Computer Vision Project Summary
IDC MSc

Date: March 21, 2021

Authors: Yossi Gavriel, Ofir Nesher

Title: **Parking Space Detection with Python & OpenCV**

Introduction:
Two young Tel-Avivians, desperate from looking for parking in the city, trying to solve the 11th plague of Egypt.
Inspired by the "Ahuzot Hahof" parking lot at HaBima Square - we aim to implement a smart parking lot system that counts the number of cars inside and detects free parking space in the lot using elements and algorithms from the Computer Vision realm and topics learned in the course this last semester.

Goals:
The project will focus on Computer Vision methods learned in the lectures to detect empty parking spots in an image/video of a parking lot (given as inputs) and marking when these are becoming available or occupied.
Also, we implement a feature (section 3 in "Approach" below) for being able to count how many cars go in/out of the lot and also draw contours around moving objects in a pre-defined area (around the gate for example).

The outputs will be the given video with all the markings on top of the frames.

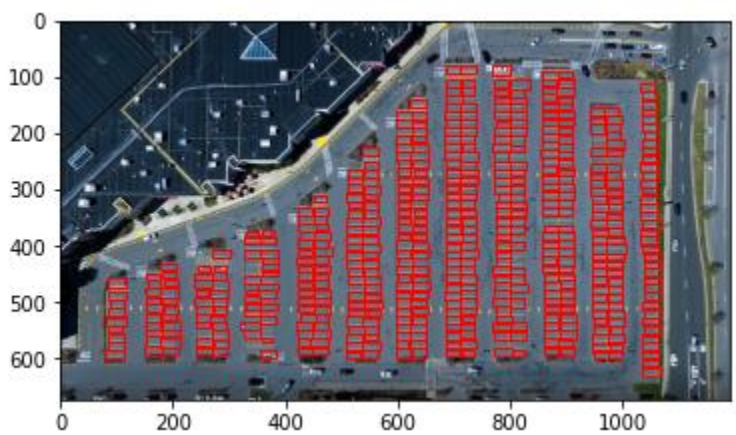
- Footage needs to be still (CCTV, for example).

Approach:
We have a few approaches for implementing a thorough end-to-end smart parking lot system, each is a different feature:

1. The first feature is the ability to **detect and count how many cars go in/out of a parking lot** and draw contours around moving objects in a region of choice in the video.
   This enables monitoring the capacity and also to extract more data like calculating the average time a car is parked in the lot.

2. A “naive” approach for detecting free space - we manually mark empty spots (rectangles) in an image using the "Label Image" tool which created an XML file that contains the parking spot objects (the x, y, and the label) on the image of the empty parking lot we photoshopped.
   Then, we monitor the drawn rectangles to see if the pixels within them were changed and therefore concluding that the spot is occupied/freed again.
   This approach is partially implemented in some CV tutorials and we implemented this approach from scratch as an independent project and improved it.
   We also made the algorithm more robust and more accurate overall, using various metrics for deciding if the spot is occupied or free, such as histogram comparison (learned in class), correlations between spots, and sequence matching.
   According to these metrics, we learn and classify if a spot is empty/occupied.

   The photoshopped empty image after we marked all spots manually:



   The output of the classification on the video:

3. Similar to the above approach where each spot is compared to its own location in the empty lot's image, we also implemented the same scenario where all parking spots are compared to a single "basic" example of a parking spot. This gives another flavor to our implementation.

4. Another small adjustment to detecting the parking spots on the empty lot's image was using code and not manually.
   We used the Canny Edge Detection algorithm, removed noise, and then used the Hough Line Transform method for detecting straight lines. After processing the results a little more, we came up with this (which is a little more structured):

5. We also count the number of empty/occupied spots and analyze the results using the accuracy, precision, recall, and f1 scores.

After sorting the results by each metric, we can see that “var_change_detection” comparison method got the highest results (99% accuracy) in the “median” approach and one of the faster methods:

Sorted results for median approach by accuracy_score

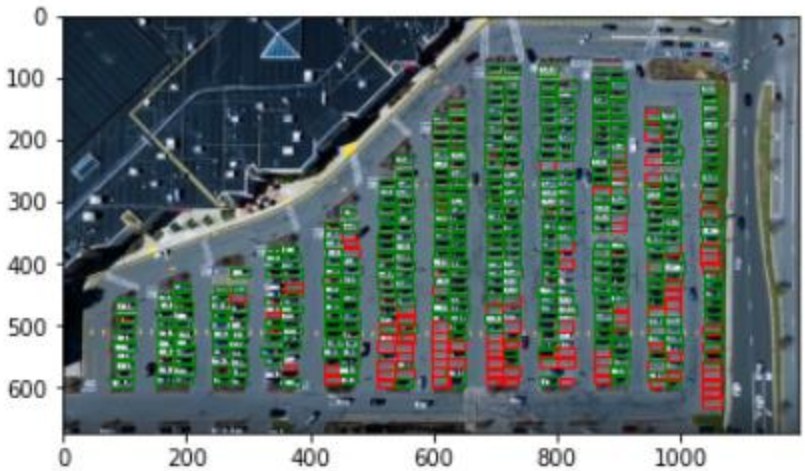
	accuracy_score	running_time_seconds
var_change_detection	0.990177	0.402
predictImage	0.986248	18.432
absdiff	0.980354	0.409
MSE	0.956778	0.380
histogram	0.941061	1.653
change_dection	0.819253	10.506
kendal	0.795678	1.454
cosine_change_detection	0.190570	0.654

Sorted results for background approach by accuracy_score

	accuracy_score	running_time_seconds
absdiff	0.992141	0.409
var_change_detection	0.986248	0.402
predictImage	0.986248	18.432
MSE	0.982318	0.380
histogram	0.944990	1.653
change_dection	0.915521	10.506
kendal	0.846758	1.454
cosine_change_detection	0.783890	0.654

Visual results:

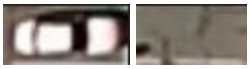
	precision	recall	f1-score	support
0	1.00	0.95	0.97	98
1	0.99	1.00	0.99	411
accuracy			0.99	509
macro avg	0.99	0.97	0.98	509
weighted avg	0.99	0.99	0.99	509



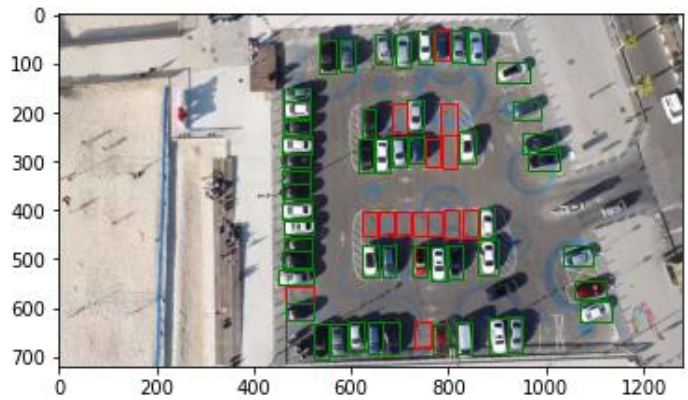

```
approach: predict_by_median_image
func: var_change_detection
Free parking lots: 93
running time in seconds: 0.401706
```

6. Using a **neural network**, we look at each parking spot that we marked (both manually as in section 2, and also using code, and in particular using Canny Edge Detection algorithm and Hough Line Transform method) and make a prediction of occupied or not.
   This approach ended up working well:
   The model learns what is an occupied and a free parking spot based on labeled images we extracted, such that during the video, it can classify if a spot's status and if it changed between frames.

   Examples of the training images:



7. Finally, the best part is to see it "in the real world" so we ran the model on our own footage we took with our drone:




```
func: absdiff
Free parking lots 14
num_predictions - 64
true_positive - 13, true_negative - 50, false_positive - 0, false_negative - 1
accuracy: 0.984375, presicion: 1.0, recall: 0.9285714285714286, f1: 0.962962962962963
```

<u>Methods we learned in class and are used</u>:

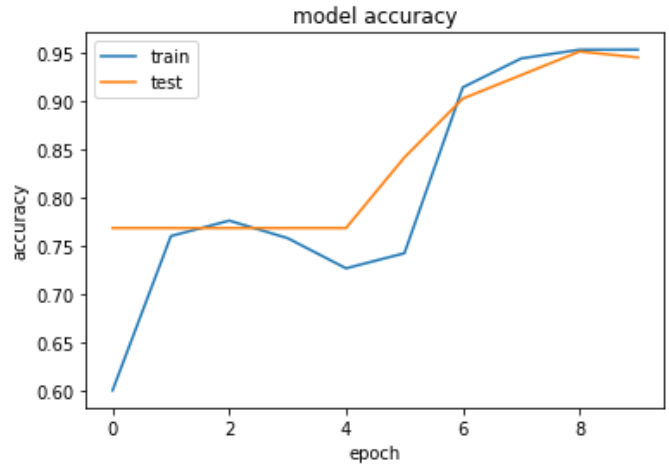
- Image processing (noise removal for example)
- Change Detection
- Edge Detection
- Deep Learning
- etc.

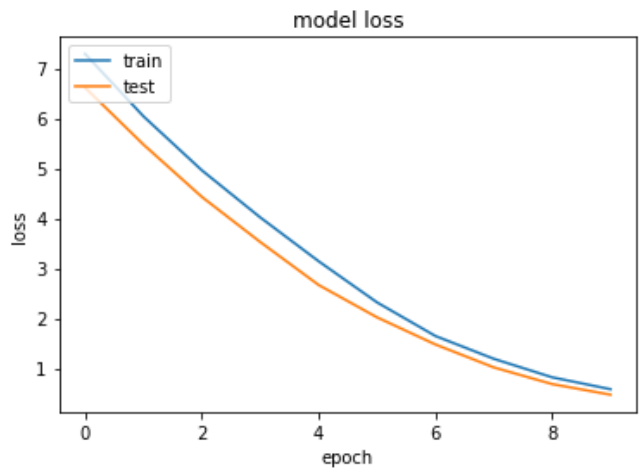
<u>Summary of experimental results</u>:

- The accuracy of the neural network is high and we are happy with the final result.
- Playing around with the different metrics was fun and we got interesting data:

	f1_score	precision_score	recall_score	accuracy_score
MSE	0.935791	0.908333	0.973236	0.956778
median_change_dection	0.773384	0.752126	0.872536	0.819253
var_change_detection	0.983888	0.993990	0.974490	0.990177
cosine_change_detection	0.160066	0.095472	0.494898	0.190570
sequence_matcher	0.446739	0.403733	0.500000	0.807466
predictImage	0.987460	0.983778	0.991248	0.992141
kendal	0.747144	0.731118	0.846281	0.795678
absdiff	0.967109	0.988124	0.948980	0.980354
histogram	0.892033	0.965986	0.846939	0.941061

- The Deep Learning implementation gave us these numbers:





- The results on our drone footage are pretty good and we were excited to go out and test it, it gave it a more personal aspect.

Future work ("What would you do if you had more time"):

- It would be nice to have a model which is able to identify cars in an image where the view is from high above.
  The popular COCO dataset which is large-scale object detection, segmentation and captioning dataset trained on many images of objects (and cars in particular) did not help us because our input video's view is too high up (although we started with it at the beginning).
- Furthermore, we would have liked to play around and try to run some of the code on the GPU, as we learned in class, but we did not have enough time to dig into it.
- We implemented some features in a few notebooks and we would like to combine all features in a single user-friendly online web-application such that a user would be able to upload a video and get the processed video back or to implement a system that sends an SMS anytime a spot is freed :)

Resources:

- CV/ML tutorials and GitHub repositories from around the web: This paper (Image Handling and Processing for Efficient Parking Space Detection and Navigation Aid),
  Deep Learning approach, Olga's naïve partial-implementation, a report about the process we are trying to implement

(Just to name a few which we were inspired by and tuned)

- Our own code and lecture slides from the semester (from the Computer Vision course and the Deep Learning course homework assignments).