INSTITUTE OF TECHNOLOGY OF CAMBODIA

Department of applied mathematics and statistics

PROJECT REPORT OF PROGRAMMING FOR DATA SCIENCE

Topic: Car License Plate Detection

Lecturers: Mr. OL Say (course)

Mr. KHEAN Vesal (TP)

No	Name of Students	ID of Students	Score
1.	LUN RATHANA	e20220368	.....
2.	NOB SREYNICH	e20220741	.....
3.	NOV PANHAVATH	e20221643	.....
4.	PHYRUN PICHCHHORDA	e20220895	.....
5.	PRIM VEASNA	e20221402	.....

Academic Year 2024-2025

Table of Content

1. INTRODUCTION

` Car plate detection and recognition are a widely used application in intelligent transportation systems, security monitoring, and vehicle tracking. The goal of this project is to develop a system that can detect vehicle license plates from images and extract the plate numbers using Optical Character Recognition (OCR). The main challenge in car plate detection lies in accurately identifying the plate region under different lighting conditions, angles, and image quality. Once the plate is detected, the characters must be correctly extracted for further usage such as database logging, vehicle monitoring, or automation systems. This project uses image processing techniques, object detection, and OCR tools to build an end-to-end pipeline. The detection phase focuses on locating the plate in the image using bounding boxes. After detection, OCR is applied to the cropped region to extract alphanumeric characters.

1.1 Objectives

- To develop a simple and efficient car plate detection system.
- To apply image preprocessing for better OCR accuracy.
- To extract readable text from plates using EasyOCR.
- To optionally deploy the system using Flask for interactive testing.

1.2 Tools and Technologies

- Python: for scripting and model implementation
- OpenCV: for image processing and preprocessing
- EasyOCR: for optical character recognition
- Flask *(optional)*: for deploying the model in a web app
- Pascal VOC XML: for handling annotated data

2. DATASET OVERVIEW

The dataset used in this project consists of vehicle images annotated with car plate bounding boxes. It is organized into three main folders: train, val, and test, each containing two subfolders – images and annotations.

- images/: Contains .png files of vehicles.
- annotations/: Contains .xml files in Pascal VOC format, defining the bounding box coordinates of the license plates.

2.1 Dataset Structure

```
dataset/
├── train/
│   ├── images/
│   └── annotations/
├── val/
│   ├── images/
│   └── annotations/
└── test/
    ├── images/
    └── annotations/
```

Figure 1 and 2: Image of cars

2.2 Annotation Format

Each .xml file contains:

- Image filename and size
- Object class (e.g., "plate")
- Bounding box (xmin, ymin, xmax, ymax)

This structure allows the model to learn where the plates are located and apply detection algorithms accordingly.

2.3 Challenges in the Dataset

- Variations in lighting and shadows
- Low-resolution or blurred plates
- Plates at different angles or partially occluded
- Multiple vehicles in a single image (some noisy samples)

To handle these challenges, preprocessing and filtering steps were applied to improve the model's performance and OCR accuracy.

3. Methodology

The car plate detection and recognition system were developed through a step-by-step pipeline that includes image preprocessing, plate detection, and text recognition using OCR. Below are the three main stages involved:

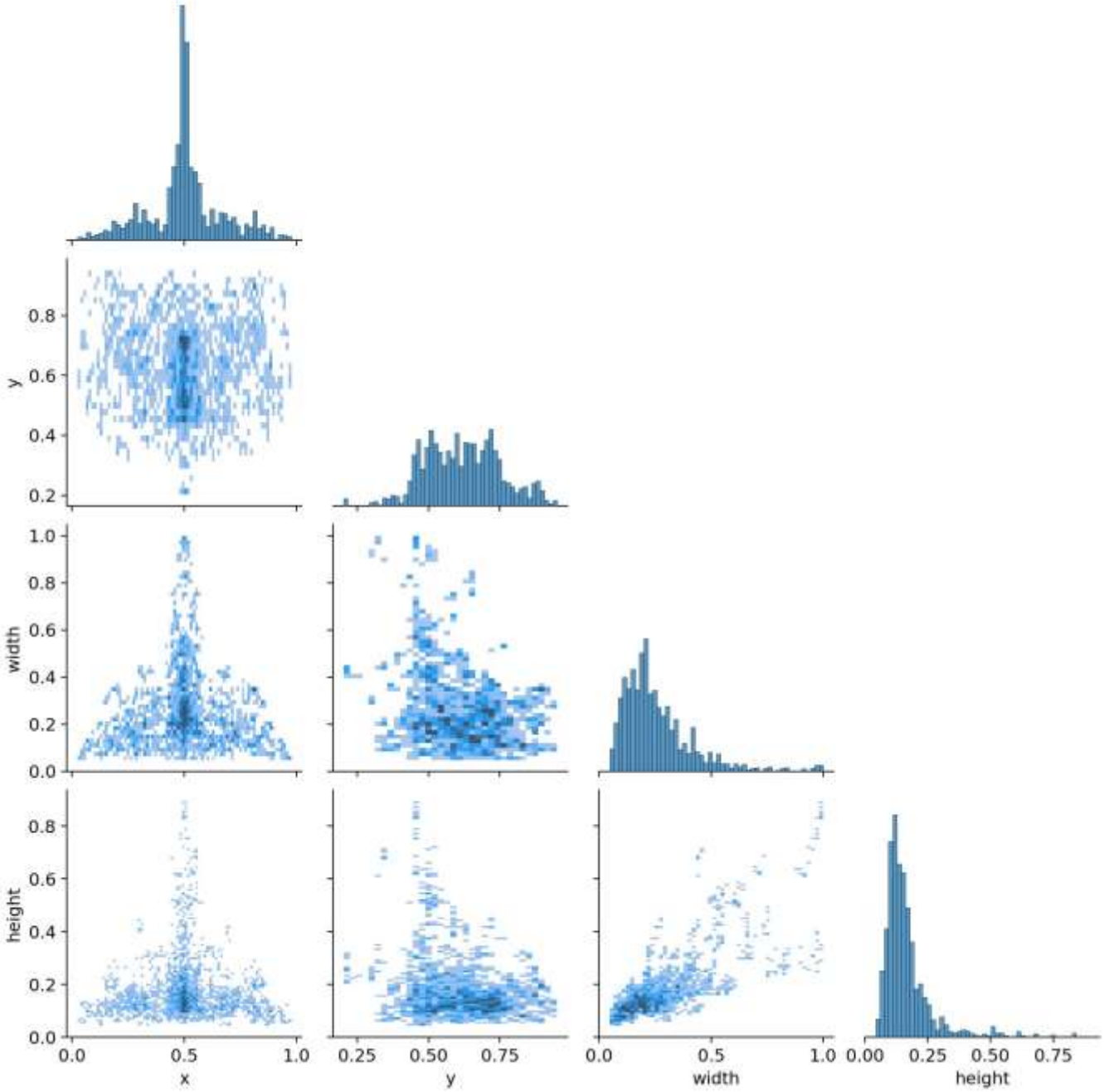
3.1 Preprocessing

Before applying detection or OCR, image preprocessing is essential to improve clarity and reduce noise. The following steps were performed:

- Cropping: The detected plate region from the image is cropped using bounding box coordinates.
- Resizing: The cropped image is resized to a standard size (e.g., 300×100) for consistency.
- Grayscale Conversion: Converted the cropped image to grayscale to reduce complexity.
- Bilateral Filtering: Applied to preserve edges while smoothing noise.
- Contrast Enhancement (CLAHE): Adaptive histogram equalization was used to improve visibility under poor lighting.

These preprocessing steps help increase the accuracy of text detection when passed to the OCR engine.

	filename	width	height	class	xmin	ymin	xmax	ymax
0	Cars0.png	500	268	licence	221	120	424	178
1	Cars1.png	400	248	licence	129	123	267	165
2	Cars10.png	400	225	licence	135	0	308	153
3	Cars100.png	400	267	licence	170	109	219	136
4	Cars101.png	400	300	licence	162	197	245	225



3.2. Plate Detection

To identify the plate region from full vehicle images, object detection techniques were applied.

- Annotation Usage: XML annotation files in Pascal VOC format were used to extract ground-truth bounding boxes for training and evaluation.
- Bounding Box Extraction: Coordinates (xmin, ymin, xmax, ymax) were used to crop out the plate regions programmatically.
- YOLO or Manual Cropping *(Depending on project)*: YOLO can be used for real-time detection, or manual cropping using annotation data for simplified testing.

This step localizes the plate area to prepare it for character recognition.

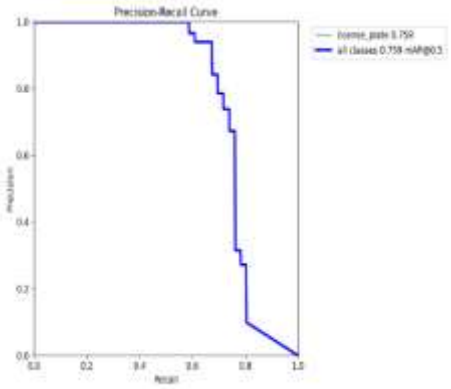
3.3 OCR and Recognition

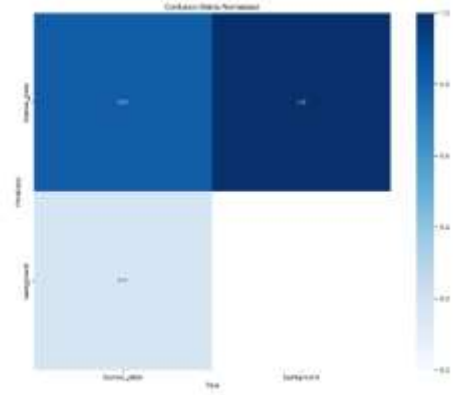
After detection, OCR (Optical Character Recognition) was used to extract the characters from the license plate.

- Tool Used: EasyOCR
- Character Filtering: Only uppercase alphabets and numbers were allowed using allowlist="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".
- Output: The OCR result returns the plate number as text.

3.4 Model Evaluation

Data training





4. RESULTS

The system was tested using various images from the test set to evaluate its ability to detect license plates and extract the correct characters.

4.1 Detection and OCR Output

After preprocessing and detection, the cropped plate region was passed to EasyOCR for recognition. Below is an example of a successful output:

Figure: Original image

Figure: Cropped Plate Region

Detected Plate Text: **MH12NE8922**

The model was able to correctly identify and read the plate text in most test images with good lighting and clear visibility.

4.2 Sample Results

Figure: Original image

Figure: Plate Crop 1
Detected Text: **DZI7YXR**

-
-

- Image 2 (Original Image)

Figure : Plate Crop 2
Detected Text: **DL7CN5617**

- Image 3

Figure: Plate Crop 3
Detected Text: **9214**

4.3 Error Cases:

Expected: **P018299**

OCR Output: **4012927**

In some cases, the OCR results were incorrect due to

- Low contrast or dark images
- Blurry or tilted plates
- Complex backgrounds or small plate size

5. DEPLOYMENT

To make the car plate detection and recognition system more accessible and user-friendly, the model was deployed using **Flask**, a lightweight Python web framework. This allows users to upload images through a simple web interface and view the results directly in their browser.

5.1 Tools and Technologies

- Flask – Used to create the web server and routing logic.
- OpenCV – Handles image preprocessing.
- EasyOCR – Extracts text from detected plates.
- HTML/CSS – Builds the front-end user interface.
- Werkzeug/File Handling – Manages file uploads.

5.2 Application Workflow

1. The user opens the web app in a browser.
2. The user uploads an image of a vehicle.
3. The image is processed to detect the car plate using a trained model.
4. The detected plate region is cropped and passed to EasyOCR.
5. The recognized plate number is displayed on the web page along with the detection box.

5.3 Folder Structure

Below is the project folder structure used for deploying the car plate detection system:

```
Car-Plate-Detection/
├── app.py                # Main Flask application
├── static/               # Folder for uploaded images or static assets
├── templates/            # HTML files for front-end interface
├── runs/                 # YOLOv8 model training results
├── requirements.txt      # Python dependencies
└── render.yaml           # Configuration for deployment (e.g., Render.com or Docker)
```

- app.py: Contains the core logic for image upload, detection, and OCR.
- static/: Stores uploaded images and any output (like detected results).
- templates/: Contains the front-end HTML, typically index.html.
- runs/: Stores output folders from training (e.g., weights, metrics).
- render.yaml: Configuration file for deployment on platforms like Render or similar.
- requirements.txt: Lists all required Python packages to install.

5.4 Running the Application

To run the Flask app locally:

pip install -r requirements.txt

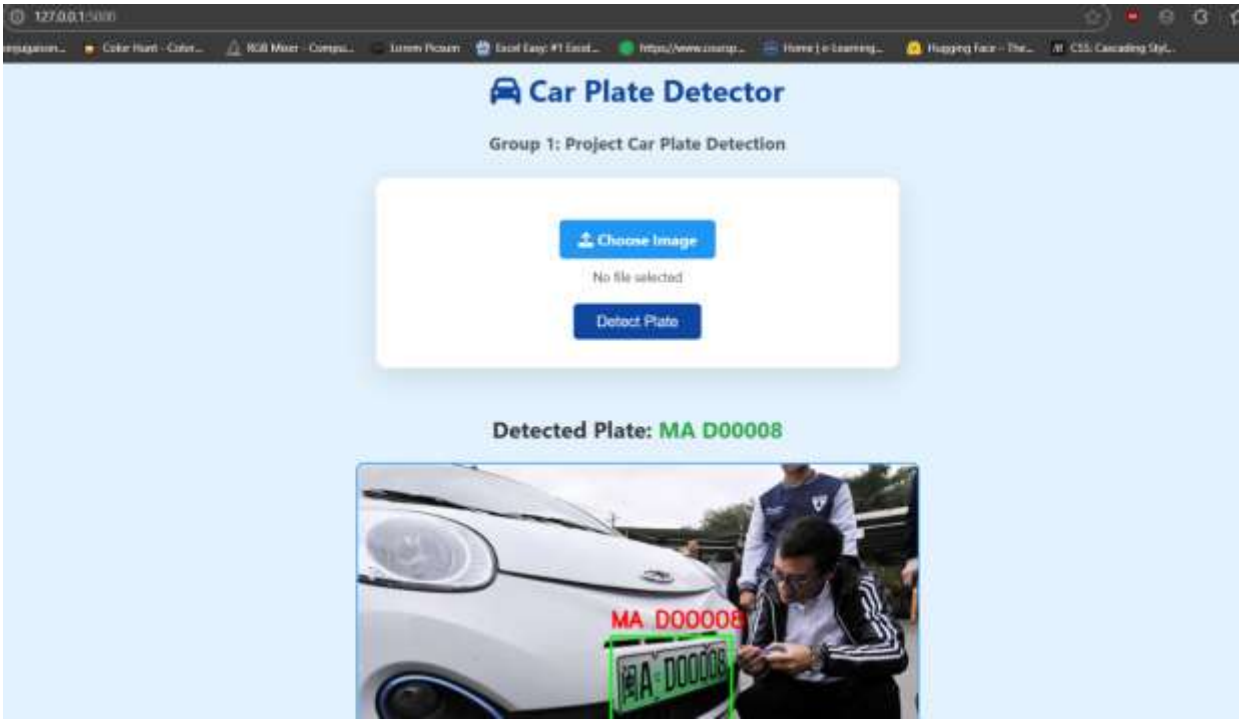
python app.py

Once started, the app runs at http://127.0.0.1:5000/, where users can interact with it in a web browser.

5.5 User Interface

The web app includes:

- An image upload section
- A result display area showing the detection box and OCR result



6. DISCUSSION

The project successfully demonstrated a complete pipeline for car plate detection and recognition using Python, OpenCV, YOLO, and EasyOCR. The following observations highlight the strengths and limitations encountered during implementation and testing.

6.1 What Worked Well

- The YOLOv8 detection model accurately localized license plates in most test images, even under different angles and lighting.
- Image preprocessing steps (like CLAHE and bilateral filtering) significantly improved OCR accuracy, especially on low-contrast images.
- EasyOCR performed well for reading clear alphanumeric characters without the need for heavy OCR model training.
- The Flask web app made the system interactive and user-friendly, allowing anyone to test the model by uploading an image.

6.2 Limitations

Despite the overall success, the system has some limitations:

- OCR Accuracy Drops with blurry, small, or tilted plates, especially if the characters are partially visible or occluded.
- No Multiple Plate Detection: The current version assumes one plate per image and does not handle multiple cars.
- Model Sensitivity: YOLOv8's performance depends on proper annotation quality and balanced training samples.

6.3 Opportunities for Improvement

Future versions of this project can include:

- Data Augmentation: Improve model robustness by training on a more diverse dataset with augmented samples.
- Post-processing OCR: Add logic to fix common OCR mistakes (e.g., misreading Z as 2).
- Multiplate Support: Enhance the app to detect and extract multiple plates from a single image.
- Deployment Upgrade: Consider converting the Flask app into a Docker container for easier deployment across environments.

7. CONCLUSION

This project successfully implemented a car plate detection and recognition system using Python, YOLOv8, OpenCV, and EasyOCR. The system was able to detect license plates from vehicle images and extract readable alphanumeric text with high accuracy under favorable conditions. The process included:

- Preprocessing input images to enhance OCR performance
- Detecting plate regions using a YOLO model
- Applying EasyOCR to read the plate text
- Optionally deploying the system using Flask for user interaction

The results showed strong performance in clean images with clear plates, while also revealing challenges in low-quality or complex scenes. Overall, the project highlights the potential of combining modern object detection and OCR techniques to build real-world computer vision applications. With further improvements, such as error correction and model tuning, this system can be developed into a more reliable tool for use in traffic monitoring, parking systems, and smart security solutions.

8. FUTURE WORK

To improve the system and make it more reliable in real-world applications, several enhancements can be considered in future development:

- Improve OCR Accuracy: Implement post-processing techniques to automatically correct common OCR mistakes (e.g., confusing Z with 2 or O with 0).
- Train Custom OCR Model: Fine-tune a character recognition model specifically for license plates to outperform general OCR tools.
- Handle Multiple Plates: Extend the system to detect and process multiple car plates in a single image.
- Video Stream Support: Add real-time detection from video or webcam input for live surveillance applications.
- User History and Logs: Store user uploads and detected results in a database for history tracking and usage statistics.
- Deployment Upgrade: Package the app in Docker or deploy it to cloud services like Render, AWS, or Heroku for wider accessibility.

REFERENCES

Docs + Quickstarts | Render. (n.d.). https://render.com/docs

JaidedAI. (n.d.). *GitHub - JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic and etc.* GitHub. https://github.com/JaidedAI/EasyOCR

OpenCV documentation index. (n.d.). https://docs.opencv.org/

Pjreddie. (n.d.). *darknet/scripts/voc_label.py at master · pjreddie/darknet*. GitHub. https://github.com/pjreddie/darknet/blob/master/scripts/voc_label.py

Ultralytics. (2025, April 14). *Home*. Ultralytics YOLO Docs. https://docs.ultralytics.com/

Welcome to Flask — Flask Documentation (3.1.x). (n.d.). https://flask.palletsprojects.com/