LAB: CNN Object Detection

Parking Management System

Date: 2024.06.06

Author: Jin Kwak/ 21900031

Github

Demo Video

Introduction

1. Objective

Vehicle counting using a CNN based object detection model.

In this lab, the author is required to create a simple program that

(1) counts the number of vehicles in the parking lot

(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'.

Problem Conditions

- Use Python OpenCV (*.py)
- Use pretrained YOLO v8.

2. Preparation

Software Installation

- Python 3.9
- opencv-python 4.9.0
- PyTorch 2.2.0
- PyCharm
- Miniconda (Virtual Environment)

3. Procedure

- Download the test video file: click here to download
- Need to count the number of vehicles in the parking lot for each frame
  - **DO NOT COUNT** the vehicles outside the parking spaces
  - Consider the vehicle is outside the parking area if the car's center is outside the parking space

- Make sure to not count duplicates of the same vehicle
- It should accurately display the current number of vehicle and available parking spaces
- Save the vehicle counting results in '**counting_result.txt'** file.
    - When program is executed, the 'counting_result.txt' file should be created automatically for a given input video.
    - Each line in text file('counting_result.txt') should be the pair of **frame# and number of detected car**.
    - Frame number should start from 0. ex) 0, 12 1, 12 ...
- In the report, evaluate the model performance with numbers (accuracy etc)
    - Answer File for Frame 0 to Frame 1500 are provided: download file
- Program will be scored depending on the accuracy of the vehicle numbers
    - TA will check the Frame 0 to the last frame

Algorithm

1. Overview

Flowchart of the program is described in Figure 1. The preprocessing part in the left top box is subdivided into ROI selection, dividing the parking spaces, and get Pre-trained YOLO model (`yolo8s.pt`)

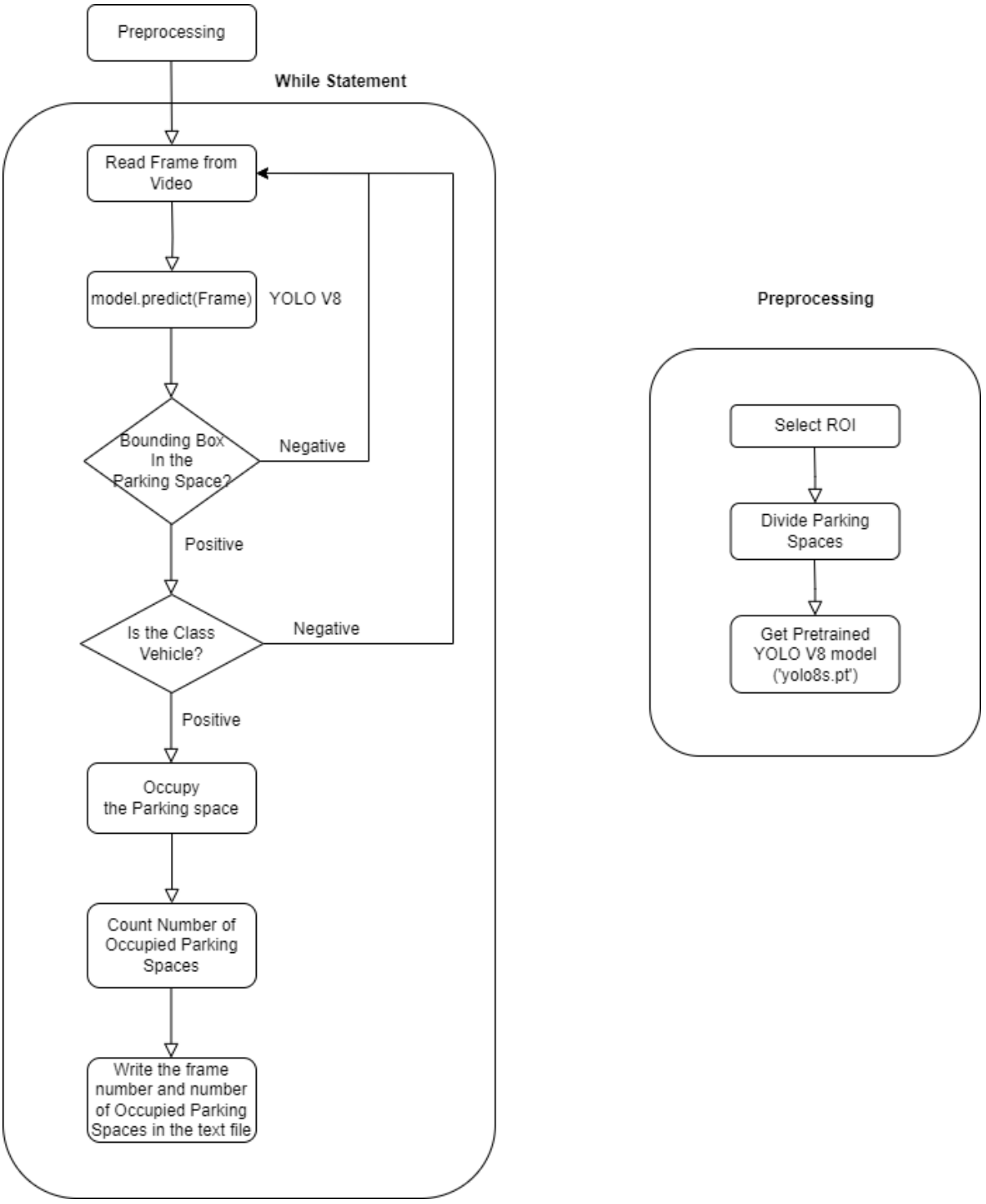

Figure 1. Flowchart of the Program

2. Procedure

Preprocessing

Region of Interest(ROI)

To only count the number of vehicles in the parking lot, the ROI is selected. Position on x-axis does not matter so the y-axis was considered.

Position is selected

Position	Coordinate [pixel]
Top-ROI	323

Position	Coordinate [pixel]
Bottom-ROI	432

Divide Parking Spaces

The Parking Spaces are divided for the parking space occupancy check. This is defined at the start of the program with 2-Dimensional list.

```
LowerY = 432
UpperY = 323

parking_spaces = [
    [(56, UpperY), (140, UpperY), (85, LowerY), (0, LowerY)],  # Parking #1
    [(140, UpperY), (255, UpperY), (196, LowerY), (85, LowerY)],  # Parking #2
    [(255, UpperY), (350, UpperY), (308, LowerY), (196, LowerY)],  # Parking #3
    [(350, UpperY), (452, UpperY), (422, LowerY), (308, LowerY)],  # Parking #4
    [(452, UpperY), (540, UpperY), (530, LowerY), (422, LowerY)],  # Parking #5
    [(540, UpperY), (634, UpperY), (635, LowerY), (530, LowerY)],  # Parking #6
    [(634, UpperY), (727, UpperY), (742, LowerY), (635, LowerY)],  # Parking #7
    [(727, UpperY), (818, UpperY), (848, LowerY), (742, LowerY)],  # Parking #8
    [(818, UpperY), (912, UpperY), (954, LowerY), (848, LowerY)],  # Parking #9
    [(912, UpperY), (1000, UpperY), (1050, LowerY), (954, LowerY)],  # Parking
#10
    [(1000, UpperY), (1090, UpperY), (1156, LowerY), (1050, LowerY)],  # Parking
#11
    [(1090, UpperY), (1200, UpperY), (1264, LowerY), (1156, LowerY)],  # Parking
#12
    [(1200, UpperY), (1280, UpperY), (1280, LowerY), (1264, LowerY)]  # Parking
#13
]
```

These coordinates represent each edge of parking spaces. As the image is taken in the view of CCTV not a bird-eye view, the parking spaces are trapezoid-shaped which is visualized in Figure 2. As there were always occupied spaces in the video, the exact parking space edges are not selected. However, the bounding box is always a rectangle without any rotation. To maximize the overlapped area, the parking spaces are carefully selected by hands to enlarge the parking space height. Furthermore, `LowerY`, `UpperY` is the average y- coordinates of the parking spaces in the y direction of pixel frame. Constant `LowerY` is later used to consider the existence of a car in the parking lot(compare with car's center)

Figure 2. Divided Parking Spaces

Python Packages

Ultralytics

`pip install ultralytics`

The `ultralytics` version used in the program is **8.2.18**.

Shapely

To download the package, on the terminal, `pip install shapely`

In the source code, `from shapely.geometry import Polygon`

This package is used to calculate the overlapped area of the bounding box and pre-defined parking spaces. The maximum overlapped area is used to select which parking space the car has parked.

Application

Class selection

To select the correct classes in the parking lot, classes of detected objects allowed are car, bus and truck.

Figure 3. YOLO model Object detection in ROI

Occupancy Check

1. If the area overlapped by parking space and the vehicle's bounding box is above a threshold, the parking space is checked as occupied.

2. The model sometimes detect a sedan as a truck or a SUV as a bus. Furthermore, sometimes it detects a vehicle twice as a bus and a car. To prevent overlapping, the preoccupied area is never be occupied again.

```
if parking_location in parking_idx or parking_location == 0:
    continue
parking_idx.append(parking_location)
occupied_space += 1
```

   Then occupied space's bounding box is manually drawn using openCV function `cv2.rectangle()`

3. Free spaces is calculated using the total number of parking lot subtracted by occupied parking lot.

Result and Discussion

1. Final Result

The number of occupied space and free space is detected by using CNN model.

Using ROI and area calculation, the bounding box is only drawn in the ROI area.

At the top of each frame, the parked location and free parking spaces are displayed.

[Demo Video Embedded]

2. Discussion

Occupancy Check

1. Occupied parking space can be checked using different methods. To conservatively detect the area, I calculated the area overlapped by a bounding box and predefined parking lot. If the area calculated is over a threshold, the program detects the vehicle is occupying a parking space. Else, the bottommost coordinate can be checked with the entrance of the parking space.

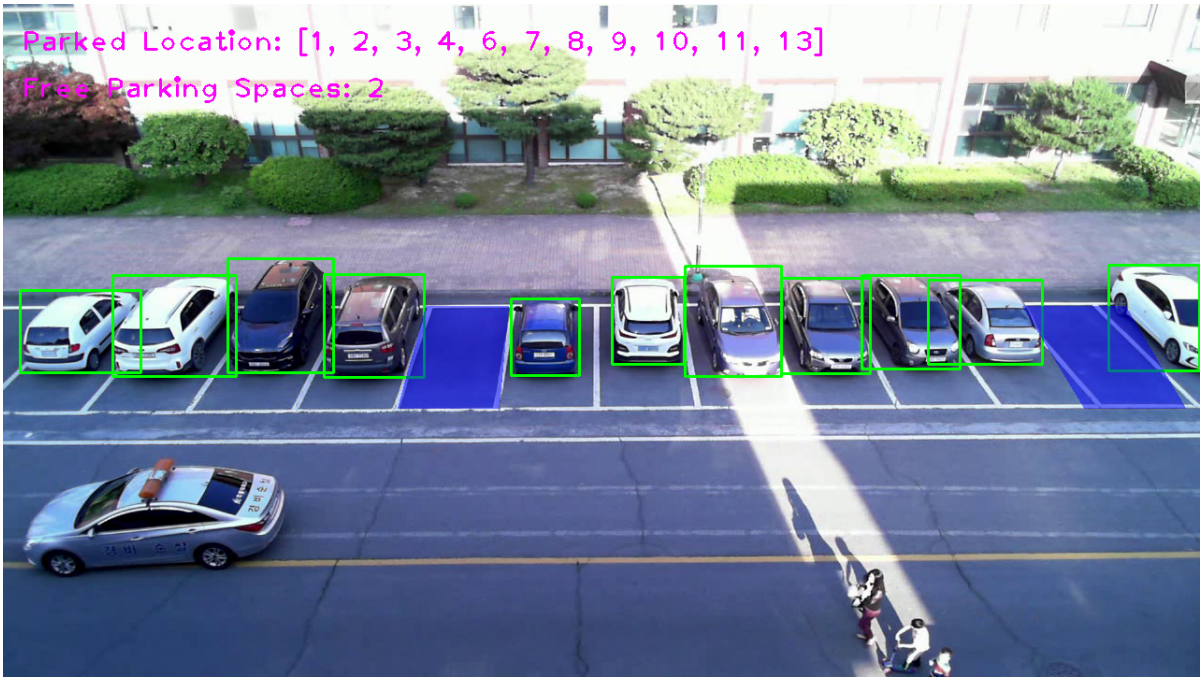

Figure 4. Bounding box at the vehicles occupying parking lot

Conclusion

Lab Purpose Achievement

The first 1501 frames are compared with the true data. Confusion Matrix is drawn in Figure 5.

As the maximum number of miss detected objects for each frame is 1, the recall, precision, F1 score can be calculated with simple calculation.

1. The accuracy of the model compared with 1501 frames is 99.07%
2. Recall of the program is 99.4%
3. Precision of the program is 99.7%
4. F1 Score of the program is 0.9953
5. Bounding boxes are only drawn on the vehicles in the parking space, therefore no true negatives.

		Predicted	
		Negative	Positive
Actual	Negative	0	5
	Positive	9	1487

Figure 5. Confusion Matrix

Therefore, the program achieves the purpose of the lab introduced in the **Procedure**

Improvement

1. To improve the performance, larger model can be used to improve accuracy. However, to save energy and real-time margin, another light model can be ran together to cross check.
2. If the camera can be set on top of the parking space, the performance can be improved as the projection of parking spaces will be relatively parallel in the image.

3. To further increase efficiency, apply Jetson Nano Board(NVIDIA Jetson TX2) to save power, money and compact management.

Appendix

`Source Code`

```
"""
================================================================================
========================================
                                        DLIP LAB 4 Python CNN Object Detection
                                            Parking Management System
                                            Handong Global University
================================================================================
========================================
@Author  : Jin Kwak
@Created : 2024.05.24
@Modified: 2024.06.06
@Version : Final
"""
import cv2 as cv
from ultralytics import YOLO
import numpy as np
from shapely.geometry import Polygon

ENUM_CAR = 2  # Car
ENUM_MOT = 3  # MotorCycle
ENUM_BUS = 5  # Bus
ENUM_TRU = 7  # Truck
lower_y = 432
upper_y = 323
parking_spaces = (((56  , upper_y), (140 , upper_y), (85  , lower_y), (0   ,
lower_y)),  # Parking #1
                  ((140 , upper_y), (255 , upper_y), (196 , lower_y), (85  ,
lower_y)),  # Parking #2
                  ((255 , upper_y), (350 , upper_y), (308 , lower_y), (196 ,
lower_y)),  # Parking #3
                  ((350 , upper_y), (452 , upper_y), (422 , lower_y), (308 ,
lower_y)),  # Parking #4
                  ((452 , upper_y), (540 , upper_y), (530 , lower_y), (422 ,
lower_y)),  # Parking #5
                  ((540 , upper_y), (634 , upper_y), (635 , lower_y), (530 ,
lower_y)),  # Parking #6
                  ((634 , upper_y), (727 , upper_y), (742 , lower_y), (635 ,
lower_y)),  # Parking #7
                  ((727 , upper_y), (818 , upper_y), (848 , lower_y), (742 ,
lower_y)),  # Parking #8
                  ((818 , upper_y), (912 , upper_y), (954 , lower_y), (848 ,
lower_y)),  # Parking #9
                  ((912 , upper_y), (1000, upper_y), (1050, lower_y), (954 ,
lower_y)),  # Parking #10
                  ((1000, upper_y), (1090, upper_y), (1156, lower_y), (1050,
lower_y)),  # Parking #11
                  ((1090, upper_y), (1200, upper_y), (1264, lower_y), (1156,
lower_y)),  # Parking #12
```

```
                  ((1200, upper_y), (1280, upper_y), (1280, lower_y), (1264,
lower_y)))  # Parking #13

def main() -> None:
    cap = cv.VideoCapture('DLIP_parking_test_video.avi')
    cv.namedWindow('DLIP_parking_test_video', cv.WINDOW_AUTOSIZE)
    model = YOLO('yolov8s.pt')
    isStop = False
    frame_number = 0
    fourcc = cv.VideoWriter_fourcc(*'MJPG')
    out = cv.VideoWriter('output.avi', fourcc, 20.0, (1280, 720))
    f = open("counting_result.txt", "w")
    while isStop == False:
        ret, frame = cap.read()
        if not ret or (cv.waitKey(1) & 0xFF == (' ')):
            isStop = True
            break
        result = model.predict(source=frame, save=False, save_txt=False)
        bbox_coords_cpu = result[0].boxes.xyxy.to('cpu').numpy()
        class_cpu = result[0].boxes.cls.detach().cpu().numpy()
        occupied_space = 0
        parking_idx = []
        for bbox_idx in range(len(bbox_coords_cpu)):
            bbox = bbox_coords_cpu[bbox_idx]
            if class_cpu[bbox_idx] == ENUM_CAR or class_cpu[bbox_idx] == ENUM_BUS
or class_cpu[bbox_idx] == ENUM_TRU and bbox[0]+bbox[2] < lower_y:
                bounding_box_coordinates = ((bbox[0], bbox[1]), (bbox[0],
bbox[3]), (bbox[2], bbox[3]), (bbox[2], bbox[1]), (bbox[0], bbox[1]))
                parking_location = calculate_intersection_area(parking_spaces,
bounding_box_coordinates)
                if parking_location in parking_idx or parking_location == 0:
                    continue
                parking_idx.append(parking_location)
                occupied_space += 1
                cv.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]),
int(bbox[3])), (0  , 255, 0  ), 2)
        overlay = frame.copy()
        parking_idx.sort()
        for parking in range(1, 14):
            if parking in parking_idx:
                continue
            cv.fillConvexPoly(frame, np.array(list(parking_spaces[parking - 1])),
(255, 0  , 0  ))
        frame = cv.addWeighted(overlay, 0.5, frame, 0.5, 0)
        cv.putText(frame, "Parked Location: " + str(parking_idx), (20, 50),
cv.FONT_HERSHEY_PLAIN, 2, (255, 0  , 255), 2)
        cv.putText(frame, 'Free Parking Spaces: ' + str(13 - occupied_space),
(20, 100), cv.FONT_HERSHEY_PLAIN, 2, (255, 0  , 255),2)
        cv.imshow('DLIP_parking_test_video', frame)
        out.write(frame)
        f.write("%d, %d \n" % (frame_number, occupied_space))
        frame_number += 1
    cv.destroyAllWindows()
    cap.release()
    out.release()
    f.close()
```

```
def calculate_intersection_area(parking_coordinates: tuple, bbox_coordinates:
tuple) -> int:
    rectangle = Polygon(bbox_coordinates)
    idx = 1
    num = 0
    for coordinates in parking_coordinates:
        trapezoid = Polygon(coordinates)
        intersection = trapezoid.intersection(rectangle)
        if intersection.area > 4000:
            num = idx
        idx += 1
    return num

if __name__ == '__main__':
    print(__doc__)
    main()
```