License Plate Recognition Using LLMs — Project Documentation

Introduction

1. What is License Plate Recognition?

License Plate Recognition (LPR), also known as Automatic Number Plate Recognition (ANPR), is a computer vision-based technology used to identify and read vehicle registration plates from images or video frames. This technology plays a key role in modern traffic systems, parking management, toll booths, and law enforcement. It involves detecting the region of the license plate and recognizing the characters using Optical Character Recognition (OCR) techniques.

2. What is OCR and CNN?

OCR, or Optical Character Recognition, is the process of converting text from images into machine-readable form. It is used to extract alphanumeric characters from detected license plates.
CNN, or Convolutional Neural Networks, are deep learning models specialized in image analysis. In this project, OCR is powered by a CNN-based architecture provided by the EasyOCR library, which combines convolutional layers with recurrent networks for accurate character recognition.

3. What are the models used in OCR?

The models used in this project include:

- EasyOCR, which uses a combination of Convolutional Recurrent Neural Networks (CRNN) and Connectionist Temporal Classification (CTC) loss to recognize sequences of characters in natural images.
- OpenCV, which is employed for image preprocessing, edge detection, and extracting the region of interest where license plates are likely located.
- Imutils, a helper library used to simplify image manipulation tasks like resizing and sorting contours.

4. Features of License Plate Recognition

This system is capable of:

- Detecting vehicle license plates from both static images and video feeds.

- Recognizing the text present in the plate using EasyOCR.
- Drawing bounding boxes around detected plates and displaying the recognized text.
- Handling varying lighting conditions, angles, and different fonts to a reasonable extent.

5. How is OCR used in License Plate Recognition?
Once a potential license plate region is detected using contour-based methods in OpenCV, it is cropped and passed to the OCR module. EasyOCR processes the cropped image using a pretrained model that detects and interprets the characters. The recognized text is then printed and displayed as an annotation on the original image.

6. Evolution of License Plate Recognition
Initially, license plate recognition was based on rule-based methods such as edge detection and template matching. These systems struggled with complex real-world conditions like noise and perspective distortion. With the advancement of deep learning and OCR models, modern systems like EasyOCR have significantly improved in accuracy, robustness, and speed, making LPR systems more viable for real-time applications.

Problem Statement

The goal of this project is to develop an automated system that can detect and recognize vehicle license plates from images and video footage in real-time. The system should use computer vision and OCR techniques to identify the plate region, extract it, and recognize the characters accurately.

Limitation

While the system is effective under good conditions, it has the following limitations:

- It may produce inaccurate results when the license plate is heavily blurred, tilted, or poorly illuminated.
- It is currently limited to recognizing English alphanumeric characters, as supported by the default EasyOCR model.
- It may not perform well when multiple vehicles or overlapping plates are present in the frame.

- False positives can occur when other rectangular shapes are mistaken for license plates.

Proposed System / Solution

The proposed system captures input either from an image or a video stream. Using OpenCV, the image is preprocessed through grayscale conversion, noise reduction, and edge detection. Then, contours are detected and filtered to identify the most likely license plate region. This region is extracted and passed to EasyOCR, which performs text recognition. The recognized text is then overlaid on the original image or video along with a bounding box, giving a real-time license plate recognition system.

Pipeline of Project

Dataflow Chart and Explanation:

The data flows through the following major stages:

1. The system begins with an input source such as an image or video file.
2. The input is preprocessed using grayscale conversion and noise filtering.
3. Edge detection techniques are applied to identify possible shapes in the image.
4. Contour detection is used to find rectangular regions, which may correspond to license plates.
5. The most probable plate region is cropped and sent to the OCR module.
6. EasyOCR processes the image and returns the recognized text.
7. The recognized license plate number is then displayed along with the plate's location in the image.

Data Collection, Evaluation, and Error Minimization:

- Data can be collected from various sources such as public datasets or video feeds.
- OCR outputs are validated by length or pattern to reduce false readings.
- Error handling is implemented to skip frames where plates are not detected confidently.

Coding:

Line-by-Line Code Explanation

1. Importing Required Libraries

import cv2
Imports OpenCV for image and video processing operations.

import numpy as np
Imports NumPy for numerical operations, such as array manipulations.

import easyocr
Imports EasyOCR for performing Optical Character Recognition (OCR) to extract text from license plates.

import imutils
Provides utility functions to simplify OpenCV tasks like resizing and contour management.

import re
Imports the Regular Expressions module for cleaning up the detected text.

from google.colab.patches import cv2_imshow
Used to display images in Google Colab notebooks (as cv2.imshow() doesn't work in Colab).

from google.colab import files
Enables uploading and downloading files within Google Colab.

import time
Used to measure and display the processing speed (frames per second).

---

2. Initialize EasyOCR Reader

reader = easyocr.Reader(['en'], gpu=True)
Creates an OCR reader object with English language support and enables GPU acceleration if available.

---

3. License Plate Detection Function

def detect_plate(image):
Defines a function to detect the license plate region in an image.

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
Converts the input image from color (BGR) to grayscale for easier processing.

filtered = cv2.bilateralFilter(gray, 11, 17, 17)
Applies a bilateral filter to reduce noise while preserving edges.

edged = cv2.Canny(filtered, 30, 200)
Performs edge detection using the Canny algorithm.

cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
Finds the contours (boundaries) of objects in the image.

cnts = imutils.grab_contours(cnts)
Extracts the correct format of contours from the returned tuple.

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:10]
Sorts the contours by area (largest first) and keeps the top 10.

location = None
Initializes a variable to store the plate contour.

for c in cnts:
Iterates over the top contours.

peri = cv2.arcLength(c, True)
Calculates the perimeter of each contour.

approx = cv2.approxPolyDP(c, 0.018 * peri, True)
Approximates the contour to a polygon with fewer vertices.

if len(approx) == 4:
Checks if the polygon has 4 sides (rectangle-like).

location = approx
Stores the rectangle as the potential license plate region.

break
Stops the loop once a suitable rectangle is found.

return location
Returns the detected location (bounding box) of the plate.

---

4. Recognize License Plate Text

def recognize_plate_text(image, location):
Defines a function to recognize the text from the detected license plate region.

if location is None:
Checks if any plate location was found.

return "No plate detected", None
Returns a message if no plate was detected.

mask = np.zeros(image.shape[0:2], dtype=np.uint8)
Creates a blank (black) mask with the same height and width as the input image.

cv2.drawContours(mask, [location], 0, 255, -1)
Draws the detected license plate area on the mask in white.

(x, y) = np.where(mask == 255)
Finds the coordinates where the mask is white (i.e., the plate region).

(topx, topy) = (np.min(x), np.min(y))
Finds the top-left corner of the license plate region.

(bottomx, bottomy) = (np.max(x), np.max(y))
Finds the bottom-right corner.

plate_region = image[topx:bottomx+1, topy:bottomy+1]
Crops the license plate from the image.

result = reader.readtext(plate_region)
Uses EasyOCR to detect text from the cropped region.

text = ""
Initializes an empty string to store the detected text.

for detection in result:
Iterates over each detected text area.

text += detection[1] + " "
Appends the recognized text to the string.

cleaned_text = re.sub(r'[^A-Za-z0-9]', '', text.strip())
Removes any non-alphanumeric characters from the text.

return cleaned_text, plate_region
Returns the cleaned text and cropped plate region.

---

5. Image Processing Function

def process_image(image):
Defines a function to process an image (detect and recognize license plate).

if image.shape[0] > 1000 or image.shape[1] > 1000:
Checks if the image is too large.

image = imutils.resize(image, width=1000)
Resizes the image for faster processing.

original = image.copy()
Keeps a copy of the original image for OCR.

location = detect_plate(image)
Calls the detection function.

if location is not None:
If a plate is detected:

cv2.drawContours(image, [location], -1, (0, 255, 0), 3)
Draws a green box around the detected plate.

plate_text, plate_region = recognize_plate_text(original, location)
Recognizes the text inside the detected plate.

cv2.putText(image, plate_text, (20, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
Displays the recognized text at the top-left of the image in red.

return image, plate_text, plate_region
Returns the processed image, detected text, and cropped plate.

else:
If no plate is found:

return image, "No plate detected", None
Returns the original image with a "not detected" message.

---

6. Real-Time Video Processing Function

def process_video(video_path=0):
Defines a function to handle video file upload and processing.

uploaded = files.upload()
Opens a file upload dialog in Google Colab.

for filename in uploaded.keys():
Processes each uploaded file.

cap = cv2.VideoCapture(filename)
Opens the video file for reading.

fps = cap.get(cv2.CAP_PROP_FPS)
Gets frames per second of the video.

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
Gets video width.

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
Gets video height.

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
Specifies the codec for writing output video.

out = cv2.VideoWriter(output_filename, fourcc, fps, (width, height))
Initializes the video writer for output.

while True:
Starts processing the video frame by frame.

ret, frame = cap.read()
Reads one frame from the video.

if frame_count % 5 == 0:
Processes every 5th frame for performance reasons.

processed_frame, plate_text, _ = process_image(process_frame)
Detects and recognizes the license plate.

out.write(processed_frame)
Writes the processed frame to the output video.

if processed_count % 10 == 0:
Prints processing stats every 10 processed frames.

cap.release()
Closes the video input stream.

out.release()
Saves the output video file.

---

7. Main Function

def main():
Defines the entry point of the script.

print("License Plate Detection and Recognition")
Prints the project title.

process_video()
Calls the video processing function.

if __name__ == "__main__":
Checks if this script is being run as the main program.

main()
Calls the main function to start the application.

Architecture Description:

The architecture of the system includes the following layers:

- Input Layer: Accepts an image or video.
- Preprocessing Layer: Applies filters and detects edges.
- Detection Layer: Identifies and extracts the license plate region.
- OCR Layer: Reads and returns the text using EasyOCR.
- Output Layer: Annotates and displays the results.

Feature Enhancement:

To enhance the system and overcome the current limitations, the following improvements can be introduced:

- Replace contour detection with YOLO or SSD object detection models for more reliable plate localization.

- Train a custom OCR model to support different languages and license plate styles.
- Add a vehicle tracking system to monitor license plates over time in video streams.
- Improve low-light performance by applying advanced image enhancement techniques.
- Deploy the model as a web service using Flask or Streamlit for easier accessibility.

These features would significantly increase the system's accuracy, versatility, and usability in real-world scenarios.

Conclusion

This project presents a working license plate recognition system built using OpenCV and EasyOCR. It effectively detects license plates and recognizes their characters from both images and videos. The system demonstrates how deep learning-based OCR can be integrated with classical image processing techniques to solve practical problems in traffic monitoring and surveillance. With planned enhancements, it has the potential to become a robust, real-time license plate reader for large-scale deployment.