LAB: CNN Object Detection 1

Date: 2024-June-06

Author: Taegeon Han 21900793

Demo Video: LAB4 Demo Video

Github(Image): https://github.com/hhangun/Picture-for-LAB/issues/13

Parking Management System

Vehicle counting using a CNN based object detection model

I. Introduction

In this lab, 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'.

Figure 1. example Image

1. Software Installation

- OpenCV 4.9.0 with Visual Studio 2022
- Visual Studio Code 1.89.1
- Cuda 11.8
- Python 3.9.18
- Pytorch 2.1.2
- Torchvision 0.16.2

- YOLO v8

2. Dataset

- Video of Handong University's parking lot
- Video Link: click here to download

3. Guidelines

The whole code should be programmed using OpenCV-Python and Pytorch.

- DO NOT copy a project from online sites.
- I can refer to any online material and github repository for assistance and getting ideas with proper reference citation.
- Use pretrained YOLO v8.
    - I can also use any pretrained object detection model, such as YOLO v5
    - I can also train the model using custom/other open datasets
- I can clone a github repository of the object detection model(e.g. YOLOv8), as long as I cite it in the reference.

II. Problem

1. Procedure

- Download dataset
- 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 do 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, I must evaluate the model performance with numbers (accuracy etc)
    - Answer File for Frame 0 to Frame 1500 are provided: download file
- Your program will be scored depending on the accuracy of the vehicle numbers
    - TA will check the Frame 0 to the last frame

2. Algorithm

2.1 Overview

This is a flowchart of this lab. It measures the number of cars and available park space.

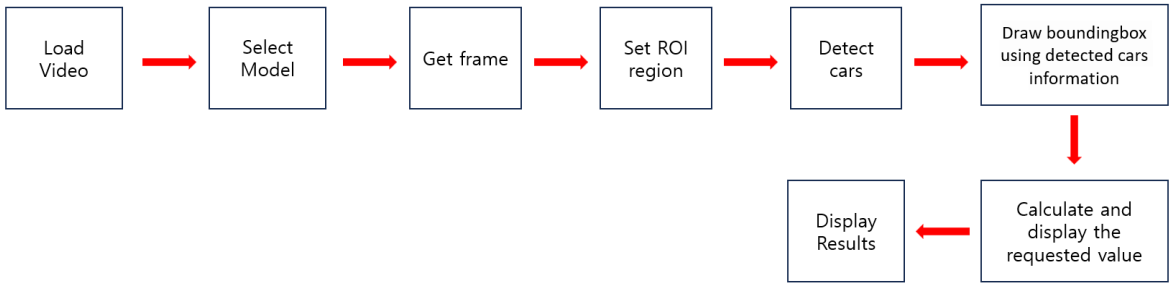

Figure 2. Outline to results

2.2 Process

2.2.1. Load Video & select model

I downloaded the video from the database above. After downloading, you need to find an appropriate pre-trained model for object detection. YOLOv8 was used, and the type can be confirmed in Figure 3 below. In conclusion, YOLOv8l was selected. Several models were downloaded and object detection was performed, and the selection was made in consideration of the balance of speed and accuracy. Video size, speed, and efficiency were considered in detecting only cars.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 3. Model type of YOLOv8

The information in Figure 3 is as follows.

- Size (pixels): This indicates the input image size used for training and inference.
- mAP_{val} 50-95: Mean Average Precision, a common metric to evaluate the accuracy of object detection models, where higher values indicate better accuracy. The range 50-95 typically refers to different thresholds used to compute the metric.
- Speed CPU ONNX (ms): This represents the inference speed (in milliseconds) when the model is run on a CPU using the ONNX runtime. Lower values are better as they indicate faster inference.

- Speed A100 TensorRT (ms): This represents the inference speed (in milliseconds) when the model is run on an NVIDIA A100 GPU using TensorRT. Again, lower values are better.
- params (M): The number of parameters in the model, measured in millions. Fewer parameters typically mean a smaller, less complex model.
- FLOPs (B): Floating Point Operations per second, measured in billions. This indicates the computational complexity of the model. Fewer FLOPs typically mean a less computationally intensive model.

2.2.2. Get frame & Set ROI region

Frames are obtained from the video. Read the following frame in the video, and return it with a boolean value indicating whether the frame was read successfully. Considering that the goal is to count the number of parked cars rather than conducting detection for the entire video, it is necessary to set a region of interest (ROI) for the entire parking area and perform detection within that area for proper processing. The figure below shows the ROI region defined in the source video.

2.2.3. Detect cars & Draw boundingbox using detected cars information

Using the YOLOv8 model, cars are detected within the ROI. The YOLO model is applied to the cropped frame to detect objects. Once the cars are detected, bounding boxes are drawn around them using the coordinates provided by the YOLO model. Only car detections with confidence scores greater than or equal to 0.5 are considered.

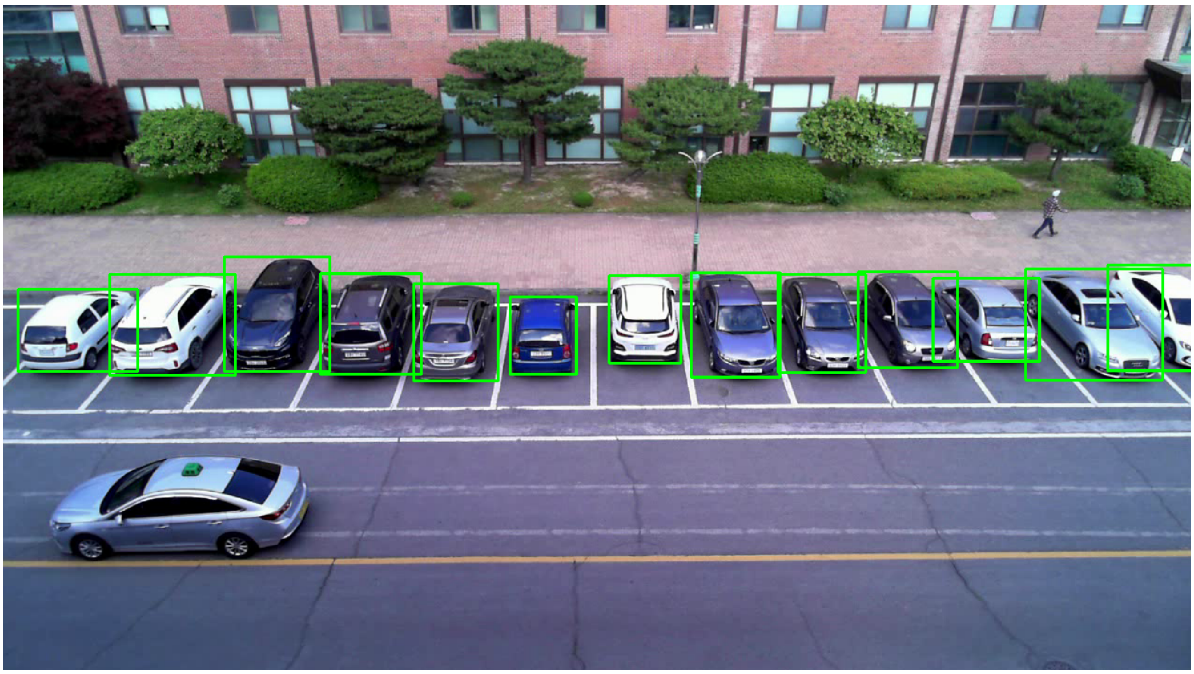

Figure 4. Detect the cars within ROI

2.2.4. Calculate and display the requested value

After detecting cars and drawing bounding boxes, the number of cars is counted. If the number of cars exceeds 13, indicating that all parking spaces are occupied, the available parking space is set to 0. Otherwise, the available parking space is calculated as 13 minus the number of detected cars. Based on these calculations, I displayed the frame, the number of cars parked, and the number of cars on the video. In addition, the number of frames and number of cars are stored in a file named 'counting_result.txt'. In addition, numbers are displayed in 13 parking areas based on the x-coordinate. According to the x-value of the square, a value was allocated to a designated

variable, and compared with the preset value, the number available for parking was set to come out.

Figure 5. Detect the cars and display information

2.2.4. Check accuracy

The difference between the values of the Answer file and counting_result.txt was determined, and the account was obtained through this. The code is as follows.

```
% Load data from the first file
file1 = 'LAB_Parking_counting_result_answer_student_modified.txt';
data1 = readmatrix(file1);

% Load data from the second file
file2 = 'counting_result.txt';
data2 = readmatrix(file2);

% Extract the relevant columns (assuming the same structure for both files)
values1 = data1(1:1501, 2); % first 1501 items from the second column of the
first file
values2 = data2(1:1501, 2); % first 1501 items from the second column of the
second file

% Compute the differences
differences = values1 - values2;

% Find indices where differences are non-zero
diff_indices = find(differences ~= 0);

% Count the number of differences
num_differences = length(diff_indices);

% Display the number of differences
fprintf('Number of differences between the two files: %d\n', num_differences);

check_accuracy = (1500 - num_differences)/1500*100;
```

```
fprintf('Accuracy: %.2f\n', check_accuracy)
```

The result of the code is as follows.

```
Number of differences between the two files: 12
Accuracy: 99.20
```

Figure 6. Result of accuracy

III. Result and Discussion

1. Result

Through Figure 5 and Figure 6, we can confirm the results and their accuracy. The frame count continues to increase up to 6198 frames, after which the process terminates. The number of parked cars changes as the video progresses. Consequently, the number of available parking spaces also fluctuates. The code is designed to detect only cars, not people, and the ROI is set to exclude cars passing on the main road. YOLOv8l model is employed, and its high accuracy is verified. Thus, the initial goal of displaying the number of parked cars and available parking spaces has been achieved. In addition, numbers were assigned to each parking area to indicate the available parking location.

Figure 7. Result images

It was checked through MATLAB whether it was similar to the previously given correct answer file. As a result, there were 12 differences and the accuracy was 99.20.

Accuracy_check.m

```m
% Load data from the first file
file1 = 'LAB_Parking_counting_result_answer_student_modified.txt';
data1 = readmatrix(file1);

% Load data from the second file
file2 = 'counting_result.txt';
data2 = readmatrix(file2);

% Extract the relevant columns (assuming the same structure for both files)
values1 = data1(1:1501, 2); % first 1501 items from the second column of the first file
values2 = data2(1:1501, 2); % first 1501 items from the second column of the second file

% Compute the differences
differences = values1 - values2;

% Find indices where differences are non-zero
diff_indices = find(differences ~= 0);

% Count the number of differences
num_differences = length(diff_indices);

% Display the number of differences
fprintf('Number of differences between the two files: %d\n', num_differences);

check_accuracy = (1500 - num_differences)/1500*100;
fprintf('Accuracy: %.2f\n', check_accuracy)
```

명령 창

```
Number of differences between the two files: 12
Accuracy: 99.20
fx >>
```

Figure 8. Accuracy check

Demo Video

I created a demo video that shows the bounding box of the cars within the parking space only.

Video Link: [Demo Video]

2. Discussion

In this LAB4, the main focus was to detect cars and determine the number of available parking spaces and the number of parked cars. To achieve this, it was essential to choose an appropriate YOLO model. YOLOv8l was selected due to its crucial role in balancing speed and accuracy. After testing various models, it was found that YOLOv8l outperformed other models in detecting cars. This system accurately identified cars within the region of interest (ROI) while excluding vehicles passing on the main road in front of the parking lot. It was also configured to recognize only cars, even though people occasionally passed by. This made it easy to calculate the number of available parking spaces and the number of parked cars. As seen in Figures 5 and 6, the count of

available spaces and parked cars matched the values displayed in the video, confirming the system's high accuracy.

performance	score
Accuracy	99.2%
Precision	99.73%
Recall	99.47%
f1 score	99.60%

3. Trouble shooting

If the YOLO model is executed as it is, pre-trained objects such as a person, a vehicle, etc. are detected. Therefore, the class number of the vehicle of the YOLO model selected to output only the vehicle is checked through the code 'class_list = model.names, print(class_list)'. As a result, the class number of the vehicle was number 2, and through this, the vehicle could be well identified.

The ROI needs to be resized later to match the size of the original frame in order to accurately reflect the detected objects' positions in the original frame. First, the ROI is cropped from the original frame, and this cropped region is then input to the YOLO model for object detection. Subsequently, to accurately display the results of the model on the original frame, the coordinates of the detected objects are adjusted to fit the original size. The resized ROI is then inserted back into the original frame to ensure that the objects are displayed in their correct positions.

IV. Appendix

Python code

```
"""
* author    :   Taegeon Han
* Date      :   2024-06-06
* Brief     :   DLIP_LAB: CNN Object Detection 1 (Parking Management System)
"""

import torch
import cv2 as cv
import numpy as np
from ultralytics import YOLO
import pandas


# Create counting_result.txt and enter a value
def store_count(value):
    with open('counting_result.txt', 'a') as file:
        file.write(value + '\n')

# Find available parking space number
def find_num(order):
    indexes = []
    for i, v in enumerate(order):
```

```
        if v == 0:
            indexes.append(i+1)
    return indexes

# Write the text on the display
def print_text(frame):
    cv.putText(frame,"Frame : " + f"{frame_count}",
(20,585),cv.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv.putText(frame,"Number of cars : " + f"{car_count}",
(20,625),cv.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv.putText(frame,"Number of available parking spaces : " + f"{park_space}",
(20,665),cv.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv.putText(frame,"Available space number : " + f"{find_num(state_list)}",
(20,705),cv.FONT_HERSHEY_SIMPLEX,1,(0,255,255),2)

    cv.putText(frame,"Area Number: ", (5,230),cv.FONT_HERSHEY_SIMPLEX,0.7,
(0,255,255),2)
    cv.putText(frame,"1", (100,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"2", (213,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"3", (312,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"4", (412,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"5", (500,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"6", (588,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"7", (678,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"8", (761,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"9", (855,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"10", (941,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"11", (1031,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"12", (1133,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
    cv.putText(frame,"13", (1237,265),cv.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)


# Load the Model
model = YOLO('yolov8l.pt')

# # Check the class number of car
# class_list = model.names
# print(class_list)
car_num = 2

# Open the video camera
cap = cv.VideoCapture('DLIP_parking_test_video.avi')


# If not success, exit the program
if not cap.isOpened():
    print('Cannot open camera')
    exit()

# Define the region of interest (ROI)
roi = [[0, 255], [1280, 420]]
x1, y1 = roi[0]
x2, y2 = roi[1]

frame_count = 0

park_point = [110, 213, 312, 412, 500, 588, 678, 761, 855, 941, 1031, 1133,
1237]
```

```
park_len = len(park_point)

while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()

    if success:
        # Crop the frame to the ROI
        roi_frame = frame[y1:y2, x1:x2]

        # Run YOLOv8 inference on the cropped frame
        results = model(roi_frame)

        # Initialize the car count
        car_count = 0

        state_list = [0 for no_use in range(park_len)]
        # Set only cars to come out
        for result in results:
            boxes = result.boxes.xyxy
            scores = result.boxes.conf
            class_order = result.boxes.cls

            car_rect = boxes[class_order == car_num]
            car_confidence = scores[class_order == car_num]



            for box, score in zip(car_rect, car_confidence):
                if score >= 0.5:
                    xmin, ymin, xmax, ymax = box    # Allocate 4 coordinates
                    xmin, ymin, xmax, ymax = int(xmin), int(ymin), int(xmax),
int(ymax)     # Convert to an integer to make a rectangle
                    cv.rectangle(roi_frame, (xmin, ymin), (xmax, ymax), (0, 255,
0), 2)

                    # Making state_list to 1 if park_point is located between
xmin and xmax
                    for i in range(park_len):
                            if xmin < park_point[i] < xmax:
                                state_list[i] = 1

                    car_count += 1

            # If car_count is more than 13, available space is '0'
            if(car_count > park_len):
                park_space = 0

            # Calculate available parking spaces
            park_space = park_len - car_count

            # Print what we want
            print_text(frame)

            # Write the frame_count and car_count to the file
            counts = f"{frame_count},{car_count}"
            store_count(counts)
```

```
        #Resize the ROI to match the original region
        resize_ROI = cv.resize(roi_frame, (x2-x1, y2-y1))

        # Replace the ROI part in the original frame with the annotated ROI
frame
        frame[y1:y2, x1:x2] = resize_ROI

        # Display the annotated frame
        cv.imshow("Video", frame)

        # Increase frame_count value by 1
        frame_count += 1

        # Break the loop if 'q' is pressed
        if cv.waitKey(1) & 0xFF == ord("q"):
            break

    else:
        # Break the loop if the end of the video is reached
        break

# Release the video capture object and close the display window
cap.release()
cv.destroyAllWindows()
```

Matlab code

```
% Load data from the first file
file1 = 'LAB_Parking_counting_result_answer_student_modified.txt';
data1 = readmatrix(file1);

% Load data from the second file
file2 = 'counting_result.txt';
data2 = readmatrix(file2);

% Extract the relevant columns (assuming the same structure for both files)
values1 = data1(1:1501, 2); % first 1501 items from the second column of the
first file
values2 = data2(1:1501, 2); % first 1501 items from the second column of the
second file

% Compute the differences
differences = values1 - values2;

% Find indices where differences are non-zero
diff_indices = find(differences ~= 0);

% Count the number of differences
num_differences = length(diff_indices);

% Display the number of differences
fprintf('Number of differences between the two files: %d\n', num_differences);

check_accuracy = (1500 - num_differences)/1500*100;
fprintf('Accuracy: %.2f\n', check_accuracy)
```