Parking Lot Utilization

https://github.com/sascott7/parkingLot

Sarah Scott

sascott7@crimson.ua.edu

sascott2002@gmail.com

901-428-5517

This document describes the processes and code used in the system to track the utilization of parking lots. The following information is for code running in Python 3.10.

Project Overview

The project's goal is to develop a cost-effective method of tracking parking lot utilization with cameras. We aim to develop software that can dynamically identify the location of the parking spots so the program can be run on various parking lots without additional manual input. After locating the parking spots, the system tracks utilization rates by analyzing the current location of cars in relation to the parking spots to determine if each spot is full or empty and records the data.

The approach to learning the location of the parking spots is based on the idea that, generally, cars are parked correctly in spots. If we gather data over a long period of time on where cars are parking, we can use that data to estimate the location of parking spots. Object detection software is used to identify the location of cars in an image. Running the object detection on collection of images of a parking lot results in a set of data corresponding to locations where cars have parked in the lot. A machine learning algorithm is used to analyze the data and separate the set of detections into groups where each group is a parking spot.

The center of each parking spot is saved to be used for tracking utilization. Current images of the parking lot can be used to track the utilization rate. The object detection program is run to determine a set of locations of cars currently parked in the lot. This set is compared to the set of parking spot points to determine whether each spot is currently empty or full. The status of each spot is recorded to track over a period to determine the utilization rate of the parking spots.

Background Information

Yolov5

YOLO (you only look once) is an open-source real-time object detection algorithm that is very fast. The algorithm takes an image and determines a bounding box and classification for every identified object.

Version 5 was chosen for its ease of use as part of the PyTorch library. An older version was also chosen for greater support available online. Yolov5 returns the minimum and maximum x and y values (each edge) of the bounding box in addition to the confidence and classification of each detection.

More information on the use of yolov5 can be found here:
https://docs.ultralytics.com/yolov5/tutorials/pytorch_hub_model_loading/

Clustering Algorithms

Clustering algorithms group a set of objects so that each group, or cluster, is more similar to each other than other groups. This is typically accomplished by mapping features to data points and grouping close points. In our case, the coordinates of the detections are the data points, and the

clustering algorithm will group the detections into clusters of points that are closest to each other which represent each parking spot.

Several different clustering algorithms were examined to determine the best fit for this project. After testing a variety of algorithms, it was found that with the proper tuning, all of them had a similar accuracy level. The speed and tuning process were used to choose the algorithm.

Agglomerative clustering was chosen because it can be scaled for a large dataset with some adjustments and the parameter passed to the algorithm is a distance threshold. Since a distance is the parameter that improves the performance of the algorithm, the best number will likely be the same for images where the parking lot is at the same scale. The value could also potentially be estimated based on the width of the cars given from the object detection as the width represents the size of the parking spot and therefore the distance between the clusters. Additionally, this algorithm will ignore outliers leaving them in their own cluster.

The agglomerative algorithm works by initially treating each point as its own cluster. The two closest clusters are then combined into a larger cluster. This process is repeated until the distance between the closest clusters is greater than the distance threshold that was passed in as a parameter.

KD Trees

KD trees are used in this project to calculate the nearest neighbor of points. A KD tree is a data structure organized in a similar manner to a binary search tree except the nodes are points in space. This structure is efficient for searching for points in space like the closest point to another.

Required Packages

Numpy

> Numpy is used to create an array of detection points and store the array of parking spot locations

Pillow

> Pillow is used to open the images in a format that can be used by the object detection software. It is also used to draw points on an image to visualize the detection and parking spot points.

Scikit-learn

> Scikit-learn is a machine learning library. The agglomerative clustering algorithm from this package is used to determine the parking spots.

PyTorch

PyTorch contains the code for Yolov5 which is used for object detection. Specific instructions for downloading this package with support for cuda can be found on their website. Using the GPU greatly reduces the time required for detections when compared to CPU only.

Argparse

Argparse is used to retrieve command line arguments.

Additional requirements if capturing images from a YouTube livestream

OpenCv

OpenCv is used for video processing

Dateutil

Dateutil is used as part of the capture script to record the time of capture.

Yt-dlp

Yt-dlp is a command line program that supports downloading videos from YouTube.

Locate Parking Spots

The program to identify the location of the parking spots requires a folder of images. The program loops through the images and runs yolo on each of them. The detections for each image are all added into one list. The list is passed to the agglomerative clustering algorithm and the center of each cluster returned is saved to disk as a NumPy array.

Usage

The program requires an argument, the path to the directory where the parking lot images are stored. There are also optional arguments which can be added by using a flag as follows:

-s or --save: path to the directory where any information should be saved (including the parking lot spots array and image with spots if included)

-i or --image: path to an image to be used to draw the parking spot locations

Functionality

The script calls the detectFolder which loops through the folder of images to open each image using Pillow and pass it to the object detection model. The detections for each image are combined into a large list of detections and returned. This list of detections is then converted into a numpy array.

The runAgglomerative function is then run. An agglomerative clustering model is created and run on the detections array which returns a list the same length as the input where the element in

the list represents the index of the cluster to which the input belongs. A list is then created where each element contains a list all of the detection coordinates that were assigned to that cluster. The center of each cluster is then calculated by finding the average x and y value of the points in that cluster. The list of cluster centers is then filtered by checking the number of detections that were assigned to that cluster to ignore any outliers. The accepted cluster centers are returned and saved as an array representing the parking spots.

Track Utilization

This program takes a saved numpy array of parking spot locations and determines whether each spot is full or empty in a given image. The program saves a copy of the image with the parking spots drawn on in green or red based on whether the spot is currently full or empty. In the future, the program should use the empty/full data to calculate a utilization rate.

Usage

Two arguments are required: the path to the saved numpy array containing the parking spot locations and the path to the folder containing the images to track the utilization of. There is also an optional argument:

-s or --save: path to the directory where the copy of the images with the drawn empty or full dots should be saved

Functionality

The program loops through each image in a directory and determines the use of each image. The detection function passes the image to the object detection model and returns an array of the detections in the image.

Calculate Utilization Rate determines whether each spot is full or empty. A KD Tree is constructed for the parking spot coordinates and for the detection coordinates. The parking spot nearest neighbors and detection nearest neighbors are then determined. Each spot is then looped through. If the parking spot and a detection are both the nearest neighbor of each other and the distance between them is shorter than a certain amount, then the parking spot is marked as full.

Utility

The utility program contains code that is used by multiple programs.

The detect function passes the given image to the given model which returns the detection results in a pandas data frame. The edges of the bounding box of the detection are given, so the center of the box is calculated. A string containing the classification and the center is added to the list of detections. This list of strings is returned.

Load model checks for a saved local copy of the model and loads either the local model or retrieves the model from the source.

Create array takes a list of strings containing the detection information. If the detection is a car or a truck (class 2 or 7) then the string is converted to a list. These lists are combined into a large list of lists and converted into a NumPy array which is returned.

Capture Livestream

The capture livestream script takes a YouTube livestream URL and saves an image of the video every ten minutes.

Usage

Two arguments are required: the link to a YouTube livestream and the path to the folder in which to save the images.

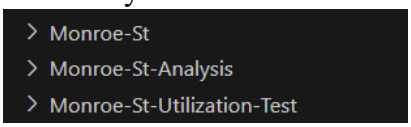
Functionality

The get_url function uses the yt-dlp program to convert the YouTube livestream link into a link to a m3u8 stream link. This link expires after some time, so the script updates the link every five hours. The stream link is passed to the capture script which was provided by Stephen Kirby. Calling this script returns an image of the video which is then saved to the provided folder.

Example Use

The following is an example of the steps taken to run the existing program using a YouTube livestream

1. Create any desired directories

   In this case, Monroe-St will contain the images saved from the livestream. Monroe-St-Analysis will be the directory where all of the information is saved. Monroe-St-Utilization-Test has the images for tracking utilization.

2. Save images from livestream:

   ```
   rwdougherty@chanstag-System-Product-Name:~/parkingLot$ python capture_livestream.py https://www.youtube.com/watch?v=mwN6l3O1MNI /home/rwdoughert
   y/Monroe-St
   ```

3. Wait for enough images to be captured

4. Determine parking spots

   ```
   rwdougherty@chanstag-System-Product-Name:~/parkingLot$ python LocateParkingSpots.py /home/rwdougherty/Monroe-St -s /home/rwdougherty/Monroe-St-Analysis -i
    /home/rwdougherty/Monroe-St-Analysis/image.jpg
   Detecting Vehicles...
   Using cache found in /home/rwdougherty/.cache/torch/hub/ultralytics_yolov5_master
   /home/rwdougherty/.local/lib/python3.10/site-packages/matplotlib/projections/__init__.py:63: UserWarning: Unable to import Axes3D. This may be due to mult
   iple versions of Matplotlib being installed (e.g. as a system package and as a pip package). As a result, the 3D projection is not available.
     warnings.warn("Unable to import Axes3D. This may be due to multiple versions of "
   Detection Duration: 0:03:13.665471
   Locating Parking Spots...
   Saving Parking Spots...
   Done
   Duration: 0:03:25.284536
   ```

   The blue dots are all of the detections from all of the images. The green dots are the location of the identified parking spots.

5. Track if spots are full or empty in a particular image

   ```
   rwdougherty@chanstag-System-Product-Name:~/parkingLot$ python TrackUtilization.py /home/rwdougherty/Monroe-St-Analysis/parkingLocations.npy /home/rwdoughe
   rty/Monroe-St-Utilization-Test -s /home/rwdougherty/Monroe-St-Analysis
   ```

The blue dots are the detection from this image. The green dots are full parking spots. The red dots are empty parking spots.

To Do

- Update Locate Parking Spots to use the size of the bounding box to estimate the best parameter to use for the agglomerative algorithm
- Update the parameter of how many cars to consider a cluster a parking spot to be a fraction of the total number of images instead of a hard coded number
- Add calculation of utilization to Track Utilization instead of just drawing full and empty on the images
    - Determine how to calculate utilization rate (minutes used/minutes total – have to keep track of time when spot was first full and add the time when spot became empty)
    - Overall utilization rate of parking lot
    - Utilization rate of each spot
    - Potentially track the utilization rate at different times of day
- Decide the best distance to use to determine if a spot should be considered full