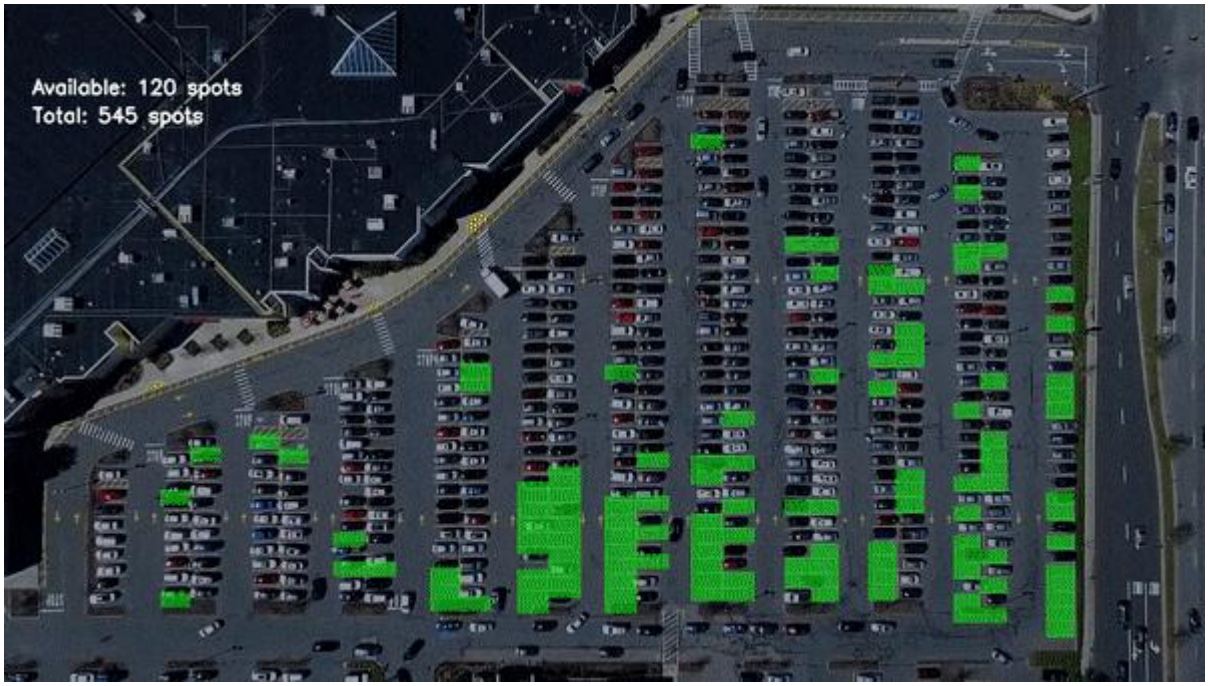
# AI ENABLED CAR PARKING SYSTEM OPEN CV

## Introduction

How many times has it happened to you that you are searching for a parking spot by driving around and around the parking lot. How convenient would it be if your phone could tell you exactly where the closest parking spot is!

It turns out that this is a relatively easy problem to solve using deep learning and OpenCV. All that is needed is an aerial shot of the parking lot. See GIF below where my model is highlighting all available parking spots on the LA airport parking lot as well as displaying a count of how many spots are available. And best of all this can work real time!



Real time parking spot detection

You can find the code I used on my Github repo.

### Overview of the steps

There are two main steps in building this parking detection model:

1. Detecting the position of all available parking spots
2. Identifying if a parking spot is vacant or occupied

Since the camera view here is mounted, we can use OpenCV to do a one time mapping of each parking spot. Once you know the location of each parking spot, you can use deep learning to make a prediction on whether it is vacant or not.

## Detecting the position of all available parking spots

The basic idea I used for detecting the parking spots was that all parking spot dividers here are horizontal lines and the parking spots in a column are roughly equally spaced apart. I first used Canny edge detection to get an edge image. I also masked out the area where no parking spots are present. See below:

Canny edge detection output

I then did hough line transform on the edge image which draws out all lines it can identify. I isolated the horizontal lines by only selecting lines with slope close to zero. See output of hough transform below:

Line Detection using HoughLines

As you can see the hough lines does a reasonably good job of identifying parking lines but the output is not clean — several parking lines are detected many times while some are missed. So how do we clean this up?

I then used observation and intuition to proceed from here. Using the coordinates returned by hough lines, I clustered x- observations to identify the main parking lanes. The clustering logic works by identifying gaps in x coordinates of detected lane lines. This allowed me to identify the 12 parking lanes here. See below

Identify parking lanes by clustering x-coordinates from hough lines

If all this looks complicated, don't worry — I have documented the code step by step in my jupyter notebook on github.

Now that I know quite well where all the parking lanes was, I identified each individual parking spot by assuming all parking spots were the same size which is a reasonable assumption. I eyeballed the results to ensure I was capturing the boundary between spots as accurately as possible. I was finally able to mark out each parking spot. See below.

Each parking spot marked out

Now we are done — We can assign each spot an ID and save its coordinates in a dictionary. I pickled this dictionary so it can be retrieved later. This is possible here since the camera is mounted and we don't need to compute the location of each spot in its view again and again.

**Identifying if the spot is marked or not**

Now that we have the parking map, there are a few ways I think we can identify if the spot is occupied or not:

1. Use OpenCV to check if the pixel colour of a spot aligns with the colour of an empty parking spot. This is a simple approach but prone to errors. For example change in lighting will change the colour of an empty parking spot which will make it difficult for this logic to work through the day. Also if is possible that

the logic will confuse cars of grey colour as an empty parking spot

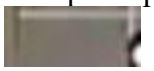
2. Use object detection to identify all cars and then check if the location of the car overlaps with a parking spot. I did try this and found that object detection models that can work real time really struggle with detecting objects of small size. No more than 30% of all the cars were detected

3. Use a CNN to look at each parking spot and make a prediction of occupied or not. This approach ended up working best

To build a CNN, we need to have images of parking spots with and without cars. I extracted an image of each spot and saved it in the folder and then grouped those images into occupied or not. I have also shared this training folder on Github.

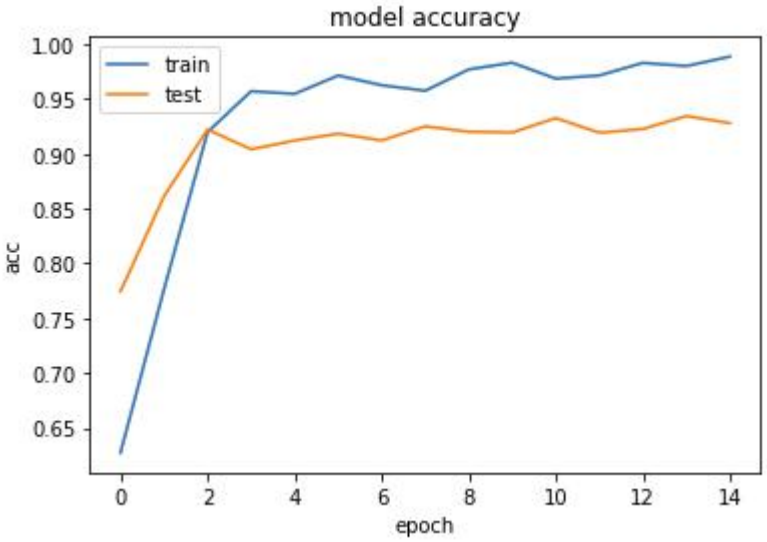
Since there are close to 550 parking spots in an image of dimension 1280x720, each parking spot is only about 15x60 pixels in size. See sample images below of empty and occupied spots:


Occupied spot


empty spot

However since the occupied spots and empty spots look quite different it shouldn't be a challenging problem for CNN

However I only had around ~550 images for both the classes so decided to use transfer learning by taking the first 10 layers of VGG and adding a single softmax layers to the output of VGG model. You can find the code for this transfer learning model here. The model got to a validation accuracy of **94%**. See below:




CNN model test and train accuracies

Now I combined the parking spot detection with the CNN predictor to build a parking spot detector. It works amazingly accurately.


Empty Spot Predictions.

I have also included in the notebook, code to run this on a video stream.

## Conclusion

I am amazed at how easy it has now become to chain different tools and use deep learning to build practical applications. I did this work in two afternoons.

Couple of additional ideas for further exploration:

- Would be great if the parking spot detection logic can be extended to work on any parking map possibly using deep learning. OpenCV has the limitation of requiring tuning for each use case
- The VGG model used in the CNN is quite a heavy model. Would love to experiment with lighter weight models

I have my own deep learning consultancy and love to work on interesting problems. I have helped many startups deploy innovative AI based solutions.