Experimental setup

Based on the problem, recognise the spaces occupied and free in a parking lot automatically. I assume that this problem is an object detection problem. Therefore, I began by searching for one public dataset with images from a parking lot annotated to train my network, Yolov7, that was selected to make object detection.

In the next sections, I will describe each step of this project.

Dataset

To develop this project, it used the PK Lot dataset (https://www.kaggle.com/datasets/ammarnassanalhajali/pklot-dataset). This dataset contains 12,416 images annotated with parking lots, with the classes “space-occupied” and "space-empty." However, these images are annotated in COCO format, which is not compatible with the format of the Yolov7 network. Then, through the Roboflow tool (https://roboflow.com/), I upload the dataset and convert it from the COCO format to the Yolov7 format.

This tool allowed me to save the dataset in the new format and, after a few lines of code, load the dataset in my notebook. As can be seen in the challenge-parking-lot-detection.ipynb file. The dataset was split into 70% to train, 20% to validate, and 10% to test.

Training

After loading the dataset, it cloned the Yolov7 network and downloaded its weights. It was trained for 20 epochs.

Inference

To test the network, one image from the test dataset was selected, as well as the best weights obtained during network training. This way, with the execution of the command “!python '/kaggle/working/yolov7/detect.py' –weights '/kaggle/working/yolov7/yolov7_size640_epochs25_batch4/weights/best.pt' --conf 0.25 --img-size 640 --source '/kaggle/working/Parking-Lot-1/test/images/2012-12-23_16_40_13_jpg.rf.8012842ec5d1208559c0ea628c4fdcba.jpg'”, it was possible to infer the result of this image. After the inference is concluded, it shows the count of each class present on the image. As can be seen in the challenge-parking-lot-detection.ipynb file.

How the image is inferred is saved in the “runs/detect/exp/” directory. In another cell, the image is read and shown with the bounding boxes.

Deployment

To deploy this solution using docker compose. I started by creating an API in Flask, which consists of deploying this app in a Docker container and then using docker compose to start a service, which allows a POST request and the application to display the result (number of seats available or occupied, and the image with the bounding boxes).

So, my idea was to use the code in the python file "detect.py", which is in the Yolov7 repository, as a basis and adapt it to my solution. As you can see from the "app.py" file. The files in the "utils"

and "models" directories have therefore been extracted from the Yolov7 repository because they are important for the operation of my application.

However, when I ran the "docker compose up" command to containerise my application and start the service, an error was triggered due to Opencv dependencies that I haven't been able to resolve so far. Therefore, the deployment component is not working now.