LAB: Parking Management System

Date: 2024-June-07

Author: Gyeonheal An (21900416)

Github: https://github.com/AnGyeonheal/DLIP_GH

Demo Video: Video

I. Introduction

1. Objective

We are required to create a simple program that (1) counts the number of vehicles in the parking lot and (2) display the number of available parking space.

For the given dataset, the maximum available parking space is 13. If the current number of vehicles is more than 13, then, the available space should display as '0'.

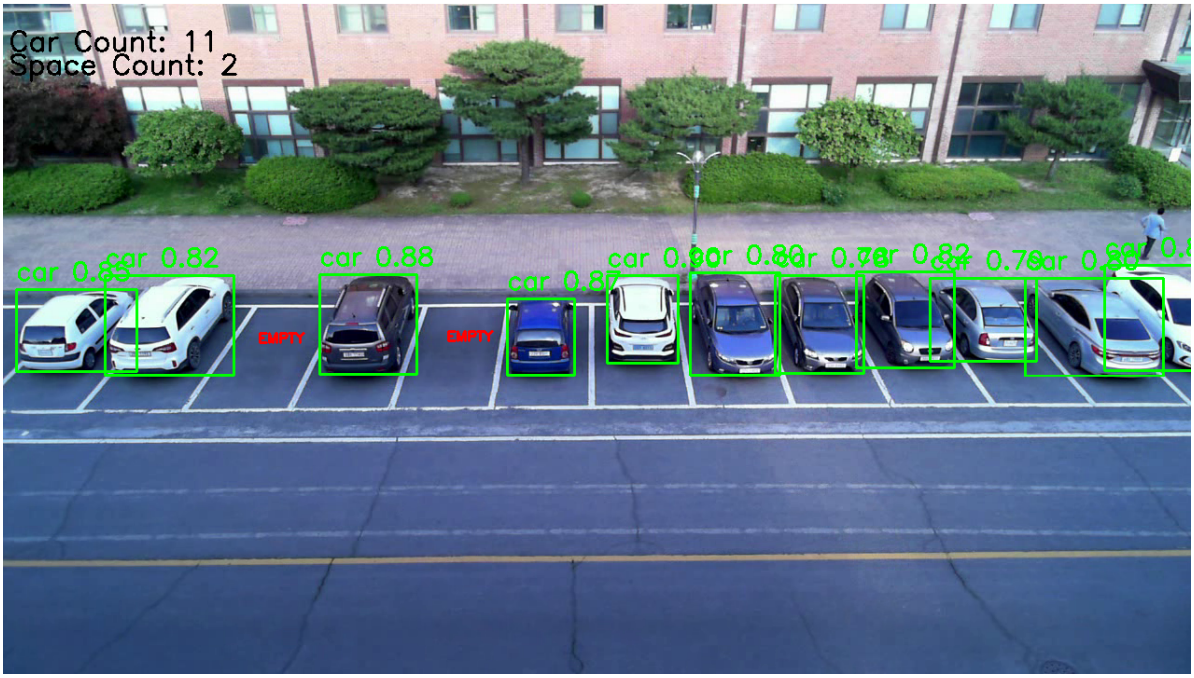

Fig 1. LAB: Parking Management System (YOLOv8)

2. Problem Conditions

- Count the number of vehicles in the parking lot for each frame
  - Exclude vehicles outside the designated parking spaces
  - Consider a vehicle to be outside the parking area if the center of the car is outside the parking space
- Ensure that the same vehicle is not counted more than once
- Display the current number of vehicles and available parking spaces accurately

- Save the vehicle counting results in a file named '**counting_result.txt**'.
    - When the program is executed, the 'counting_result.txt' file should be created automatically for the given input video.
    - Each line in the 'counting_result.txt' file should contain the pair of **frame number and number of detected cars**.
    - Frame numbers should start from 0.

3. Preparation

Software Installation

- Pycharm JetBrain

Dataset

- Download the test video file: click here to download

II. Algorithm

1. Overview

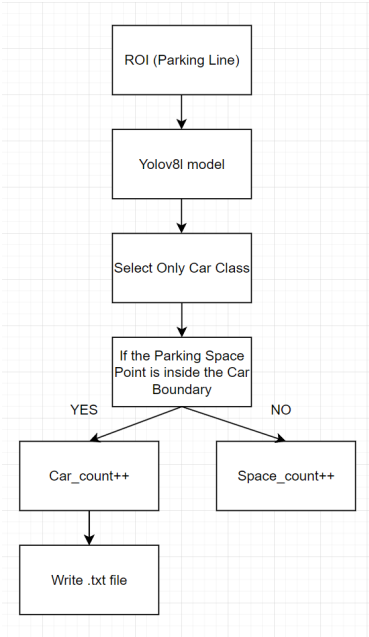

Fig 2. Algorithm Flowchart

The objective of our analysis is to determine the available parking spaces in the parking lot. We defined the region of interest (ROI) based on the upper and lower lines of the parking lot to determine the area where the YOLO model will be applied.

We used the YOLOv8l model for detection. Since the video contains not only cars but also people, we configured the model to recognize only the class with `class_id` 2 (Car). The central points of each parking spot in the video were previously stored as points. If a point lies outside the boundary box (defined by x1, y1, x2, y2) of the detected car, it indicates that the parking spot is available, and the `space_count` is increased. Conversely, if the point lies inside the boundary box, the `car_count` is increased.

Subsequently, the number of cars in each frame is saved in a txt file for comparison. This data is used to evaluate the performance of the algorithm.

2. Procedure

#1. ROI Setting

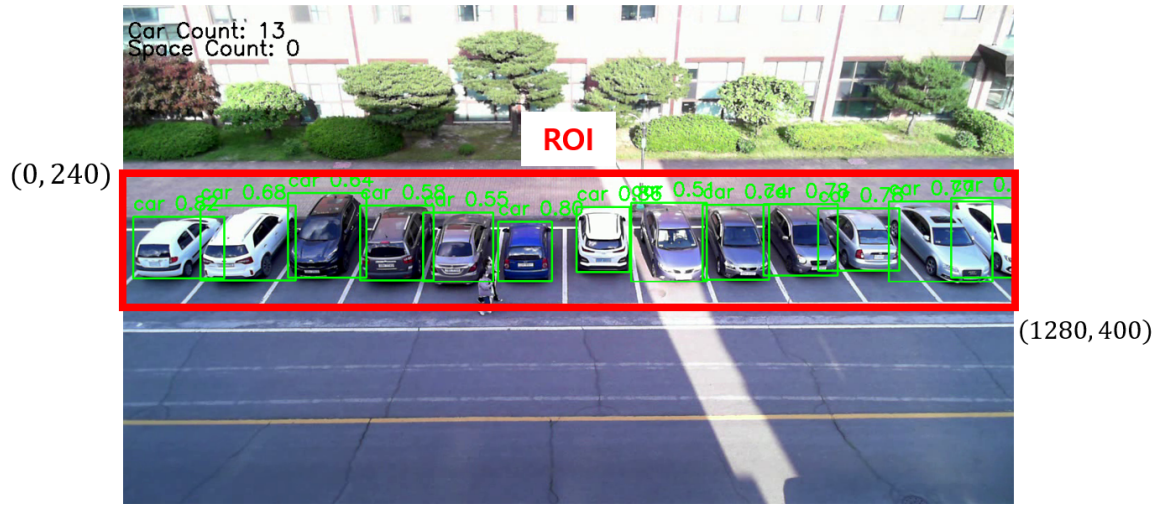

Fig 3. ROI (Region of Interest)

To avoid recognizing vehicles outside the parking lot, we set a Region of Interest (ROI).

```
x, y, w, h = 0, 240, 1280, 160
roi = frame[y:y+h, x:x+w]
results = model(roi)
```

#2. Select Car Class

The video contains not only cars but also people, so to exclude people from being classified, we considered only the car class with the following exception handling.

```
if class_id == 2:
    label = f'{model.names[class_id]} {confidence:.2f}'
    cv.rectangle(roi, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv.putText(roi, label, (x1, y1 - 10), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255,
0), 2)
```

#3. Counting Cars and Spaces

When the central point of a parking spot is outside the boundary box of a car, it is marked as 'Empty' and the `space_count` is increased.

```
space_points = [(83, 366), (202, 367), (297, 363), (404, 364), (499, 361), (588,
358), (684, 357), (785, 357), (877, 357), (971, 355), (1063, 359), (1160, 360),
(1249, 354)]


for point in space_points:
px, py = point
if x1 < px < x2 and y1 < py < y2 + 240:
car_count += 1
occupied_points.add(point)
break

for point in space_points:
```

```
if point not in occupied_points:
px, py = point
space_count += 1
```

III. Result and Discussion

1. Final Result

Fig 4. Result Image.

Demo Video Embedded: Video

2. Discussion

```
Number of frames with different car counts in the first 1500 frames: 12

Accuracy is: 99.20
```

Fig 5. Accuracy Image.

Evaluation	Score
Accuracy	99.2%
Precision	99.73%
Recall	99.47%
F1-Score	99.6%

Table 1. Accuracy of Algorithm

It was found that the results of 12 frames out of 1500 given frames differed.

Conclusion

In this project, we developed a system to accurately count the number of vehicles in a parking lot using the YOLOv8l object detection model. By defining a Region of Interest (ROI), we ensured that only vehicles within the parking lot boundaries were counted, effectively ignoring those outside the designated areas. This was achieved by setting specific criteria where the central point of a parking spot must lie within the boundary box of a detected vehicle for it to be considered parked.

We also implemented exception handling to consider only the car class, thereby excluding other objects such as people from our detection results. This approach was crucial in maintaining the accuracy of the vehicle count and the subsequent determination of available parking spaces.

Through rigorous testing, it was found that our algorithm's results deviated in only 12 frames out of the 1500 frames provided, indicating a high level of accuracy. Specifically, using the YOLOv8l model, which is known for its high computational complexity and precise inference capabilities, we achieved a detection accuracy of 99.2%.

While the YOLOv8l model demonstrated excellent accuracy, it also required significant memory usage and had slower inference speeds compared to the smaller m, s, and n models. To address this, we propose fine-tuning pretrained m, s, and n models to balance the need for speed and accuracy, potentially enhancing overall performance.

The vehicle counting results were automatically saved in a text file named 'counting_result.txt', where each line contained the frame number and the corresponding number of detected cars. This ensures that the system not only performs real-time detection but also maintains a record of vehicle counts for further analysis.

In conclusion, our system proved to be highly effective in counting vehicles and determining available parking spaces in real-time, with minimal errors. By leveraging advanced object detection models and fine-tuning techniques, we can further optimize this system to meet various operational requirements, making it a robust solution for modern parking management systems.

Appendix

```
# * DLIP_LAB4_CNN Object Detection 1_Parking Management System
# * author: Gyeonheal An
# * Date: 2024-06-07

import logging
from ultralytics import YOLO
import cv2 as cv

def main():
    setup_logging()
    video_path = 'DLIP_parking_test_video.avi'
    cap = load_video(video_path)
    model = initialize_model('yolov8l.pt')
    space_points = [(83, 366), (202, 367), (297, 363), (404, 364), (499, 361),
(588, 358), (684, 357),
                    (785, 357), (877, 357), (971, 355), (1063, 359), (1160,
360), (1249, 354)]  # Parking Space의 중앙점
    roi_coords = (0, 240, 1280, 160)  # ROI points
    frame_index = 0

    with open('counting_result.txt', 'w') as f:
        while cap.isOpened():
```

```
            ret, frame = cap.read()
            if not ret:
                print("Cannot find video file")
                break

            car_count, occupied_points, roi = detect_and_count_cars(frame,
model, space_points, roi_coords)
            space_count = label_empty_spaces(frame, space_points,
occupied_points)
            display_counts(frame, car_count, space_count)

            # Record the number of cars per frame in a text file.
            f.write(f'{frame_index},{car_count}\n')
            frame_index += 1
            x, y, w, h = roi_coords
            frame[y:y + h, x:x + w] = roi
            cv.imshow('YOLOv8 Detection', frame)
            if cv.waitKey(1) & 0xFF == ord('q'):
                break
    cap.release()
    cv.destroyAllWindows()

def setup_logging():
    logging.getLogger('ultralytics').setLevel(logging.CRITICAL)  # Do not output
logs to the console.

def load_video(video_path):
    return cv.VideoCapture(video_path)

def initialize_model(model_path='yolov8l.pt'):
    return YOLO(model_path)

def detect_and_count_cars(frame, model, space_points, roi_coords):
    x, y, w, h = roi_coords
    roi = frame[y:y + h, x:x + w]  # ROI
    results = model(roi)  # YOLOv8 model
    car_count = 0
    occupied_points = set()

    for result in results:
        for box in result.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            confidence = box.conf[0]  # Initializing
            class_id = int(box.cls[0])

            if class_id == 2:  # Consider only car class
                label = f'{model.names[class_id]} {confidence:.2f}'
                cv.rectangle(roi, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv.putText(roi, label, (x1, y1 - 10), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 2)

                for point in space_points:  # Check if the center point of
Parking Space is inside the car Boundary Box
                    px, py = point
                    if x1 < px < x2 and y1 < py < y2 + 240:
                        car_count += 1
                        occupied_points.add(point)  # Save the parking space
center points which is already occupied
```

```
                        break

    return car_count, occupied_points, roi

def label_empty_spaces(frame, space_points, occupied_points):
    space_count = 0
    for point in space_points:
        if point not in occupied_points:  # If the parking space is empty
            px, py = point
            space_count += 1
            cv.putText(frame, "EMPTY", (px - 20, py), cv.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 255), 2)
    return space_count

def display_counts(frame, car_count, space_count):
    count_label = f'Car Count: {car_count}'
    space_label = f'Space Count: {space_count}'

    cv.putText(frame, count_label, (10, 50), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
0), 2)
    cv.putText(frame, space_label, (10, 75), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
0), 2)

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