[Type here]

Number Plate Recognition Using OpenCV

A Project Report submitted for the fulfilments of one of the requirements for the award of the degree of

Bachelor of Technology in Computer Science and Engineering – AIML Specialization

Submitted by

Y. Sai Vijaya Leela(20811A4236)

P. Sahithi Suma(20811A4221)

K. Raj Kumar(208811A4214)

CH. Harsha Vardhan(20811A4202)

CH. Madhav Reddy(20811A4204)

Under the esteemed guidance of

Name of Guide

***(Qualification and position of guide) ***

Department of Computer Science and Engineering

Avanthi Institute of Engineering and Technology (AIET)

Makavarpalem,Tamaram

22-03-2024.

Department of Computer Science and Engineering

Avanthi Institute of Engineering and Technology (AIET)

DECLARATION

We, hereby declare that the project report titled, "**Car and Number Plate Recognition Using OpenCV**" is done solely by our team members in the Department of Computer Science and Engineering, Avanthi Institute of technology and Management, submitted as one of the requirements for awarding of the Degree of Bachelor in Technology in Computer Science and Engineering. This work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration Number	**Name**	**Signature**
20811A4236	Y. Sai Vijaya Leela	
20811A4221	P. Sahithi Suma	
20811A4214	K. Raj Kumar	
20811A4202	CH. Harsha Vardhan	
20811A4204	CH. Madhav Reddy	

Department of Computer Science and Engineering Avanthi Institute of Engineering and Technology

CERTIFICATE

This is to certify that the project titled, "**Car and Number Plate Recognition Using OpenCV**" is bonafide record of the work carried out by "Y. Sai Vijaya Leela(20811A4236), P. Sahithi Suma(20811A4221), K. Raj Kumar(208811A4214), CH. Harsha Vardhan(20811A4202), CH. Madhav Reddy(20811A4204)" for fulfilling one of the requirements for the award of the Degree of Bachelors of Technology in Computer Science and Engineering.

Mrs. Asha Devi, MTech	**N.V. Ashok Kumar**
Assistant Professor	Assistant Professor
(Project Guide)	**(Head of the Department)**

TABLE OF CONTENTS

ABSTRACT

Automatic Number Plate Recognition (ANPR) systems have gained significant traction in various applications, including traffic management, law enforcement, and vehicle tracking. This paper presents an implementation of an ANPR system using Python, OpenCV, and EasyOCR. The system leverages computer vision techniques for image preprocessing, plate localization, and character recognition. OpenCV provides robust functionalities for image processing and manipulation, while EasyOCR offers a simple yet powerful tool for optical character recognition (OCR). Through the integration of these technologies, the ANPR system demonstrates efficient and accurate detection and recognition of license plates in real-world scenarios.

INTRODUCTION

Automatic Number Plate Recognition (ANPR) technology has emerged as a crucial component in modern traffic management and surveillance systems. With advancements in computer vision and machine learning, ANPR systems have become increasingly accurate and reliable. The ability to automatically detect and recognize license plates from images or video streams enables a wide range of applications, including toll collection, parking management, and vehicle identification in law enforcement.

In this paper, we present a Python-based implementation of an ANPR system utilizing two powerful libraries: OpenCV and EasyOCR. OpenCV, an open-source computer vision library, provides a comprehensive set of tools for image processing, feature extraction, and object detection. EasyOCR, on the other hand, offers a user-friendly interface for optical character recognition, capable of recognizing text in various languages with high accuracy.

The proposed ANPR system follows a systematic approach to license plate recognition. It begins with preprocessing techniques to enhance image quality and improve plate visibility. Subsequently, plate localization algorithms are employed to identify candidate regions containing license plates within the image. Finally, EasyOCR is utilized to perform character recognition on the segmented plate regions, extracting the alphanumeric characters comprising the license plate number.

Through the integration of OpenCV and EasyOCR, our ANPR system demonstrates efficient and reliable performance in real-world scenarios. The system's ability to accurately detect and recognize license plates makes it applicable in diverse environments, ranging from traffic surveillance to parking management systems. Moreover, the Python implementation ensures flexibility and ease of integration with existing software frameworks, making it accessible to developers and researchers in the field of computer vision and intelligent transportation systems.

LITERATURE REVIEW

S. No	Title and Authors Names	Description	Algorithms and Technologies used
1	**“Automatic Vehicle License Plate Recognition Using Optimal K-Means with Convolutional neural Networks for Intelligent Transportation Systems” –** Irana Valeryevna Pustokhina, Denis Alexandrovich “Pustokhin, Joel J.P.C. Rodrigues, Deepak Gupta	This paper has presented a new OKMCNN technique for effective detection and recognition of LPs. The OKM-CNN model consists of three stages: localization and detection of localized pixels (LP), OKM-based clustering, and character recognition. It can be used in intelligent infrastructure for toll fee collection, parking management, and traffic surveillance, with potential improvements through bio-inspired optimization algorithm-based parameter tuning.	➢ Optimal K-means (OKM) clustering ➢ Convolutional Neural Network ➢ Improved Bernsen Algorithm (IBA) ➢ Connected Component Analysis (CCA) models. ➢ OKM-clustering with Krill Herd (KH) algorithm.
2	**"Number Plate Recognition using Machine Learning"-** Prasad Molawade, Shruti Shanbhag, Rushabh Rale. **"Artificial Neural Networks based Vehicle license plate recognition"-**H. Erdinc Kocer, K. Kursat Cev	The study presents a supervised K-means - machine learning algorithm for character recognition, which segregates characters into subgroups and uses Support Vector Machine (SVM) to classify them. This method improves accuracy in blurred license plate images and differentiates obstacles due to camera angle, vehicle speed, and surrounding light and shadow.	➢ SVM ➢ K - Means
3	**Automatic number plate recognition using deep learning"-** V Gnanaprakash, N Kanthimathi, N Saranya	The paper presents an automated vehicle tracking system using roadside surveillance cameras and the IMAGEAI framework. The system uses morphological processing for license plate localization, edge detection algorithms for localization, and the IMAGEAI framework for object detection, achieving 97% accuracy	➢ Metal Remove Rate (MRR) ➢ Analysis of variance (ANOVA) ➢ RSM ➢ Teaching Learning Based optimization (TLBO)
4	**"Artificial Neural Networks based Vehicle license plate recognition"-** H. Erdinc Kocer, K. Kursat Cev	The increasing number of vehicles in traffic necessitates the development of computer-based automatic control systems, including automatic vehicle license plate recognition systems. This study presents an efficient automatic vehicle license plate recognition system using artificial neural networks. It uses 259 vehicle pictures, image processing algorithms, and character features to identify license plate regions and classify digitized characters.	➢ Artificial Neural Networks (ANN) ➢ Multi Layered Perceptron (MLP)
5	**"Automatic Vehicle Number Plate Recognition System using Matlab"-** Bhawna	Automatic number plate recognition (ANPR) is a mass surveillance method that uses optical character recognition to read	➢ CRS (Computer Recognition System)

	Tiwari. Archana Sharma, Malti Gautam Singh. Bhawana Rathi	vehicle number plates. The Automatic Number Plate Recognition System (ANPR) uses infrared lighting and flash to capture images and text from number plates, but is region-specific due to plate variations. It's being developed using MATLAB.	➢ OCR (Optical Character Recognition) ➢ MATLAB
6	**“SVM Based License Plate Recognition System"-**Kumar Parasuraman, Member IEEE and Subin P. S	This paper discusses the use of support vector machines (SVMs) in license plate recognition. A number plate recognition algorithm uses supervised learning methods to recognize numbers from number plates. The method improves recognition accuracy and processing speed compared to traditional SVM based multi-class classifiers, making it suitable for real-time applications in dark, poor weather conditions, and broken or dirty images.	➢ Support Vector Machines
7	**“License number plate recognition system using entropy-based features selection approach with SVM" –** Muhammad Attique Khan, Muhammad Sharif, Muhammad Younus Javed, Tallha Akram, Mussarat Yasmin, Tanzila Saba	License plate recognition (LPR) is crucial in security applications like road traffic monitoring and identifying potential threats. The proposed approach addresses light variations, occlusion, and multi-views using luminance channels, binary segmentation, HOG fusion, and SVM feature classification. It performs well, but future work should focus on adding features and improving selection techniques.	➢ Artificial Neural Network (ANN) ➢ optical character recognition (OCR) ➢ SVM ➢ fuzzy-based algorithm
8	**"Automatic Number Plate Recognition using Random Forest Classifier"-** Zuhaib Akhtar, Rashid Ali	Number plate localization, Character segmentation and Character recognition are steps used in the algorithm. The algorithm uses steps such as number plate localization, character segmentation, and character recognition. It converts RGB images to grayscale, removes noise, extracts number plate regions, segments individual characters, and uses Random Forest Classification Algorithm for character recognition, which is best suited for decision trees.	➢ KNN Classification Algorithm ➢ Deep Learning
9	**"Automatic Number Plate Recognition using Machine Learning Algorithms"-**Miss. Shraddha S. Ghadage, Mr. Sagar, R. Khedkar	This project uses computer vision technology to recognize moving vehicle number plate characters using video sequences. It preprocesses features using HOG and classifies each number and alphabet. The aim is to explore challenges in machine vision applications and stimulate knowledge exchange in effective techniques.	➢ K nearest neighbor (KNN). ➢ Multi-classification SVM classifier ➢ Extreme learning Machine (ELM) ANRP
10	**An Automatic Number Plate Recognition System using**	This paper proposes an algorithm optimized for Ghanaian vehicle number plates, using edge detection, feature	➢ Opensource Tesseract OCR engine.

	OpenCV and Tesseract OCR Engine"- Andrew S.Agbemenu, Jepthah Yankey, Ernest O. Addo	detection, and mathematical morphology. Automatic Number Plate Recognition (ANPR) is a widely studied problem, but its universal solution is challenging due to variations in number plate features.	
11	**"Number Plate Recognition without Segmentation"-** Lihong Zheng. Xiangjian He, Qiang Wu and Tom Hintz	The paper introduces a new recognition method using Support Vector Machines (SVMs) for real-time number plate recognition. The algorithm uses multiclass SVMs to classify candidates, achieving higher recognition accuracy and processing speed than traditional SVM-based classifiers. This method offers a new approach for toll charge supervision.	➢ OCR approach ➢ SVM
12	**"Helmet Detection and Number Plate Recognition using Machine Learning" –** Ranveer Roy, Shivam Kumar, Paritosh Dumbhare, Mahesh Barde**.**	Motorcycles are the primary mode of transport in developing countries, leading to increasing motorcycle crashes. Traffic police use CCTV footage to control and penalize helmetless motorcyclists. OCR software and image classification help identify and penalize irresponsible drivers. ensuring a safer environment.	➢ CNN technology ➢ Machine learning algorithms
13	**"Recognition of Vehicle License Plates Using Matlab"-** M. K.B Ashan and N. G. J. Dias	The identification process of a vehicle's license plate is a complex task using MATLAB's predefined functions. This paper proposes a system for recognizing license plates in Sri Lanka using video footages or real-time streams. The process involves detecting the vehicle, isolating the license plate area, and recognizing the license plate using character segmentation and template matching techniques.	➢ SVM ➢ MATLAB ➢ OCRS ➢ LPRS

PROBLEM STATEMENT AND OBJECTIVE

Problem Statement:

The problem lies in the need for an efficient and accurate system for automatic recognition and extraction of license plate information from images or video streams. Traditional manual methods are time-consuming, error-prone, and not scalable for handling large volumes of data. There is a growing demand for automated solutions to enhance traffic management, law enforcement, parking management, toll collection, and vehicle tracking systems.

Objective of the Project:

The objective of the number plate recognition project is to design, develop, and implement a robust system capable of automatically detecting, localizing, and recognizing license plates from images or video feeds in real-time. The system should achieve high accuracy and efficiency in recognizing alphanumeric characters on license plates under various environmental conditions, including different lighting conditions, camera angles, vehicle speeds, and distances.

Key Components and Objectives:

Camera:

- A camera captures images or video footage of vehicles and their license plates.

License Plate Localization:

- Develop algorithms to accurately locate and extract the region of interest containing the license plate within images or video frames.

Image Pre-processing:

This involves tasks such as image enhancement, noise reduction and image segmentation to improve the quality of the captured images and facilitate accurate OCR.

- ➢ Character Segmentation:
  - Implement techniques to segment individual characters or digits on the license plate, ensuring accurate recognition in the subsequent OCR stage.

- Optical Character Recognition (OCR):
  - Utilize machine learning algorithms, neural networks, or deep learning models to perform accurate recognition and extraction of alphanumeric characters from the segmented license plate images.
- Postprocessing:
  - Apply post-processing methods to refine and improve the accuracy of the recognized license plate information.

Real-time Performance:

- Ensure that the system can process images or video frames in real-time, allowing for efficient and timely detection and recognition of license plates.

Robustness and Accuracy:

- Develop the system to be robust against variations in lighting conditions, camera angles, vehicle speeds, and other environmental factors while maintaining high accuracy in license plate recognition.

Integration and Deployment:

- Design the system to be easily integrated into existing infrastructure and deployable in various environments, including roadside cameras, surveillance systems, and traffic management centers.

User Interface:

- Create a user-friendly interface for system configuration, monitoring, and result visualization, allowing users to interact with the system efficiently.

Testing and Evaluation:

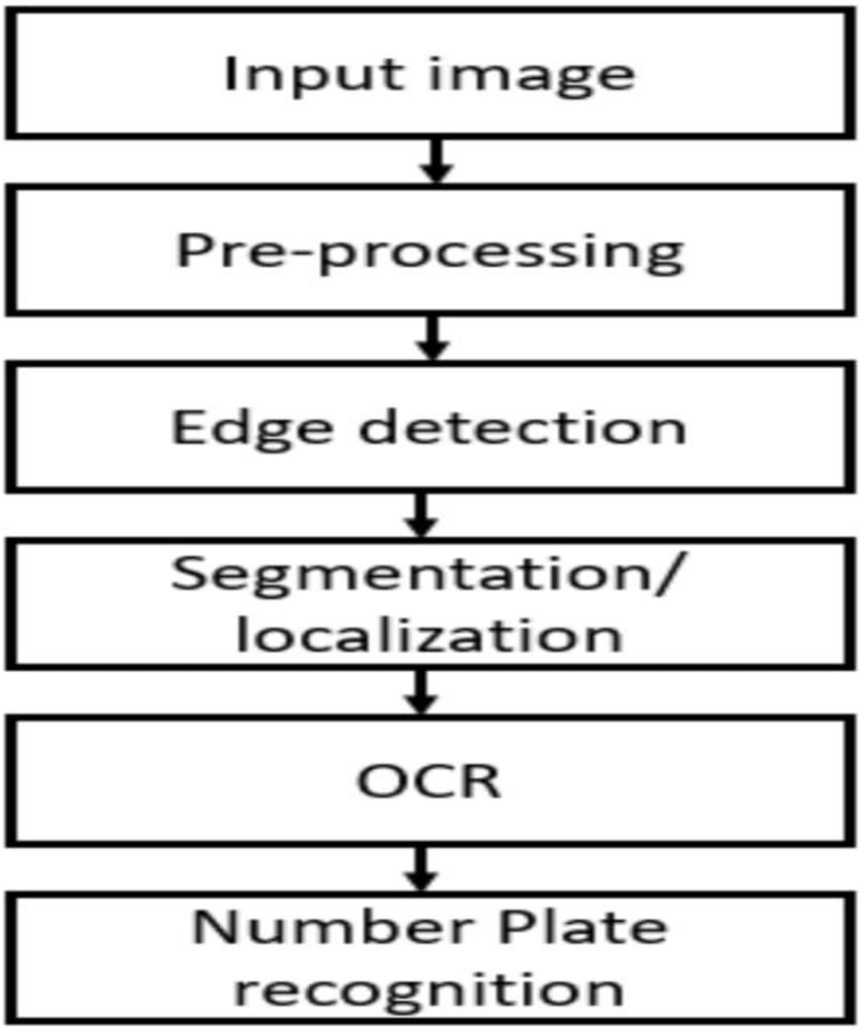
- Conduct rigorous testing and evaluation of the system's performance using diverse datasets and real-world scenarios to assess its accuracy, reliability, and efficiency.

Scalability and Maintenance:

- Ensure that the system is scalable to handle large volumes of data and can be maintained and updated to adapt to changing requirements and technological advancements over time.

The successful implementation of the number plate recognition system will provide valuable capabilities for automating tasks related to traffic management, law enforcement, and security, leading to improved efficiency, safety, and convenience in transportation systems and urban environments.

System Design and Methodology



The projected system is to observe every character from number plate one by one. This could be done by morphological operation. It includes a way to section all the characters employed in the quantity plate. Number plate extraction is that stage wherever vehicle number plate is detected. The detected number plate is pre-processed to get rid of the noise. The divided characters normalized associate degreed passed to an OCR formula.

Ultimately the optical character info is going to be regenerate into encoded text. The characters recognized exploitation template matching. The ultimate output should be within the type of string of characters. In this paper the text found on the vehicle plates is detected from the input image and this requires the localization of number plate area in order to identify the characters present on it. In literature we can find many methods for number plate detection and recognition system.

Input Image

The major drawback is that how long it will take to compute and recognize the particular license plates. This is critical and most needed when it is applied to real time applications. However, there is always a trade-off between computational time and performance rate. In order to achieve an accurate result and increase the performance of the system more computational time is required.

For number plate detection or localization, techniques based on edge statistic and mathematical morphology gives a very good result that uses vertical edge information to calculate the edge density of the image followed by morphology methods such as dilation to extract the region of interest. This technique works well due to the fact that number plates always have high density of vertical edges. But in this method as unwanted edges in the background are also detected which leads to confusion, it is difficult to apply this method for number plates with complex background. Colour based techniques are proposed in this thesis work. The draw back with this method is that it performs well when the lighting condition is constant but when there is various illumination condition its performance reduces. But in real-time application normally the images can be obtained with various lighting illumination. Furthermore, the proposed technique is country specific because each country will have different colour code for vehicle number plate.

since it labels binary image into several components based on their connectivity. Based on the problem one can decide on the selection of finding the connected components using 4-adjacency or 8- adjacency of pixels connectivity. Spatial measurement is a measure of spatial characteristics of a connected component such as area, orientation, aspect ratio etc. and filtering is done to eliminate unrelated or unwanted components. When Connected Component Analysis is combined with spatial measurement and filtering produces better result in number plate detection. Automatic recognition of car license plate number became a very important in our daily life because of the unlimited increase of cars and transportation systems which make it impossible to be fully managed and monitored by humans, examples are so many like traffic monitoring, tracking stolen cars, managing parking toll, red-light violation enforcement, border and customs checkpoints. Yet it’s a very challenging problem, due to the diversity of plate formats, different scales, rotations and non-uniform illumination conditions during image acquisition. This paper presents an approach using simple but efficient morphological operations, filtering and finding connected components for localization of Indian number plates.

Data Pre-Processing

The entries are present in the dataset. The null values are removed using df = df.dropna() where df is the data frame. The categorical attributes (Date,High,Low,Close,Adj value) are converted into numeric using Label Encoder. The date attribute is splitted into new attributes like total which can be used as feature for the model.

1.Data Cleaning: The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

2.DataTransformation: This step is taken in order to transform the data in appropriate forms suitable for mining process.

3.Data Reduction: Since data mining is a technique that is used to handle huge amount of data.

While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we uses data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

Convert color to grayscale

Three algorithms for converting color to grayscale

The lightness method averages the most prominent and least prominent colors: (max (R, G, B) + min R, G, B)) / 2. The average method simply averages the values: (R + G + B) / 3. The luminosity method is a more sophisticated version of the average method

Gaussian filter

Gaussian filter is a linear filter. It's usually used to blur the image or to reduce a noise. A The Gaussian filter alone will blur edges and reduce contrast. The Median filter is a non-linear filter that is most commonly used as a simple way to reduce noise in an image.

Edge Detection

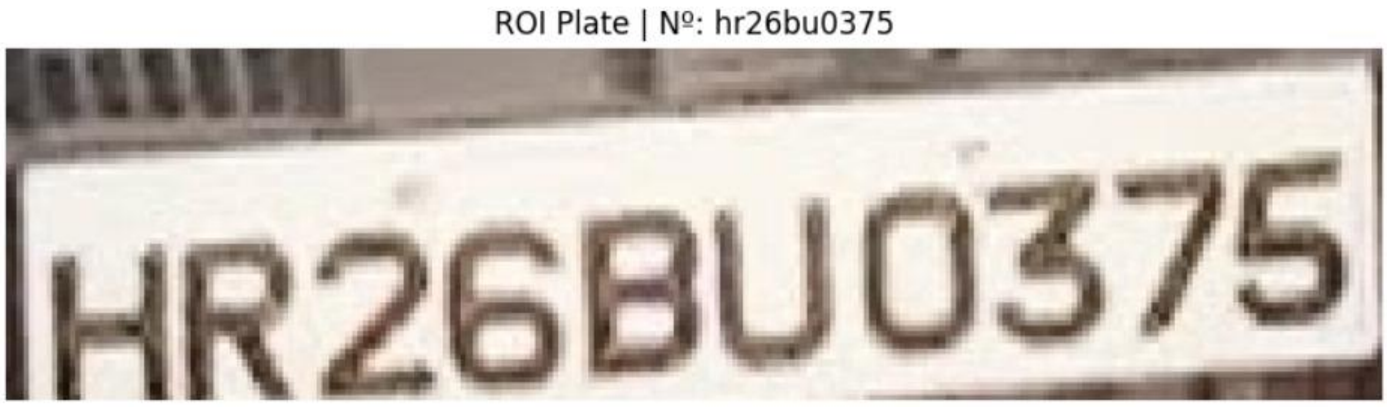
Each image (video frame) has three significant features to achieve detection goals. These features include: edges, contours and points. Among mentioned features, an appropriate option is to use edge pixels. Processing of image pixels enables us to find edge pixels, which are the main features of passing vehicles in a roadway video frame. Edge detection process is demonstrated in a binary image (threshold) with the detected edge pixels. The next step is to extract moving edges from sequential video frames and process the resulting edge information to obtain quantitative geometric measurements of passing vehicles.

Contour

Contour map uses contours or color-coded regions helps us to visualize 3D data in two dimensions. Contour maps are also used to visualize the error surfaces in deep learning/machine learning optimization techniques

Masking

The idea behind masking is to have two additional arrays that record whether an input or output is actually present for a given time step and example, or whether the input/output is just padding



After masking, which involves isolating and extracting the region of interest (ROI) containing the license plate from the rest of the image.

Vehicle ROI Image

Character Segmentation:

The license plate region is further processed to segment individual characters. This involves separating each character from the license plate image to prepare them for recognition.

Optical Character Recognition (OCR):

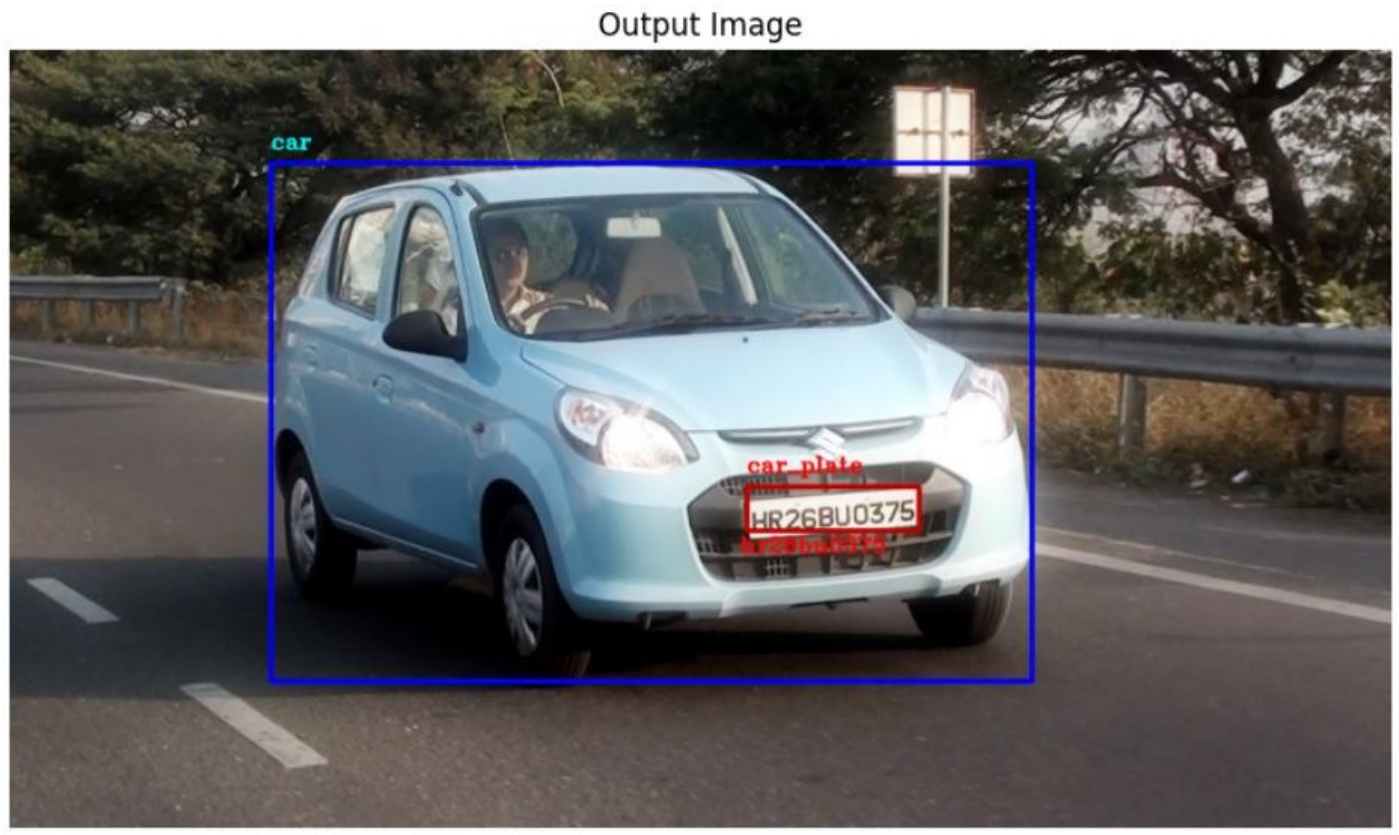
OCR algorithms are applied to each segmented character to recognize and extract the alphanumeric characters present on the license plate. OCR algorithms are trained to interpret the shapes and patterns of characters and convert them into digital text.

License Plate Composition:

The recognized characters are assembled in the correct order to form the complete license plate number. This step may involve rearranging characters based on their positions within the license plate.

Post-Processing and Verification:

Post-processing techniques may be applied to refine the recognized license plate number, such as error correction and character verification against a database of known license plates. Additionally, confidence scores or statistical methods may be used to assess the accuracy and reliability of the recognition results.



By following these steps, number plate recognition systems can accurately and reliably identify license plate numbers from masked images, enabling applications such as law enforcement, parking management, toll collection, and access control.

Overview of Technologies

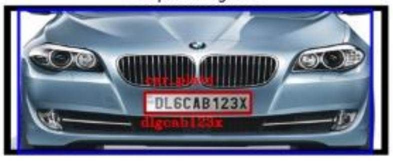
- Image Acquisition: The process starts with capturing images or video streams containing vehicles and their license plates. This can be done using CCTV cameras, surveillance systems, or dedicated cameras installed at specific locations.
- Pre-processing: The acquired images or frames undergo pre-processing techniques to enhance the quality and clarity of the license plate area. Pre-processing steps may include noise reduction, contrast enhancement, and image resizing.
- License Plate Localization: License plate localization involves identifying the region of the image containing the license plate. This is typically done using techniques like edge detection, contour detection, and morphological operations to isolate and extract the license plate area from the rest of the image.
- Character Segmentation: Once the license plate region is localized, character segmentation is performed to separate individual characters or digits on the license plate. This step is crucial for accurate character recognition in the subsequent OCR stage.
- Optical Character Recognition (OCR): OCR is the core technology used to recognize and extract alphanumeric characters from the segmented license plate image. Machine learning algorithms, neural networks, and deep learning models are commonly used in OCR systems to accurately recognize characters under various lighting and environmental conditions.
- Post-processing: After OCR, post-processing techniques may be applied to refine and improve the accuracy of the recognized license plate information. This may involve error correction, character validation, and formatting of the extracted text.
- Database Integration: The recognized license plate information can be integrated with databases containing vehicle registration details, ownership information, or other relevant data. This integration enables applications such as vehicle tracking, toll collection, parking management, and law enforcement.
- Application Development: The recognized license plate data can be utilized in various applications and systems based on project requirements. This may include real-time monitoring systems, traffic management solutions, security systems, and automated billing systems.
- Deployment and Integration: Once the number plate recognition system is developed, it needs to be deployed and integrated into the target environment or infrastructure. This involves

installing cameras, setting up hardware components, configuring software, and testing the system in real-world scenarios.

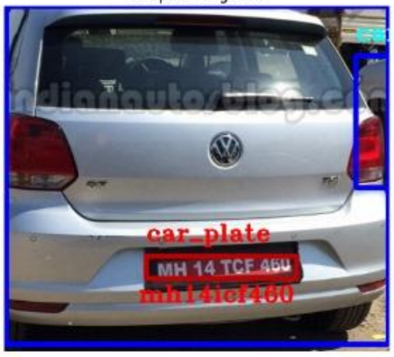
- Performance Evaluation and Optimization: Continuous performance evaluation and optimization are essential to ensure the accuracy, reliability, and efficiency of the number plate recognition system. This may involve fine-tuning algorithms, updating models, and addressing any operational challenges or limitations.

Overall, number plate recognition technology combines various image processing, machine learning, and computer vision techniques to automatically extract and interpret license plate information from images or video streams, enabling a wide range of applications in transportation, security, and surveillance.

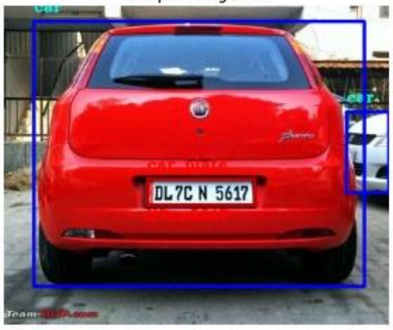
Output Image - 0



Output Image - 1



Output Image - 2



Output Image - 3

Output Image - 4

Output Image - 5

IMPLEMENTATION (CODING AND TESTING)

Implementing number plate recognition involves several steps, including image preprocessing, license plate localization, character segmentation, optical character recognition (OCR), and post-processing. Below is a Python code example using OpenCV and EasyOCR libraries for number plate recognition.

CODE

```
install --upgrade ultralytics -qq

!pip install GPUtil -qq

!pip install easyocr pip

!pip install pytesseract
```

```
!pip install ultralytics

pip install GPUtil

import warnings
warnings.filterwarnings("ignore")

import os
import shutil
import re
import glob
import subprocess
import random
import yaml
import tqdm
import gc


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import seaborn as sns

import IPython.display as display
from IPython.display import Video
from PIL import Image
import cv2

import ultralytics
from ultralytics import YOLO
```

```
import easyocr

import xml.etree.ElementTree as xet
from bs4 import BeautifulSoup

import torch
from GPUtil import showUtilization as gpu_usage
from numba import cuda
import pytesseract

print('ultralytics version: ',ultralytics.__version__)
```

```
class CFG:

    #paths
    out_folder = f'/content/drive/MyDrive/output'
    class_name = ['car_plate']
    video_test_path = 
'https://docs.google.com/uc?export=download&confirm=&id=1pz68D1Gsx80MoPg-_q-
IbEdESEmyVLm-'

    weights = '/content/drive/MyDrive/project/yolov8s.pt'
    exp_name = 'car_plate_detection'
    img_size = (240,400)
    vehicles_class = [2, 3, 5, 7]

    #Yolo train parameters
    epochs = 50
    batch_size = 16
    optimizer = 'auto' # SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, auto
    lr = 1e-5
    lr_factor = 0.01 #lo*lr_f
    weight_decay = 5e-4
    dropout = 0.5
    patience = int(0.7*epochs)
    profile = False
    label_smoothing = 0.0

    #models Confidance
    vehicle_conf = 0.5
    plate_conf = 0.3
    ocr_conf = 0.1

    seed = 42
```

```python
def get_bbox(file_path):
    '''
    This function takes a file path as input.
    It extracts information about the bounding box (coordinates) from the XML file,
specifically the values for xmin, xmax, ymin, and ymax.
    Returns a tuple containing the extracted coordinates (xmin, xmax, ymin, ymax).
    '''
    info = xet.parse(file_path)
    root = info.getroot()
    member_object = root.find('object')

    labels_info = member_object.find('bndbox')
    xmin = int(labels_info.find('xmin').text)
    xmax = int(labels_info.find('xmax').text)
    ymin = int(labels_info.find('ymin').text)
    ymax = int(labels_info.find('ymax').text)

    return xmin, xmax, ymin, ymax



def plot_random_images_from_folder(folder_path, num_images=20, seed=CFG.seed):
    '''
    It randomly selects num_images image files from the specified folder.
    It then plots the image with the bounding box using Matplotlib, arranging the
images in a grid.
    '''
    random.seed(seed)

    # Get a list of image files in the folder
    image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg',
'.png', '.jpeg', '.gif'))]



    # Ensure that we have at least num_images files to choose from
    if len(image_files) < num_images:
        raise ValueError("Not enough images in the folder")

    # Randomly select num_images image files
    selected_files = random.sample(image_files, num_images)

    # Create a subplot grid
    num_cols = 5
    num_rows = (num_images + num_cols - 1) // num_cols
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 8))

    for i, file_name in enumerate(selected_files):
        img = cv2.imread(os.path.join(folder_path, file_name))
        xmin, xmax, ymin, ymax =
get_bbox(file_path=os.path.join('/content/drive/MyDrive/car licencse plate
detection/annotations/',file_name.replace('.png','.xml')))



        start_point = (xmin, ymax)
        end_point = (xmax, ymin)

        img = cv2.rectangle(img, start_point, end_point, (255,0,0), 3)


        if num_rows == 1:
            ax = axes[i % num_cols]
        else:
            ax = axes[i // num_cols, i % num_cols]

        ax.imshow(img)
        ax.axis('off')
        # ax.set_title(file_name)

    # Remove empty subplots
    for i in range(num_images, num_rows * num_cols):
        if num_rows == 1:
            fig.delaxes(axes[i % num_cols])
        else:
            fig.delaxes(axes[i // num_cols, i % num_cols])

    plt.tight_layout()
    plt.show()

def parse_xml_tags(data):
    """Parse xml label file, return image file name, and its coordinates as a
dictionary
    """
    tags = ['filename', 'width', 'height', 'xmin', 'ymin', 'xmax', 'ymax']
    Bs_data = BeautifulSoup(data, "xml")
    d = dict()

    for t in tags:
        text = Bs_data.find(t).text
        if all(c.isdigit() for c in text):
            d[t] = int(text)
        else:
            d[t] = text
    return d
```

```python
def convert_xml_txt_yolo(file_path,w_image,h_image):

    '''
    This function converts XML label information to YOLO format.
    It reads an XML file specified by the file path, extracts bounding box 
coordinates, and converts them to YOLO format.
    Returns a string in YOLO format
    '''
    with open(file_path,  'r') as f:
        label = parse_xml_tags(f.read())


    xmin = int(label['xmin'])
    xmax = int(label['xmax'])
    ymin = int(label['ymin'])
    ymax = int(label['ymax'])


    x_center = float((xmin+((xmax-xmin)/2))/w_image)
    y_center = float((ymin+((ymax-ymin)/2))/h_image)

    width = float((xmax-xmin)/w_image)
    height = float((ymax-ymin)/h_image)

    str_out = f'0 {x_center} {y_center} {width} {height}'

    return str_out

def display_image(image, print_info = True, hide_axis = False, figsize = (15,15), 
title=None):
    '''
    This function displays an image using Matplotlib.
    It takes an image file path or a NumPy array as input.
    It can print information about the image (type and shape), hide axis, and set a 
title.
    '''
    fig = plt.figure(figsize = figsize)
    if isinstance(image, str):  # Check if it's a file path
        img = Image.open(image)

        plt.imshow(img)
    elif isinstance(image, np.ndarray):  # Check if it's a NumPy array
        if image.shape[-1] == 3:
            image = image[..., ::-1]  # BGR to RGB
            img = Image.fromarray(image)
            plt.imshow(img)
        else:
            img = np.copy(image)
            plt.imshow(img,cmap = 'gray')
    else:
        raise ValueError("Unsupported image format")

    if print_info:
        print('Type: ', type(img), '\n')
        print('Shape: ', np.array(img).shape, '\n')

    if hide_axis:
        plt.axis('off')
    if title is not None:
        plt.title(title)

    plt.show()

def create_dir(path):
    '''
    This function creates a directory at the specified path if it doesn't exist.
    '''
    if not os.path.exists(path):
        os.mkdir(path)

def gpu_report():
    '''
    This function provides information about the available GPUs, their properties,
and CUDA version.
    It also prints GPU usage.
    '''
    if torch.cuda.is_available():
        # Get the number of available GPUs
        num_gpus = torch.cuda.device_count()
        print(f"Number of available GPUs: {num_gpus}")

        if num_gpus > 1:
            train_device, test_device = 0,1

        else:
            train_device, test_device = 0,0




        # Get information about each GPU
        for i in range(num_gpus):
            gpu_properties = torch.cuda.get_device_properties(i)
            print(f"\nGPU {i}: {gpu_properties.name}")
            print(f"Total Memory: {gpu_properties.total_memory / (1024**3):.2f}
GB")
            print(f"CUDA Version: {gpu_properties.major}.{gpu_properties.minor}")
    else:
        print("CUDA is not available. You can only use CPU.")
        train_device, test_device = 'cpu', 'cpu'


    print('\n')
    gpu_usage()

    return train_device, test_device


def extract_roi(image, bounding_box):
    """
    Crop the input image based on the provided bounding box coordinates.

    Args:
        image (numpy.ndarray): The input image.
        bounding_box (tuple): A tuple containing (x_min, y_min, x_max, y_max)
            coordinates of the bounding box.

    Returns:
        numpy.ndarray: The cropped image.
    """
    x_min, x_max, y_min, y_max = bounding_box
    cropped_image = image[y_min:y_max, x_min:x_max]
    return cropped_image


def free_gpu_cache() -> None:
    print("Initial GPU Usage")
    gpu_usage()


    torch.cuda.empty_cache()


    print("GPU Usage after emptying the cache")
    gpu_usage()


def extract_ocr(roi_img, reader):
    '''
    This function performs Optical Character Recognition (OCR) on a cropped image
(roi_img) using the specified OCR reader.
    It returns the recognized text (plate number) and confidence level.
    '''
    ocr_result = reader.readtext(np.asarray(roi_img), allowlist =
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ')
    text_plate = ''
    if len(ocr_result) > 0:
        for item in ocr_result:
            text, conf = item[-2], item[-1]
            text = text if conf > CFG.ocr_conf else ''
            text_plate+=text
    else:
        text_plate, conf = 'unreco_plate', 0

    text_plate = text_plate.lower()

    #text_plate = isValidNumberPlate(text_plate)


    return text_plate,conf


def inference_inside_roi(df_coords, img, model, device ,display = False):
    '''
    This function performs object detection on the region of interest (ROI) inside
a given image.
    It takes a DataFrame (df_coords) containing bounding box coordinates, the
image, a detection model, and the device.
    It returns the original image and a DataFrame (df_plate) with information about
the detected plates.
    '''
    bboxs = df_coords[['xmin','xmax','ymin','ymax']].values.astype(int)
    classes = df_coords['class'].values


    df_plate = pd.DataFrame()
    for i,bbox in enumerate(bboxs):

        vehicle_img = extract_roi(img, bbox)

        results = model.predict(vehicle_img,
                        conf = CFG.plate_conf,
                        classes =[0],
                        device = device,
                        verbose = False)

        position_frame = pd.DataFrame(results[0].cpu().numpy().boxes.data,
                                      columns = ['xmin', 'ymin', 'xmax',
                                                 'ymax', 'conf', 'class'])

        position_frame['class'] = position_frame['class'].replace({0:'car_plate'})
        position_frame['plate_number'] = 'unreco_plate'

        #Filter cases with more them one plate per vehicle
        position_frame = position_frame.loc[position_frame['conf'] ==
position_frame['conf'].max(),:]
```

```
        #adjust bbox of plate for complete image
        position_frame['xmin']+=bbox[0]
        position_frame['xmax']+=bbox[0]
        position_frame['ymin']+=bbox[2]
        position_frame['ymax']+=bbox[2]

        if display:
            display_image(vehicle_img, hide_axis = True, figsize =(10,10),
title='Vehicle ROI Image')

        if len(position_frame) > 0:

            plate_bbox =
position_frame[['xmin','xmax','ymin','ymax']].values.squeeze().astype(int)
            plate_img = extract_roi(img, plate_bbox)
            text_plate, conf_ocr = extract_ocr(plate_img, reader)
            position_frame['plate_number'] = text_plate

            if display:
                display_image(plate_img, hide_axis = True, figsize =(10,10),
title=f'ROI Plate | N°: {text_plate}')




        position_frame = position_frame[['xmin', 'ymin', 'xmax','ymax','conf',
'class', 'plate_number']]


        df_plate = pd.concat([df_plate, position_frame], axis = 0)

    return img, df_plate

def drawBBox(df_coords, img, title = '' ,thickness=1):
    '''
    This function draws bounding boxes on the input image based on the coordinates
in the DataFrame (df_coords).
    It also labels the detected objects with their classes and, if applicable, the
plate numbers.
    '''
    cords = df_coords[['xmin','xmax','ymin','ymax']].values.astype(int)
    classes = df_coords['class'].values

    for i,detection in enumerate(cords):


        start_point = (detection[0], detection[-1]) # x_min, y_max
        end_point = (detection[1], detection[2]) # x_max, y_min
        class_detected = classes[i]
        if class_detected == 'car_plate':
            number_plate = df_coords['plate_number'].values[i]
            cv2.rectangle(img, start_point, end_point, (0,0,190), thickness)
            cv2.putText(img=img, text=f'{class_detected} ',
                org= (detection[0], detection[2]-20),
                fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=1, color=(0, 0,
255),thickness=2)
            cv2.putText(img=img, text=f'{number_plate}',
                org= (detection[0]-10, detection[-1]+30),
                fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=1, color=(0, 0,
255),thickness=2)
        else:
            cv2.rectangle(img, start_point, end_point, (255,0,0), thickness)

            cv2.putText(img=img, text=f'{class_detected}',
                org= (detection[0], detection[2]-20),
                fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=1, color=(255, 255,
0),thickness=2)



    return img



anoattions_path_xml = glob.glob('/content/drive/MyDrive/project/car licencse plate
detection/annotations/*.xml')
image_paths = glob.glob('/content/drive/MyDrive/project/car licencse plate
detection/images/*.png')

info = xet.parse(anoattions_path_xml[0])
xet.dump(info)
```

```
index = np.arange(len(anoattions_path_xml))
np.random.shuffle(index)

val_index = index[:50]
# test_index = index[50:100]
train_index = index[50:]
# val_index = np.random.choice(index, size=50, replace=False)

print('Train Size: ', len(train_index))
print('Valid Size: ', len(val_index))
# print('Test Size: ', len(test_index))
```

```
#crete paths for yolo labels
create_dir(CFG.out_folder)
datasets = ['train','valid']
folders = ['images','labels']
```

```
for datset in datasets:
    path_1 = CFG.out_folder + f'/{datset}'
    create_dir(path_1)
    for folder in folders:
        path_2 = CFG.out_folder + f'/{datset}/{folder}'

        create_dir(path_2)

        print(path_2)
for i, img_path in enumerate(image_paths):
    image = cv2.imread(img_path)

    resize_image = cv2.resize(image,CFG.img_size)
    h_image,w_image,_ = image.shape

    label_path = img_path.replace('images','annotations').replace('.png','.xml')
    #print(label_path)


    label_text = convert_xml_txt_yolo(label_path,w_image,h_image)



    text_file_name = img_path.split('/')[-1].replace('.png','.txt')
    img_file_name = img_path.split('/')[-1]
#     print(img_file_name)


    if i in val_index:
        dataset = 'valid'
    elif i in train_index:
        dataset = 'train'
    elif i in test_index:
        dataset = 'test'


    text_path = f'{CFG.out_folder}/' + dataset +'/'+'/labels/' + text_file_name

    new_img_path = f'{CFG.out_folder}/' + dataset +'/images/'+ img_file_name

    shutil.copy2(img_path,new_img_path)
    #cv2.imwrite(new_img_path, resize_image)

    print(text_path)



text_file = open(text_path, "w")
text_file.write(label_text)
text_file.close()
```

```
dict_file = {
    'train': os.path.join(CFG.out_folder, 'train'),
    'val': os.path.join(CFG.out_folder, 'valid'),
    'nc': len(CFG.class_name),
    'names': CFG.class_name
    }

with open(os.path.join('./', 'data.yaml'), 'w+') as file:
    yaml.dump(dict_file, file)

with open('./data.yaml', 'r') as file:
    data_yaml = yaml.safe_load(file)

print(yaml.dump(data_yaml))
```

```
plot_random_images_from_folder(folder_path= r'/content/drive/MyDrive/project/car licencse plate detection/images',
                               num_images=20,
                               seed=CFG.seed)
```

```
! wandb disabled
plate_model = YOLO(CFG.weights)
```

```
train_device, test_device = gpu_report()
plate_model.to(train_device)

print('\nModel Info')
print('Model: ', CFG.weights)
print('Device: ',plate_model.device)
```

```
%%time

### train
plate_model.train(
    data = os.path.join(CFG.out_folder, 'data.yaml'),

    task = 'detect',

    #imgsz = (img_properties['height'], img_properties['width']),

    epochs = CFG.epochs,
    batch = CFG.batch_size,
    optimizer = CFG.optimizer,
    lr0 = CFG.lr,
    lrf = CFG.lr_factor,
    weight_decay = CFG.weight_decay,
    dropout = CFG.dropout,
    patience = CFG.patience,
    label_smoothing = CFG.label_smoothing,
    imgsz = 640,#CFG.img_size,

    name = CFG.exp_name,
    seed = CFG.seed,
    profile = False,

    val = True,
    amp = False,    #mixed precision
    exist_ok = False, #overwrite experiment
    resume = False,
    device = train_device,
    verbose = False,
    single_cls = False,
)
```

```
res_path = CFG.out_folder + '/runs/detect/' + os.listdir(CFG.out_folder +
'/runs/detect/')[-1]
!tree {res_path}
```

```
# plate_model = YOLO('/kaggle/input/best-model-car-plate-detection/best.pt')
vehicle_model = YOLO(CFG.weights)
reader = easyocr. Reader(['en'], gpu=True if test_device! = 'cpu' else False)

plate_model.to(test_device)
vehicle_model.to(test_device)

print('\nModels Info')
print('Plate Model: ', plate_model.device, 'Vehicle Model: ', plate_model.device)
dict_all_classes = vehicle_model.model.names
dict_classes = {}
for id_class in CFG.vehicles_class:
    dict_classes[id_class] = dict_all_classes[id_class]

dict_classes
```

```
test_images = glob.glob('/content/drive/MyDrive/project/Automatic number
plate/images 1/*.jpeg')
```

```
def run_pipeline(path, display=False):

    '''
    1. Detect vehicles from a input image.
    2. Crop the ROIs with BBOX of vehicles detections.
    3. Detect plates from croped vehicle images.
    4. Crop the ROIs with BBOX of plate detections.
    5. Extract the plate number with OCR from croped plate detections.
    '''

    image = cv2.imread(path)

    if display:
        display_image(image,
                      hide_axis =True,
                      figsize = (10,10),
                      title='Input Image')

    #1
    vehicle_results = vehicle_model.predict(image,
                            conf =CFG.vehicle_conf,
                            classes = CFG.vehicles_class,
                            device = test_device,
                            verbose = False,

                                 )

    df_vehicles = pd.DataFrame(vehicle_results[0].cpu().numpy().boxes.data,
                               columns = ['xmin', 'ymin', 'xmax',
                                          'ymax','conf', 'class'])
    df_vehicles['class'] = df_vehicles['class'].replace(dict_classes)


    # 2, 3,4
    image, df_plates = inference_inside_roi(df_vehicles,
                                            image,
                                            plate_model,
                                            test_device,
                                            display = display)
    df_frame = pd.concat([df_vehicles, df_plates], axis = 0).reset_index(drop=True)
    #Draw results in output images
    image = drawBBox(df_frame, image, thickness=5)

    if display:
        display_image(image,
                      hide_axis =True,
                      figsize = (10,10),
                      title='Output Image')

    return df_frame, image
```

```
df_frame, out_image = run_pipeline(path = test_images[11], display=True)
```

```
n_inferences = 15
```

```
num_cols = 3
num_rows = (n_inferences + num_cols - 1) // num_cols
fig,axes = plt.subplots(num_rows,num_cols, figsize=(num_cols*6, num_rows*7))



for i in range(n_inferences):
    df_frame, out_image = run_pipeline(path = test_images[i], display=False)

    if num_rows == 1:
        ax = axes[i % num_cols]
    else:
        ax = axes[i // num_cols, i % num_cols]

    out_image = cv2.cvtColor(out_image, cv2.COLOR_BGR2RGB)
    ax.imshow(out_image)

    ax.axis('off')
    ax.set_title(f'Output Image - {i}')
```

```
def run_pipeline(frame, display=False):
    '''
    1. Detect vehicles from an input image.
    2. Crop the ROIs with BBOX of vehicles detections.
    3. Detect plates from cropped vehicle images.
    4. Crop the ROIs with BBOX of plate detections.
    5. Extract the plate number with OCR from cropped plate detections.
    '''

    if display:
        display_image(frame,
                      hide_axis=True,
                      figsize=(10, 10),
                      title='Input Image')

    # 1
    vehicle_results = vehicle_model.predict(frame,
                                            conf=CFG.vehicle_conf,
                                            classes=CFG.vehicles_class,
                                            device=test_device,
                                            verbose=False)

    df_vehicles = pd.DataFrame(vehicle_results[0].cpu().numpy().boxes.data,
                               columns=['xmin', 'ymin', 'xmax',
                                        'ymax', 'conf', 'class'])
    df_vehicles['class'] = df_vehicles['class'].replace(dict_classes)

    # 2, 3,4
    _, df_plates = inference_inside_roi(df_vehicles,
                                        frame,
                                        plate_model,
                                        test_device,
                                        display=display)
    df_frame = pd.concat([df_vehicles, df_plates], axis=0).reset_index(drop=True)


    # Draw results in output images
    frame = drawBBox(df_frame, frame, thickness=5)

    if display:
        display_image(frame,
                      hide_axis=True,
                      figsize=(10, 10),
                      title='Output Image')

    return df_frame, frame
```

```
import cv2
import pandas as pd

def process_video(input_video_path, output_video_path, output_csv_path):
    # Open the input video file
    cap = cv2.VideoCapture(input_video_path)

    # Get video properties
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # Create VideoWriter for output video
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out_video = cv2.VideoWriter(output_video_path, fourcc, fps, (width, height))

    # Create DataFrame to store results
    columns = ['Frame', 'Class', 'Plate Number', 'xmin', 'ymin', 'xmax', 'ymax']
    results_df = pd.DataFrame(columns=columns)

    frame_count = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Run the pipeline on the current frame
        df_frame, output_image = run_pipeline(frame, display=False)
        # Append the results to the DataFrame
        df_frame['Frame'] = frame_count
        results_df = pd.concat([results_df, df_frame], ignore_index=True)

        # Write the processed frame to the output video
        out_video.write(output_image)

        frame_count += 1

    # Release the video capture and writer objects
    cap.release()
    out_video.release()

    # Save the results DataFrame to CSV
    results_df.to_csv(output_csv_path, index=False)

if __name__ == "__main__":
    input_video_path = '/content/drive/MyDrive/project/car number plate
videovideo/Automatic Number Plate Recognition (ANPR) _ Vehicle Number Plate
Recognition (1).mp4'
    output_video_path = '/content/drive/MyDrive/output/video_output.avi'
    output_csv_path = '/content/drive/MyDrive/output/results.csv'

process_video('/content/drive/MyDrive/project/car number plate videovideo/pexels-
casey-whalen-6571483 (2160p).mp4', '/content/drive/MyDrive/output/Video1.avi',
'/content/drive/MyDrive/output/Video1.csv')
#process_video('/kaggle/input/car-number-plate-video/pexels-george-morina-5222550
(2160p).mp4', '/kaggle/working/Video2.avi', '/kaggle/working/Video2.csv')


  !ffmpeg -i /content/drive/MyDrive/output/Video1.avi -c:v libx264 -crf 23 -c:a aac
-q:a 100 -y /content/drive/MyDrive/output/Video1.mp4


from IPython.display import HTML
from base64 import b64encode

def play(filename):
    html = ''
    video = open(filename,'rb').read()
    src = 'data:video/mp4;base64,' + b64encode(video).decode()
    html += '<video width=1000 controls autoplay loop><source src="%s"
type="video/mp4"></video>' % src
    return HTML(html)

play('/content/drive/MyDrive/output/Video1.avi')
#play('/kaggle/working/Video2.mp4')
#play('/kaggle/working/Video3.mp4')
#play('/kaggle/working/Video4.mp4')
```

```
#play('/kaggle/working/Video5.mp4')
```

Replace 'car_image.jpg' with the path to the image containing the car and its number plate. This code will perform number plate recognition on the provided image using EasyOCR for character recognition. It detects the number plate, extracts the region of interest, recognizes characters, and displays the result with the detected number plate and recognized number

TESTING

Testing number plate recognition typically involves evaluating the accuracy, speed, and robustness of the system under various conditions. Here's how you can test the number plate recognition system:

1. Accuracy Testing:

• Use a dataset of images with ground truth annotations to measure the accuracy of the recognition system. Compare the recognized plate numbers with the ground truth to calculate metrics such as precision, recall, and F1 score.

• Include images with different lighting conditions, angles, distances, and vehicle speeds to test the system's robustness.

2. Real-time Performance Testing:

• Test the system's performance on live video streams to ensure it can process frames in real-time.

• Measure the processing time per frame to assess the system's speed and efficiency.

3. Robustness Testing:

• Test the system's robustness by introducing noise, blur, occlusion, and other disturbances in the images or video streams.

• Evaluate how well the system handles challenging scenarios such as partial occlusion of the license plate, reflections, and varying background textures.

4. Environmental Testing:

• Test the system in different environmental conditions such as day and night, indoor and outdoor environments, and varying weather conditions (e.g., rain, fog).

• Assess the system's performance under different lighting conditions, including bright sunlight, shadows, and artificial lighting.

5. Scale Testing:

• Test the system's scalability by increasing the volume of data, such as processing a large number of images or video frames.

• Evaluate how well the system performs when processing data from multiple cameras simultaneously or in parallel.

6. User Acceptance Testing:

• Gather feedback from end-users or stakeholders to assess the system's usability, user interface, and overall satisfaction.

• Address any usability issues or concerns raised by the users to improve the system's usability and acceptance.

7. Integration Testing:

- Test the system's integration with other software components or external systems, such as database management systems or surveillance networks.
- Ensure seamless communication and data exchange between the number plate recognition system and other modules or applications.

8. Performance Optimization Testing:

- Identify performance bottlenecks and areas for optimization by profiling the system's resource usage (e.g., CPU, memory, disk I/O).
- Optimize algorithms, data structures, and processing pipelines to improve the system's overall performance and efficiency. By conducting comprehensive testing across these areas, you can ensure that the number plate recognition system meets the desired performance, accuracy, and reliability requirements for its intended use cases.

RESULTS AND DISCUSSIONs

This proposed approach for number plate extraction work well for all types of input images. Total 70 vehicle images are tested. Images are taken in different illumination conditions. The images are taken at different distances relative to camera and are of different colors and different size images. The proposed method works well for low contrast, noisy and low-resolution images.

In this paper, the license plate location is the first step in this system of license plate recognition. Therefore, this step play an extremely role and it can affect the accuracy of following steps directly.

In this step, the quality of the edge detection of plate affects the process of plate location in the morphological algorithm. Then, the binaryzation is of great importance in cutting the character out from the license plate.

If the above operations are well done, the features of license plate will be clear in order to improve accuracy of the character matching

On the other hand, if the plates are in the different angle to the observer, it is necessary to adjust the angle in order to 28 facilitate the characters matching.

Matching template is the last step in this programming. We prepared a lot of relevant templates including the test characters and test numbers. In this step, characters are matched with templates by the black points‟ calculation algorithm. The advantage is that is can improve the accuracy of matching. Oppositely, the disadvantage of this method is that it may lengthen the time of matching.

Typically, this system can also be applied to collect the information of vehicles on road, since it can recognize the character and output the license plate number automatically. Obviously, the accuracy of the recognition is the most important in this system. Therefore, this application should be optimized and modified for overcoming the accuracy limitations.

In order to make the recognition more precise, we should add some preprocesses to remove the interferences. Moreover, we would continue the further study for license plate recognition in some complicated environments, such as vehicles at dark night or in heavy rains and so on. If we could accomplish all of the objectives, this application would have a very promising future.

CONCLUSIONS AND FUTURE SCOPE

CONCLUSIONS

License Plate Recognition was a computer system that recognizes any digital image automatically on the number plate. This system includes various operations such as taking pictures, localizing the number pad, truncating characters and OCR from alphanumeric characters. The main conclusion of this system is to design and develop effective image processing techniques and algorithms to localize the license plate in the captured image, to divide the characters from that number plate and to identify each character of the segment by using the Open Computer Vision Library. This has been implemented in CNN algorithm and python programming language. Many applications can be implemented by using this system, such as security, highway speed detection, violation of light, identification of handwritten text, discovery of stolen cars, automatic fee collection systems.

In this work, we have presented technique to recognize number plate of vehicles. For this, we introduced Image capture, preprocessing, edge detection, segmentation, character resizing, feature extraction and finally recognized character of number plate using machine learning algorithms. Dataset creation consisted number of images which are collected real times, parking and etc.

FUTURE WORK

Future work lies in producing more accurate results with lesser response time according to the prescribed specifications of vehicle number plates and automated system software is to be developed in future work.

REFERENCES

Here are some references for number plate recognition:

1. Research Papers:

• Sobhy, A., & El Kader, M. A. (2018). Automatic license plate recognition (ALPR): a state-of-the-art review. International Journal of Intelligent Transportation Systems Research, 16(1), 1-29.

• Zhen, Q., Zhang, Z., Wei, Z., & Han, J. (2017). License Plate Recognition Based on Deep Learning Method. Journal of Physics: Conference Series, 839(1), 012045.

• Suen, C. Y., & Lam, L. (1997). Automatic recognition of characters in scene images: A survey. Pattern Recognition, 30(3), 441-467.

2. Books:

• Khan, S., & Muazzam, A. (2020). Automatic Number Plate Recognition using Image Processing Techniques. LAP LAMBERT Academic Publishing.

• Coello, E. M. R., & Pineda, L. E. V. (2018). Deep Learning for Automatic License Plate Recognition Using ESP32. Academic Press.

3. Online Resources:

• OpenALPR: https://www.openalpr.com/

• EasyOCR GitHub Repository: https://github.com/JaidedAI/EasyOCR

• PyImageSearch blog: https://www.pyimagesearch.com/

4. Tutorials and Guides:

• PyImageSearch License Plate Recognition Tutorials: https://www.pyimagesearch.com/?s=license+plate+recognition

• TensorFlow Object Detection API: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

5. Conferences and Workshops:

• International Conference on Document Analysis and Recognition (ICDAR): https://www.icdar2021.org/

• IEEE Conference on Computer Vision and Pattern Recognition (CVPR): http://cvpr2021.thecvf.com/

• European Conference on Computer Vision (ECCV): https://eccv2022.eu/

6. GitHub Repositories:

• OpenALPR GitHub Repository: https://github.com/openalpr/openalpr

• ANPR Toolkit: https://github.com/gutfeeling/anpr-toolkit

These references cover a range of topics related to number plate recognition, including research papers, books, online resources, tutorials, conferences, workshops, and GitHub repositories. They provide valuable insights, methodologies, algorithms, and implementations for building and understanding number plate recognition systems.