-Plate-Recognition-System**", is the record ofthe original work done by me under the guidance of Ms. Sreelakshmi K, Assistant Professor, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis forthe award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidatein any University.

Place: Coimbatore
Date:22-06-2023

Signature of the Students

K.NANDINI	**CB.EN.U4AIE22030**
SUBASHREE.M	**CB.EN.U4AIE22048**
VALLETI MAHI VIGNESH	**CB.EN.U4AIE22056**
T.LAKSHMAN	**CB.EN.U4AIE22067**

Contents

List of figures:

ACKNOWLEDGEMENT

This project has been possible due to the sincere and dedicated efforts ofmany. First of all, we would like to thank the dean of our college for giving us the opportunity to get involved in a project and express our skills. We thank our INTRODUCTION TO PYTHON teacher , Ms. Sreelakshmi K for her guidance and support without which this project would have been impossible. Last but not least, we thank our parents and our classmates who encouraged us throughout the project

ABSTARCT:

This project presents an License Plate Recognition system implemented using the OpenCV library. The system employs computer vision techniques to detect and recognize license plates in images, offering practical applications in traffic management and surveillance.

The workflow includes plate detection, character segmentation, and character recognition. The k-nearest neighbors (KNN) algorithm is utilized for character recognition, enhancing the accuracy of the system. Plate detection involves contour analysis and image thresholding to identify potential license plate regions.

Key components of the system include modules for character detection, plate detection, and plate representation. The script emphasizes visual feedback by displaying intermediate processing steps, such as the detection of plates and characters, as well as the final recognized license plate on the original image.

The system provides essential functionality, such as drawing a red rectangle around the detected license plate and writing the recognized characters on the image. Users can visualize the processing steps and input their images for license plate recognition.

Overall, this project demonstrates the effective integration of computer vision algorithms to address challenges in license plate recognition, offering a foundation for potential applications in law enforcement, parking management, and toll collection.

Introduction:

License Plate Recognition system have become integral in modern computer vision applications, playing a crucial role in various fields such as law enforcement, traffic monitoring, and urban planning. This project focuses on the development of an ALPR system using the OpenCV library, aiming to provide an efficient and accurate solution for license plate detection and recognition.

Need for the Project:

Traffic Management:

License Plate Recognition system contribute significantly to traffic management by automating the process of identifying and tracking vehicles. This can aid in monitoring traffic flow, enforcing traffic regulations, and enhancing overall road safety.

Law Enforcement:

In law enforcement, License Plate Recognition technology is a valuable tool for identifying vehicles involved in criminal activities or those violating traffic laws. The system can quickly cross-reference license plate information with databases to aid in investigations.

Surveillance and Security:

License Plate Recognition system enhances surveillance and security measures by providing real-time monitoring of vehicles entering or exiting specific areas. This is particularly beneficial for securing restricted zones or critical infrastructure.

Parking Management:

License Plate Recognition system systems are employed in parking management to automate access control and fee collection. The automated recognition of license plates streamlines the entry and exit process in parking facilities, improving efficiency.

Toll Collection:

The system can be applied in toll collection to automate the identification of vehicles and facilitate seamless transactions. This reduces congestion at toll booths and enhances the overall efficiency of transportation systems.

Efficiency and Accuracy:

License Plate Recognition eliminates the need for manual intervention in tasks that involve vehicle identification. This not only improves efficiency but also reduces the likelihood of human errors.

Technological Advancements:

With the advancements in computer vision and image processing, License Plate Recognition system have become more sophisticated and capable. This project leverages such technological advancements to create a robust and adaptable license plate recognition solution.

OBJECTIVE:

The primary objective of this project is to design and implement a robust License Plate Recognition system using the OpenCV library, with the following specific goals:

1. **License Plate Detection:** Develop algorithms for accurately detecting license plates within given images. Utilize computer vision techniques such as contour analysis and image thresholding to identify potential license plate regions.
2. **Character Segmentation:** Implement methods for segmenting individual characters from the detected license plates. Ensure reliable separation of characters for subsequent recognition.
3. **Character Recognition:** Employ the k-nearest neighbors (KNN) algorithm to recognize characters from the segmented regions of the license plates. Train the model on a dataset of characters for accurate and efficient recognition.
4. **Visual Feedback:** Provide visual feedback by displaying intermediate processing steps, including plate detection, character segmentation, and character recognition. Enable users to visualize the system's decision-making process.
5. **User-Friendly Interface:** Design an intuitive and user-friendly interface allowing users to input images for license plate recognition. Consider incorporating options for users to toggle the display of intermediate steps.
6. **Efficient Display of Results:** Implement functions to draw a red rectangle around the detected license plate and display the recognized characters on the original image. Ensure the clarity and accuracy of the displayed results.
7. **Application in Various Contexts:** Develop the ALPR system to be adaptable and applicable in different contexts such as traffic management, law enforcement, surveillance, parking management, and toll collection.
8. **Accuracy and Reliability:** Prioritize the accuracy and reliability of the ALPR system by addressing challenges such as adverse weather conditions, varying lighting, and potential distortions in license plates.

Methodology:

Step 1: Character recognition:

1. **Loading Data:**
    - Attempts to load pre-trained classification and flattened image data from external files.

2. **Creating K-Nearest Neighbors Model:**
    - Initializes a KNN model and trains it with the loaded flattened image data and corresponding classifications.

3. **Image Preprocessing:**
    - Reads a test image and performs grayscale conversion, Gaussian blur, and adaptive thresholding to create a binary image.

4. **Contour Detection and Analysis:**
    - Identifies contours in the binary image and calculates their properties, including bounding rectangles and areas.

5. **Valid Contour Selection:**
    - Filters out contours with areas below a specified threshold.

6. **Sorting Valid Contours:**
    - Sorts the valid contours based on their x-coordinates.

7. **Character Recognition:**
    - Iterates through the sorted valid contours:
        - Draws a green rectangle around each recognized character on the original test image.

- Extracts the region of interest (ROI) from the binary image.
- Resizes the ROI to a standardized size.
- Uses the KNN model to recognize the character.

8. Result Display:

- Prints the final recognized string of characters.
- Displays the original test image with green rectangles around recognized characters, Waits for a key press and closes all windows.

9. Execution:

- The script is executed when the __main__ block is entered, triggering the overall functionality described above.

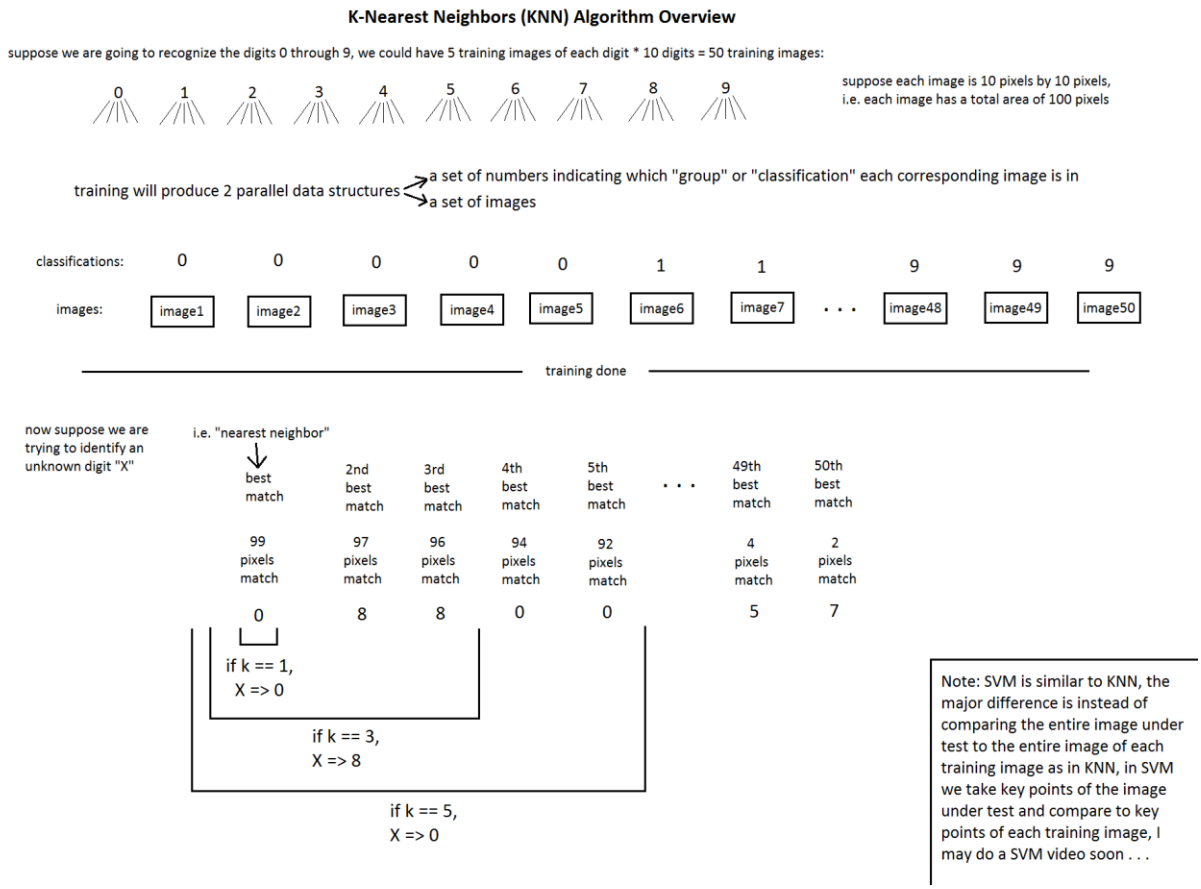


Figure 1 KNN OVERVIEW

Step 2: Number plate detection

1. **KNN Training:**
    - Attempts to load pre-trained K-nearest neighbors (KNN) data for character recognition.

2. **Image Loading:**
    - Reads an image of a license plate from a file.

3. **Plate Detection:**
    - Utilizes a module (DetectPlates) to detect possible license plates in the scene image.

4. **Character Detection:**
    - Uses another module (DetectChars) to detect characters in the detected plates.

5. **Plate Selection and Sorting:**
    - **If plates are detected**:
        - Sorts the list of possible plates based on the number of recognized characters, in descending order.

6. **Plate Display and Character Recognition:**
    - **If plates are detected:**
        - Displays the original scene image.
        - Selects the plate with the most recognized characters as the potential license plate.

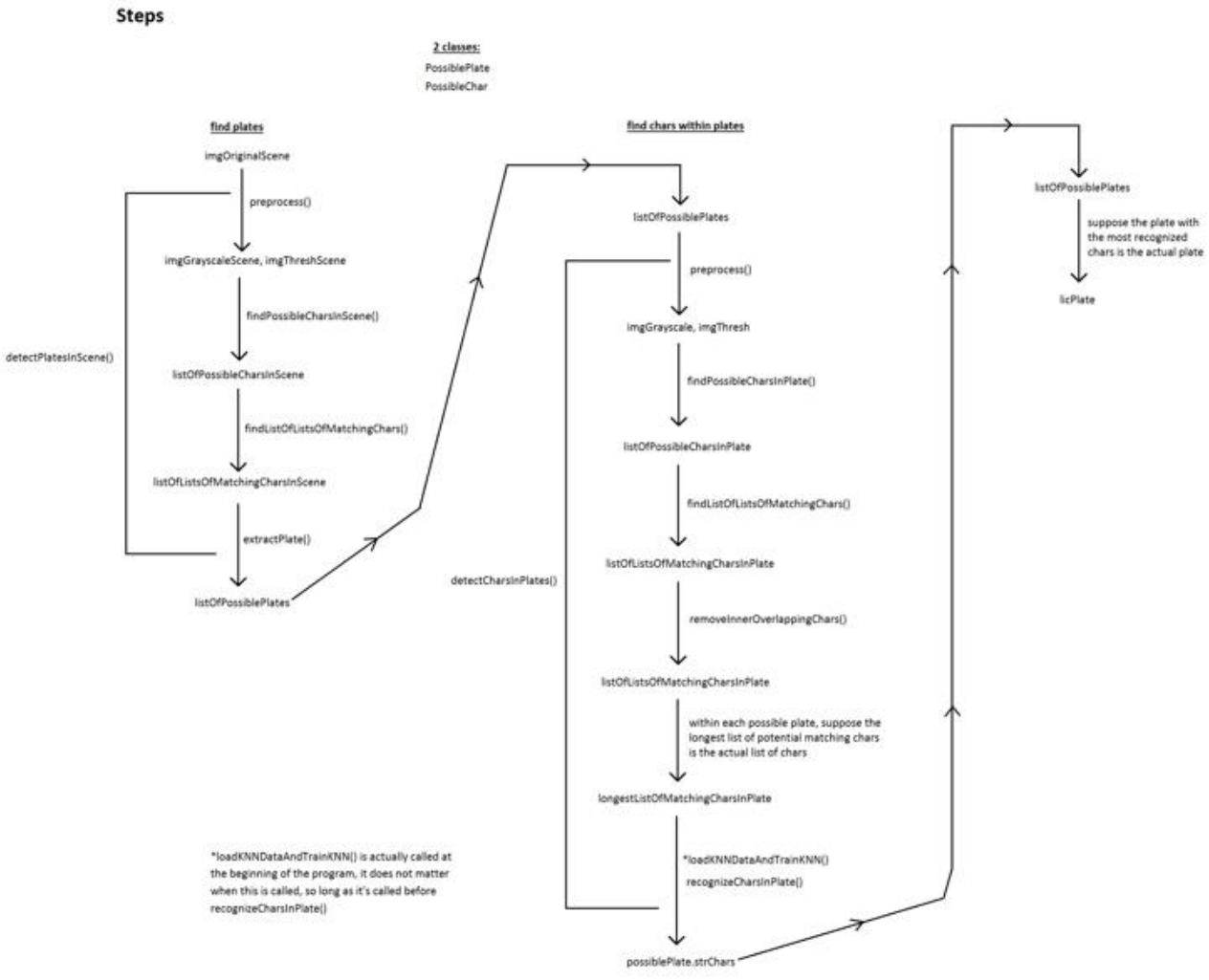
7. **Character Validation:**
    - If characters are recognized in the selected plate:
        - Draws a red rectangle around the selected license plate on the original scene image.
        - Prints the recognized license plate characters.

8. **Character Display on Image:**
    - Writes the recognized license plate characters on the original scene image.

9. **Result Display:**
    - Re-displays the modified scene image with character information.
    - Saves the modified scene image to a file.

10. **User Interaction:**
    - Waits for a key press before closing the windows.

11. **Main Function Execution:**
    - The script is executed when the __main__ block is entered, triggering the main function.



Challenges and solution:

1. **Dealing with bright and dark objects.**

Solution:

Many details present in an RGB image become less salient when we convert it into grayscale. To visualize the change in intensity and thus to deal with bright and dark objects in the image, converting it to grayscale works best

2. **Dealing with noisy images.**

Solution:

We have convolved the image with a Gaussian filter in the pre-processing stage to reduce the noise present in the image.

3. **Dealing with cross-angled or skewed number plates.**

Solution:

De-skewing, i.e. rotating the image to the required position can bedone to detect the text present on the number plate.

4. **Dealing with non-standard number plates.**

Solution:

The code we implemented won't give any results in case of non-standard/ partially torn number plates. Hence, they won't be stored in the database and searching is not possible on it.

CODE FOR LISENCE PLATE RECOGNITION:

train.py

```
import cv2
import numpy as np
import operator
import os

MIN_CONTOUR_AREA = 100
RESIZED_WIDTH = 20
RESIZED_HEIGHT = 30

class ContourWithData():
    npa_contour = None
    bounding_rect = None
    int_rect_x = 0
    int_rect_y = 0
    int_rect_width = 0
    int_rect_height = 0
    flt_area = 0.0

    def calculate_rect_top_left_point_and_width_and_height(self):
        [int_x, int_y, int_width, int_height] = self.bounding_rect
        self.int_rect_x = int_x
        self.int_rect_y = int_y
        self.int_rect_width = int_width
        self.int_rect_height = int_height

    def check_if_contour_is_valid(self):
        if self.flt_area < MIN_CONTOUR_AREA:
            return False
        return True

def main():
    all_contours_with_data = []
    valid_contours_with_data = []

    try:
        npa_classifications = np.loadtxt("C:/Users/mksg0/OneDrive/Desktop/character
recog/classifications.txt", np.float32)
    except:
        print("error, unable to open classifications.txt, exiting program\n")
        os.system("pause")
        return

    try:
        npa_flattened_images = np.loadtxt("C:/Users/mksg0/OneDrive/Desktop/character
recog/flattened_images.txt", np.float32)
    except:
        print("error, unable to open flattened_images.txt, exiting program\n")
        os.system("pause")
        return
```

```
    npa_classifications = npa_classifications.reshape((npa_classifications.size, 1))

    k_nearest = cv2.ml.KNearest_create()

    k_nearest.train(npa_flattened_images, cv2.ml.ROW_SAMPLE, npa_classifications)

    img_testing_numbers = cv2.imread("C:/Users/mksg0/OneDrive/Desktop/character
recog/test3.png")

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

    img_gray = cv2.cvtColor(img_testing_numbers, cv2.COLOR_BGR2GRAY)
    img_blurred = cv2.GaussianBlur(img_gray, (5, 5), 0)

    img_thresh = cv2.adaptiveThreshold(img_blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 11, 2)

    img_thresh_copy = img_thresh.copy()

    npa_contours, npa_hierarchy = cv2.findContours(img_thresh_copy, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    for npa_contour in npa_contours:
        contour_with_data = ContourWithData()
        contour_with_data.npa_contour = npa_contour
        contour_with_data.bounding_rect =
cv2.boundingRect(contour_with_data.npa_contour)
        contour_with_data.calculate_rect_top_left_point_and_width_and_height()
        contour_with_data.flt_area = cv2.contourArea(contour_with_data.npa_contour)
        all_contours_with_data.append(contour_with_data)

    for contour_with_data in all_contours_with_data:
        if contour_with_data.check_if_contour_is_valid():
            valid_contours_with_data.append(contour_with_data)

    valid_contours_with_data.sort(key=operator.attrgetter("int_rect_x"))

    str_final_string = ""

    for contour_with_data in valid_contours_with_data:
        cv2.rectangle(img_testing_numbers, (contour_with_data.int_rect_x,
contour_with_data.int_rect_y),
                      (contour_with_data.int_rect_x + contour_with_data.int_rect_width,
                       contour_with_data.int_rect_y +
contour_with_data.int_rect_height), (0, 255, 0), 2)

        img_roi = img_thresh[contour_with_data.int_rect_y: contour_with_data.int_rect_y
+ contour_with_data.int_rect_height,
```

```
                    contour_with_data.int_rect_x: contour_with_data.int_rect_x +
contour_with_data.int_rect_width]

        img_roi_resized = cv2.resize(img_roi, (RESIZED_WIDTH, RESIZED_HEIGHT))
        npa_roi_resized = img_roi_resized.reshape((1, RESIZED_WIDTH * RESIZED_HEIGHT))
        npa_roi_resized = np.float32(npa_roi_resized)

        retval, npa_results, neigh_resp, dists = k_nearest.findNearest(npa_roi_resized,
k=1)

        str_current_char = str(chr(int(npa_results[0][0])))
        print(f"Recognized Char: {str_current_char}")

        str_final_string = str_final_string + str_current_char

    print("\n" + str_final_string + "\n")

    cv2.imshow("img_testing_numbers", img_testing_numbers)
    cv2.waitKey(0)

    cv2.destroyAllWindows()

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

Explanation:

1. **Imports:**

The script starts by importing necessary libraries like OpenCV (`cv2`), NumPy (`numpy`), `operator`, and `os`.

2. **Constants:**

Defines constant values for minimum contour area, resized width, and height.

3. **ContourWithData` Class:**

Defines a class that holds information about contours (such as contour data, bounding rectangle, area, etc.).

4. **Main Function:**

- Loads data from text files (`classifications.txt` and `flattened_images.txt`) containing training data.

- Trains the KNearest algorithm with the loaded data.
- Reads an image (`test3.png`) for character recognition.
- Preprocesses the image (converts to grayscale, applies Gaussian blur, adaptive thresholding).
- Finds contours in the thresholded image, calculates their properties, and filters out invalid contours based on minimum area criteria.
- Sorts valid contours based on their X-coordinate.
- Iterates through valid contours, extracts regions of interest (ROI), resizes them, and uses the KNearest algorithm to recognize characters.
- Displays recognized characters, constructs a final string of recognized characters, and displays rectangles around recognized characters on the image.
- Displays the resulting image with recognized characters and waits for a key press before closing the window.

Generate_data.py:

```
import sys
import numpy as np
import cv2
import os

MIN_CONTOUR_AREA = 100
RESIZED_WIDTH = 20
RESIZED_HEIGHT = 30

def main():
    img_numbers = cv2.imread("C:/Users/mksg0/OneDrive/Desktop/character
recog/training_chars.png")

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

    img_gray = cv2.cvtColor(img_numbers, cv2.COLOR_BGR2GRAY)
    img_blurred = cv2.GaussianBlur(img_gray, (5, 5), 0)

    img_thresh = cv2.adaptiveThreshold(img_blurred, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)

    cv2.imshow("img_thresh", img_thresh)

    img_thresh_copy = img_thresh.copy()

    npa_contours, _ = cv2.findContours(img_thresh_copy, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    npa_flattened_images = np.empty((0, RESIZED_WIDTH * RESIZED_HEIGHT))

    int_classifications = []

    int_valid_chars = [ord('0'), ord('1'), ord('2'), ord('3'), ord('4'), ord('5'),
ord('6'), ord('7'), ord('8'), ord('9'),
                       ord('A'), ord('B'), ord('C'), ord('D'), ord('E'), ord('F'),
ord('G'), ord('H'), ord('I'), ord('J'),
                       ord('K'), ord('L'), ord('M'), ord('N'), ord('O'), ord('P'),
ord('Q'), ord('R'), ord('S'), ord('T'),
                       ord('U'), ord('V'), ord('W'), ord('X'), ord('Y'), ord('Z')]

    for npa_contour in npa_contours:
        if cv2.contourArea(npa_contour) > MIN_CONTOUR_AREA:
            [x, y, w, h] = cv2.boundingRect(npa_contour)

            cv2.rectangle(img_numbers, (x, y), (x+w, y+h), (0, 0, 255), 2)

            img_roi = img_thresh[y:y+h, x:x+w]
            img_roi_resized = cv2.resize(img_roi, (RESIZED_WIDTH, RESIZED_HEIGHT))

            cv2.imshow("img_roi", img_roi)
            cv2.imshow("img_roi_resized", img_roi_resized)
            cv2.imshow("training_numbers.png", img_numbers)

            int_char = cv2.waitKey(0)

            if int_char == 27:
                sys.exit()
            elif int_char in int_valid_chars:
                int_classifications.append(int_char)

                npa_flattened_image = img_roi_resized.reshape((1, RESIZED_WIDTH *
RESIZED_HEIGHT))
                npa_flattened_images = np.append(npa_flattened_images,
npa_flattened_image, 0)

    flt_classifications = np.array(int_classifications, np.float32)
    npa_classifications = flt_classifications.reshape((flt_classifications.size, 1))

    print("\n\ntraining complete !!\n")

    np.savetxt("classifications.txt", npa_classifications)
    np.savetxt("flattened_images.txt", npa_flattened_images)

    cv2.destroyAllWindows()

    return

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

Explanation:

1. Reading the Training Image:

- The script reads an image (`training_chars.png`) containing characters for training.

2. Preprocessing:

- Converts the image to grayscale.
- Applies Gaussian blur to reduce noise.
- Performs adaptive thresholding to create a binary image.

3. Contour Detection and Character Extraction:

- Finds contours in the thresholded image.
- Filters contours based on area and extracts bounding rectangles around valid contours.
- Displays the extracted regions of interest (ROIs) and waits for user input.

4. User Input for Labeling:

- Displays each extracted character ROI and the original image for user input.
- Waits for the user to press a key.
- If the pressed key corresponds to a valid character (0-9, A-Z), the script saves the labeled character and its flattened image data for training.

5. Data Preparation and Saving:

- Reshapes and prepares the labeled character data and classifications for training.
- Saves the flattened images and corresponding classifications into text files (`flattened_images.txt` and `classifications.txt`, respectively).

6. Termination:

- Closes all opened windows after the process is completed.

OUTPUT:

Characters that are trained:

---

0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

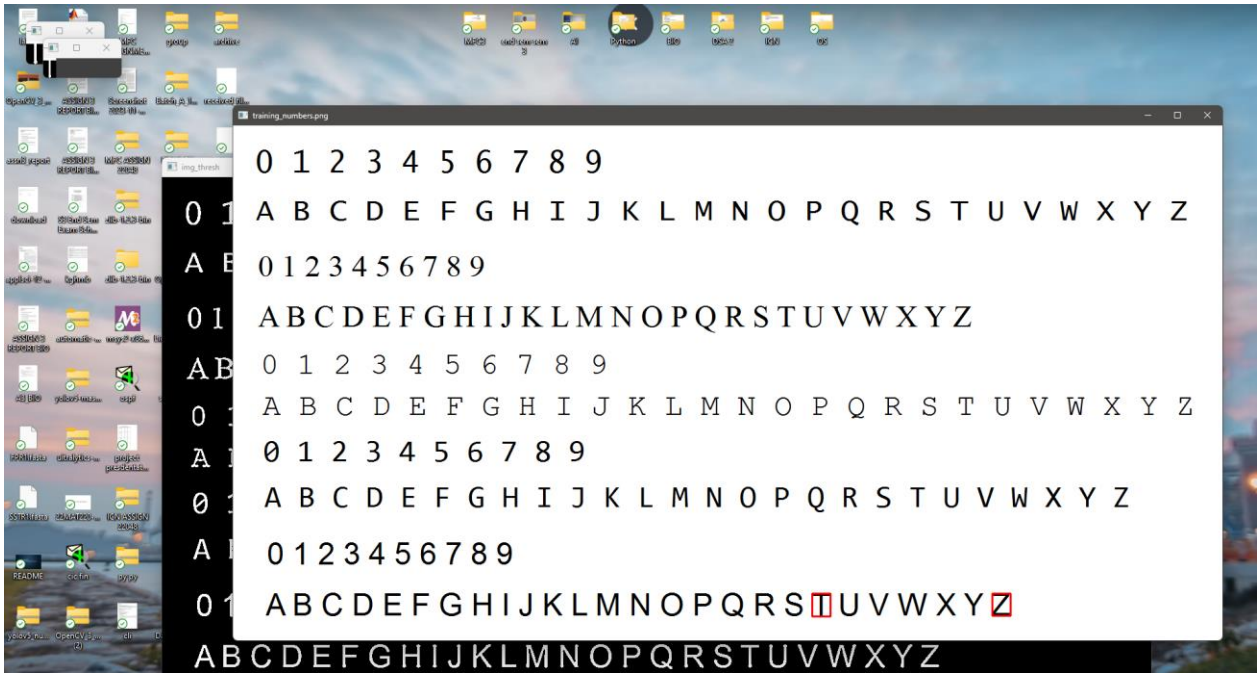
0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Training characters:



Classifications.txt

```
9.0000000000000000000e+01
8.4000000000000000000e+01
8.2000000000000000000e+01
8.0000000000000000000e+01
7.7000000000000000000e+01
7.0000000000000000000e+01
6.9000000000000000000e+01
6.8000000000000000000e+01
6.6000000000000000000e+01
6.5000000000000000000e+01
8.9000000000000000000e+01
8.8000000000000000000e+01
8.7000000000000000000e+01
8.6000000000000000000e+01
8.5000000000000000000e+01
8.3000000000000000000e+01
8.1000000000000000000e+01
7.9000000000000000000e+01
7.8000000000000000000e+01
7.6000000000000000000e+01
7.5000000000000000000e+01
7.4000000000000000000e+01
7.3000000000000000000e+01
7.2000000000000000000e+01
7.1000000000000000000e+01
6.7000000000000000000e+01
5.7000000000000000000e+01
5.6000000000000000000e+01
5.5000000000000000000e+01
5.4000000000000000000e+01
5.3000000000000000000e+01
5.1000000000000000000e+01
5.0000000000000000000e+01
4.8000000000000000000e+01
5.2000000000000000000e+01
4.9000000000000000000e+01
8.2000000000000000000e+01
8.0000000000000000000e+01
```

Flattened images.txt:

```
5.1000000000000000000e+01 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 5.1000000000000000000e+01
5.1000000000000000000e+01 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 5.1000000000000000000e+01
5.1000000000000000000e+01 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.2100000000000000000e+02 3.4000000000000000000e+01
2.7000000000000000000e+01 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02
1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02
1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02 1.3600000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 1.5300000000000000000e+02 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 1.2200000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 6.1000000000000000000e+01 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 7.5000000000000000000e+01 2.4100000000000000000e+02
2.5500000000000000000e+02 2.5500000000000000000e+02 6.8000000000000000000e+01 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 2.3500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 1.2200000000000000000e+02 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 2.5500000000000000000e+02 2.5500000000000000000e+02
2.5500000000000000000e+02 1.0200000000000000000e+02 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
7.0000000000000000000e+00 1.6600000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02
1.7700000000000000000e+02 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00 0.0000000000000000000e+00
```

CODE FOR LISENCE PLATE RECOGNITION:

Main .py

```
import cv2
import numpy as np
import os
import Main
import DetectChars
import DetectPlates
import PossiblePlate

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)

def drawRedRectangleAroundPlate(imgContours, licPlate):
    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)
    p2fRectPoints = np.intp(p2fRectPoints)
    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)

showSteps = False

def main():
    blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()

    if not blnKNNTrainingSuccessful:
        print("\nerror: KNN training was not successful\n")
        return

    imgOriginalScene  =
cv2.imread("C:/Users/mksg0/OneDrive/Desktop/lpr/LicPlateImages/1.png")

    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)
```

```
        print("\nlicense plate read from image = " + licPlate.strChars + "\n")
        print("-----------------------------------------")
        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)
        cv2.imshow("imgOriginalScene", imgOriginalScene)
        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)

    cv2.waitKey(0)
    return

def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):
    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)
    p2fRectPoints = np.intp(p2fRectPoints)
    cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]),
SCALAR_RED, 2)
    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))

    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()
```

Explanation:

- This code implements an Automatic License Plate Recognition (ALPR) system using Python and OpenCV. It's designed to:

- Load necessary libraries (`cv2`, `numpy`, `os`) and custom modules/classes (`Main`, `DetectChars`, `DetectPlates`, `PossiblePlate`).
- Define color scalar constants for use in drawing shapes.
- Provide functions to:
    - Draw a red rectangle around a detected license plate on an image.
    - Write recognized characters from a license plate on the original image near the plate.
- The `main()` function orchestrates the process:
    - Loads KNN data for character recognition.
    - Reads an image of a car scene.
    - Detects possible license plates in the scene.
    - Recognizes characters in the identified plates.
    - Displays the original scene, detected plate, thresholded plate image, and recognized characters.
- Finally, if the script is run directly (`__name__ == "__main__"`), it calls the `main()` function to execute the ALPR process.

- Overall, this code performs a series of operations to identify license plates in images and extract the characters from those plates for further processing.

Detectschars.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 = []
    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

    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

    if len(listOfPossiblePlates) == 0:
        return listOfPossiblePlates

    for possiblePlate in listOfPossiblePlates:
        possiblePlate.imgGrayscale, possiblePlate.imgThresh =
Preprocess.preprocess(possiblePlate.imgPlate)

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

        possiblePlate.imgThresh = cv2.resize(possiblePlate.imgThresh, (0, 0), fx=1.6,
fy=1.6)
        thresholdValue, possiblePlate.imgThresh =
cv2.threshold(possiblePlate.imgThresh, 0.0, 255.0,
                                                                cv2.THRESH_BINARY |
cv2.THRESH_OTSU)

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

        listOfPossibleCharsInPlate =
findPossibleCharsInPlate(possiblePlate.imgGrayscale, possiblePlate.imgThresh)

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

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

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

```
            cv2.imshow("6", imgContours)

        listOfListsOfMatchingCharsInPlate =
findListOfListsOfMatchingChars(listOfPossibleCharsInPlate)

        if Main.showSteps == True:
            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:
            possiblePlate.strChars = ""
            continue

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

        if Main.showSteps == True:
            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:
            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:
            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()

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

    for contour in contours:
        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

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)

        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

def distanceBetweenChars(firstChar, secondChar):
    intX = abs(firstChar.intCenterX - secondChar.intCenterX)
    intY = abs(firstChar.intCenterY - secondChar.intCenterY)

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

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 = ""
    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)

        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

    if Main.showSteps == True:
        cv2.imshow("10", imgThreshColor)

    return strChars
```

Explanation:

It imports necessary libraries: os, cv2, numpy, and modules/classes like Main, Preprocess, PossibleChar.

Constants: Defines various constants used in the character recognition process.

Functions:

- loadKNNDataAndTrainKNN: Loads data for K-nearest neighbors (KNN) algorithm and trains the model.
- detectCharsInPlates: Detects characters within identified license plates.
- findPossibleCharsInPlate: Finds possible characters in a plate based on certain criteria.
- findListOfListsOfMatchingChars: Identifies lists of matching characters in a plate.
- distanceBetweenChars, angleBetweenChars: Calculate distance and angle between characters.
- removeInnerOverlappingChars: Removes overlapping characters within a set of matches.
- recognizeCharsInPlate: Recognizes characters within identified matching characters using KNN.

Overall Functionality:

- The script aims to process images of license plates (listOfPossiblePlates) to identify and recognize characters within them.
- It preprocesses the images (preprocess) and then extracts characters by finding contours (findPossibleCharsInPlate).
- Matching characters are identified based on specific criteria (findListOfListsOfMatchingChars), and the KNN algorithm is used to recognize these characters (recognizeCharsInPlate).

detectsplates.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 = []
    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:
        cv2.imshow("0", imgOriginalScene)

    imgGrayscaleScene, imgThreshScene = Preprocess.preprocess(imgOriginalScene)

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

    listOfPossibleCharsInScene = findPossibleCharsInScene(imgThreshScene)

    if Main.showSteps == True:
        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:
        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:
        print("\n")
        cv2.imshow("4a", imgContours)
        for i in range(0, len(listOfPossiblePlates)):
            p2fRectPoints =
cv2.boxPoints(listOfPossiblePlates[i].rrLocationOfPlateInScene)
            p2fRectPoints = np.intp(p2fRectPoints)
            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()
    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)):
        if Main.showSteps == True:
            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:
        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
    return possiblePlate
```

Explanation:

This is a part of an Automatic License Plate Recognition (ALPR) system using OpenCV in Python. Here's a brief overview:

Imports:

- Libraries like cv2, numpy, math, random.
- Custom modules/classes for preprocessing (Preprocess), character detection (DetectChars), and handling possible plates/characters (PossiblePlate, PossibleChar).
-

Constants:

PLATE_WIDTH_PADDING_FACTOR, PLATE_HEIGHT_PADDING_FACTOR: Factors used for calculating plate width and height based on detected characters.

Functions:

- detectPlatesInScene: Detects possible plates in a given scene by preprocessing the image, finding possible characters, and matching them to form plates.
- findPossibleCharsInScene: Identifies potential characters in a scene based on contours detected in the thresholded image.
- extractPlate: Extracts a possible plate from the original image using a list of matching characters. It calculates the plate's position, dimensions, and rotation angle.

Overall Functionality:

- detectPlatesInScene is the primary function that orchestrates the plate detection process.
- It preprocesses the image, identifies potential characters, matches them, and extracts plates from the scene based on these matches.
- The script visualizes intermediate steps if Main.showSteps is set to True, showing contour detection, possible character identification, and plate extraction.

possible char.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)
```

Possible char.py

Attributes:

- contour: A contour representing a possible character.
- boundingRect: Bounding rectangle around the contour.
- intBoundingRectX, intBoundingRectY: X and Y coordinates of the top-left corner of the bounding rectangle.
- intBoundingRectWidth, intBoundingRectHeight: Width and height of the bounding rectangle.
- intBoundingRectArea: Area of the bounding rectangle.

- intCenterX, intCenterY: X and Y coordinates of the center of the bounding rectangle.
- fltDiagonalSize: Diagonal size of the bounding rectangle.
- fltAspectRatio: Aspect ratio of the bounding rectangle (width/height).

Initialization:

The class is initialized with a contour. Upon initialization, it calculates various properties of the contour such as the bounding rectangle, dimensions, area, center coordinates, diagonal size, and aspect ratio.

This class seems designed to encapsulate the characteristics and properties of a single possible character contour within an image, making it easier to handle and manipulate these properties within the context of character recognition or object detection tasks.

possible plate.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 = ""
```

Explanation:

This script defines a class named `PossiblePlate`.

Class `PossiblePlate`

Attributes:

- `imgPlate`: Represents the image of a possible license plate.
- `imgGrayscale`: Grayscale version of the plate image.
- `imgThresh`: Thresholded version of the plate image.
- `rrLocationOfPlateInScene`: Rotated rectangle location of the plate in the scene.
- `strChars`: String representation of characters found on the plate.

Initialization:

- The class is initialized without any parameters. Upon initialization, it creates placeholders for the plate image, its grayscale and thresholded versions, the location of the plate in the scene (as a rotated rectangle), and an empty string to store the characters recognized on the plate.

- This class is designed to encapsulate the data and properties related to a possible license plate detected in an image. It provides a structured way to store and manipulate information about a potential license plate during the process of automatic license plate recognition or detection.

preprcocess.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
```

Preprocess.py

1. Preprocess Function:
   - Takes an original image as input.
   - Converts the image to grayscale using the `extractValue` function.
   - Maximizes the contrast of the grayscale image using `maximizeContrast` function.
   - Blurs the contrast-maximized image using Gaussian blur.

- Applies adaptive thresholding to create a binary image.
- Returns the grayscale and thresholded images.

2. ExtractValue Function:

- Converts the original image from BGR to HSV color space.
- Extracts the value channel (brightness) from the HSV image.
- Returns the value channel (grayscale image).

3. MaximizeContrast Function:

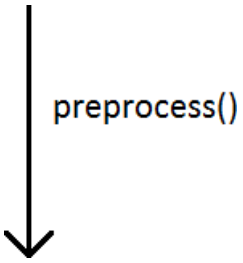
- Performs top-hat and black-hat morphological operations.
- Top-hat: Difference between the input image and its opening.
- Black-hat: Difference between the closing and the input image.
- Adds the input image and top-hat, then subtracts black-hat from it.
- Returns the resulting image with maximized contrast.

The code combines these operations to prepare the image for further processing, likely for tasks like character recognition, edge detection, or feature extraction.

OUTPUT:

Steps With Images

imgOriginalScene



imgGrayscaleScene, imgThreshScene

findPossibleCharsInScene()

all contours (2362 w/MCLRN F1 image)

vectorOfPossibleCharsInScene (131 w/MCLRN F1 image)

findVectorOfVectorsOfMatchingChars()

vectorOfVectorsOfMatchingCharsInScene (13 w/MCLRN F1 image)

extractPlate()

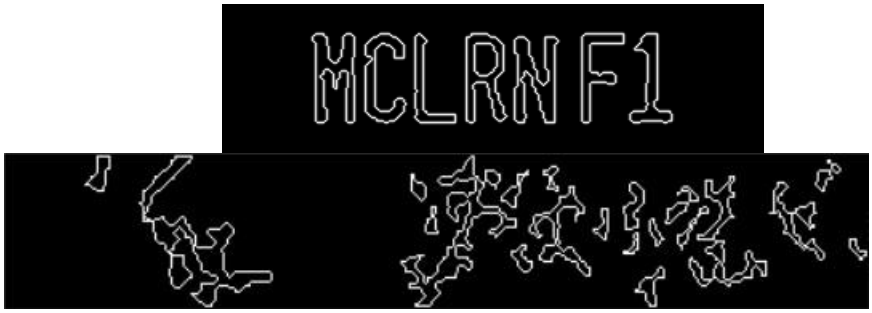
vectorOfPossiblePlates (13 w/MCLRN F1 image)

preprocess()

imgGrayscale, imgThresh

findPossibleCharsInPlate()

vectorOfPossibleCharsInPlate



findVectorOfVectorsOfMatchingChars()

vectorOfVectorsOfMatchingCharsInPlate

removeInnerOverlappingChars()

vectorOfVectorsOfMatchingCharsInPlate

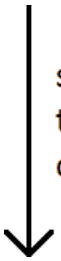
within each possible plate, suppose the longest list of potential matching chars is the actual list of chars

longestVectorOfMatchingCharsInPlate

recognizeCharsInPlate()


```
chars found in plate number 0 = MCLRNF1,
chars found in plate number 5 = I1I,
```

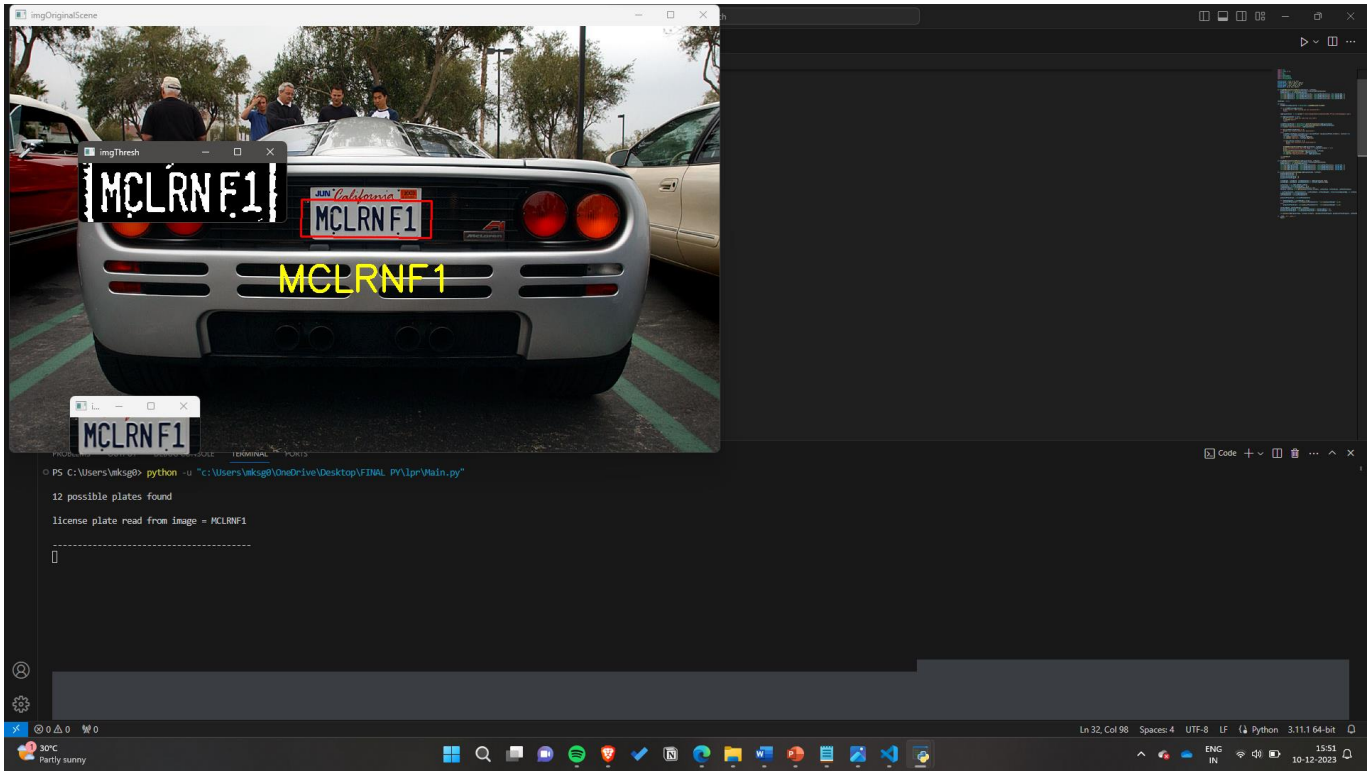
possiblePlate.strChars



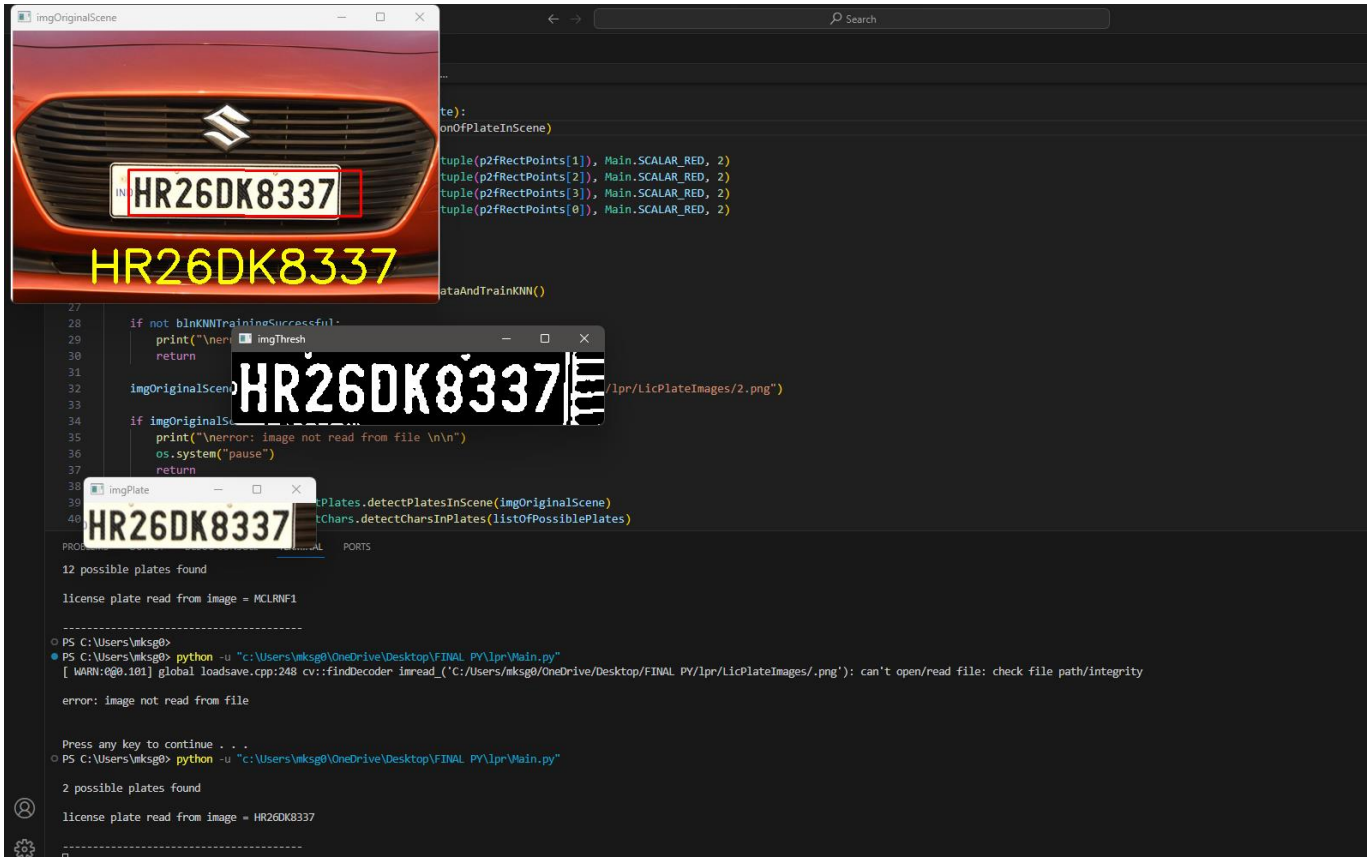
suppose the plate with the most recognized chars is the actual plate

Outputs:

For car1:



For car2:



Conclusion:

In conclusion, the implemented license plate recognition system showcases the integration of computer vision and machine learning for the purpose of automating license plate identification. The key functionalities include plate detection, character recognition, and the visualization of results on the original scene image.

The project leverages the OpenCV library for image processing tasks and employs a KNN model to recognize characters on detected license plates. While the system demonstrates the basic principles of license plate recognition, there are opportunities for further enhancements and refinements.

REFRENCES:

[1] M M Shidore, and S P Narote. (2011) "Number Plate recognition system for Indian Vehicles" International Journal of Computer Science and Network Security 11 (2): 143-146

[2] Sang Kyoon Kim, D. W. Kim and Hang Joon Kim. (1996) "A recognition of vehicle license plate using a genetic algorithm based segmentation," Proceedings of 3rd IEEE International Conference on Image Processing, Lausanne.

[3] https://docs.opencv.org/master/