Verkada - Computer Vision Coding Assignment

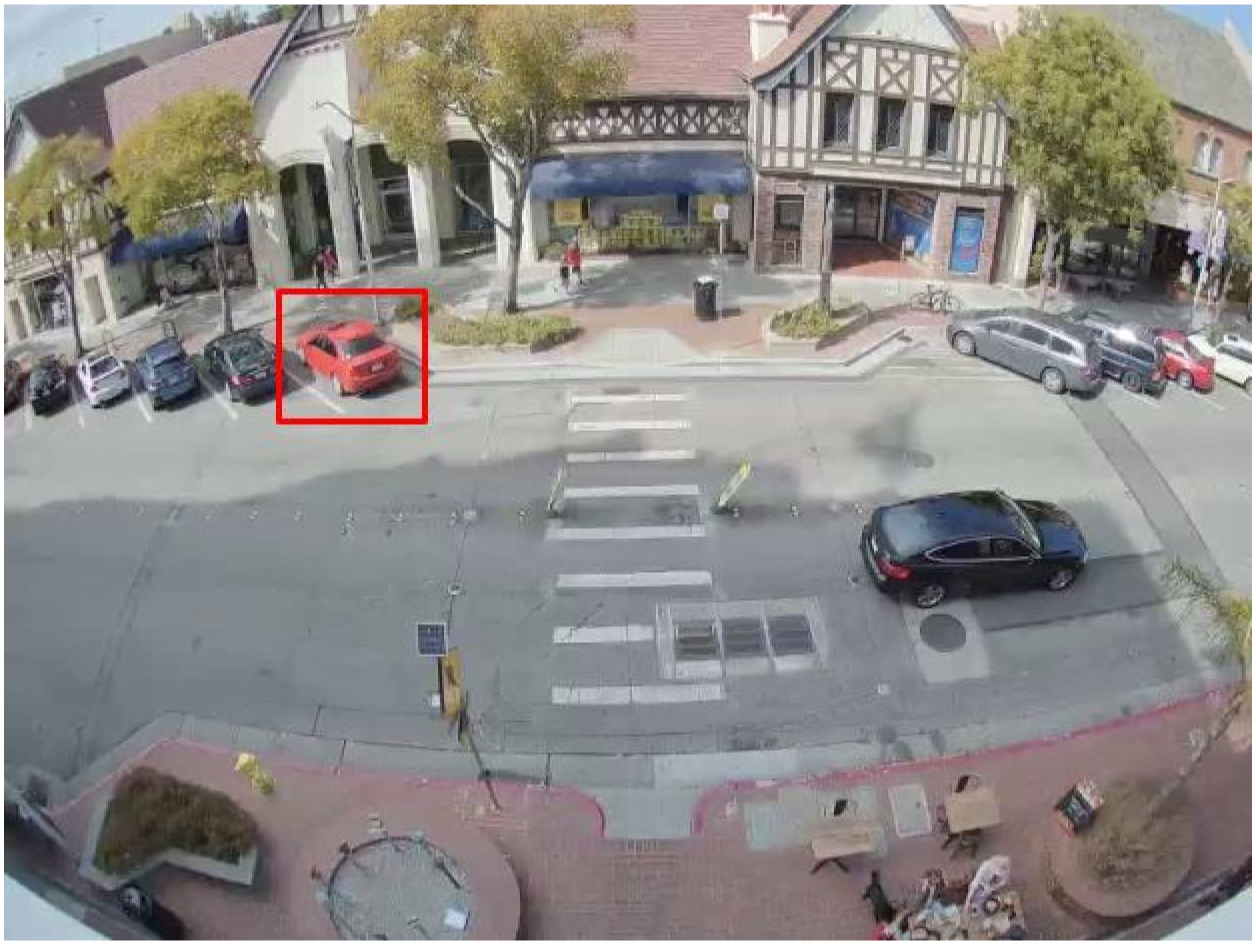
confidential, please do not share

We'll be working with data from a real camera, so please do not use a public github repository for storing your code. If you want to store your code in the cloud, please use a private repository (perhaps bitbucket, which is free).

Please note: we have a lot of people interested in working with us on computer vision, so we have to be selective. Show us why we have to work with you!

Objective

We've placed a camera outside overlooking a street. The parking spots have a time limit of several hours. You have access to approximately two weeks of low-res video footage. **Your task is to identify cars parked in the spot marked below.**



We'll break this goal down into four steps to make the task simpler.

Data Spec

Video data is at https://hiring.verkada.com/video/[timestamp].ts, where [timestamp] is a unix timestamp in seconds.

- First timestamp: 1538076003 (Thursday, September 27 @ 12:20:03pm)
    - https://hiring.verkada.com/video/1538076003.ts
- Last timestamp: 1539326113 (Thursday, October 11 @ 23:35:13pm)
    - https://hiring.verkada.com/video/1539326113.ts
- A full list of ts files is inside the following text file:
    - https://hiring.verkada.com/video/index.txt

If you were to download all the videos, you'd need approximately 50GB. **Don't worry, we'll only work with a subset of this data!**

Requirements

- You may use any programming language (we prefer Python)
- Using CV libraries is fine!
- Write shell scripts or a Jupyter notebook. Dockerfiles are appreciated if you need a lot of dependencies.

Step 1 - Write a Download Script

Write a script or function that downloads a .ts file and extracts the first frame as a jpg image.

example script

```
> ./fetch-and-extract 1538076003

downloading https://hiring.verkada.com/video/1538076003.ts...
extracting image... wrote 1538076003.jpg
```

Step 2 - Extract the car in the box shown above

We'll only look at a single parking spot (the one shown above in the red bounding box). Write a script or function that accepts a single image and runs the yolo3 algorithm (or the algorithm of your choice) on the spot above.

example script

```
> ./has-car 1538076003.jpg
```

```
running yolo3 on 1538076003.jpg...
no car detected!
```

Step 3 - Compute the similarity between two cars

Write a script or function that accepts two images. Assume that both images contain a car. Compute whether or not the image is of the same car. The result should be a boolean.

example script

```
> ./fetch-and-extract 1538076183
> ./fetch-and-extract 1538076179
> ./is-same-car 1538076179.jpg 1538076183.jpg

comparing 1538076179.jpg and 1538076183.jpg... same car!
```

Step 4 - Put it all together!

Write a script or function that accepts a time range and outputs each car that was detected and how long it was parked for (approximately). You'll probably want to get the index file to see which timestamps are available for download.

The output format or return value for all steps is up to you. A possible sample output is shown below for a python shell script:

```
> curl https://hiring.verkada.com/video/index.txt > index.txt
> python3 analyze-cars.py --index index.txt --start 1538076003 --end 1538078234

analyzing from 1538076003 to 1538078234
found car at 1538076175. parked until 1538077874 (28 minutes).
... wrote output/1538076175-28min.jpg
found car at 1538077954. parked until 1538078198 (4 minutes).
... wrote output/1538077954-4min.jpg
no more cars found!
```

Hint: if you don't have a GPU, this might take a really long time. You can speed things up by not sampling every image and returning approximate results. We're not looking for 100% accuracy.

Discussion

Provide a README file that contains the following:

- instructions for running the code, including a list of dependencies (Dockerfiles and/or iPython notebooks appreciated!)

- any implementation details/notes
- when do you think this algorithm would work well? when would it not?
- what would you suggest for future exploration?

Bonus

Consider the following optional challenges:

- Download multiple images in parallel (add a `--concurrency` flag in Step 1)
- Analyze the other nearby parking spots.
- Are there other algorithms that work better at identifying cars?
- Can you detect the color of the car?
- Can you detect parking spot (given an arbitrary image)?