Parking Lot Vacancy Detector

Introduction

A public infrastructure has various parking lots. The parking lots get completely occupied very often and the public visiting the infrastructure spend too much time looking for a parking space, unaware that the parking lot is completely occupied. They would like to implement an automated solution to convey this information by displaying the number of available parking spaces at the entrance to the parking lot.

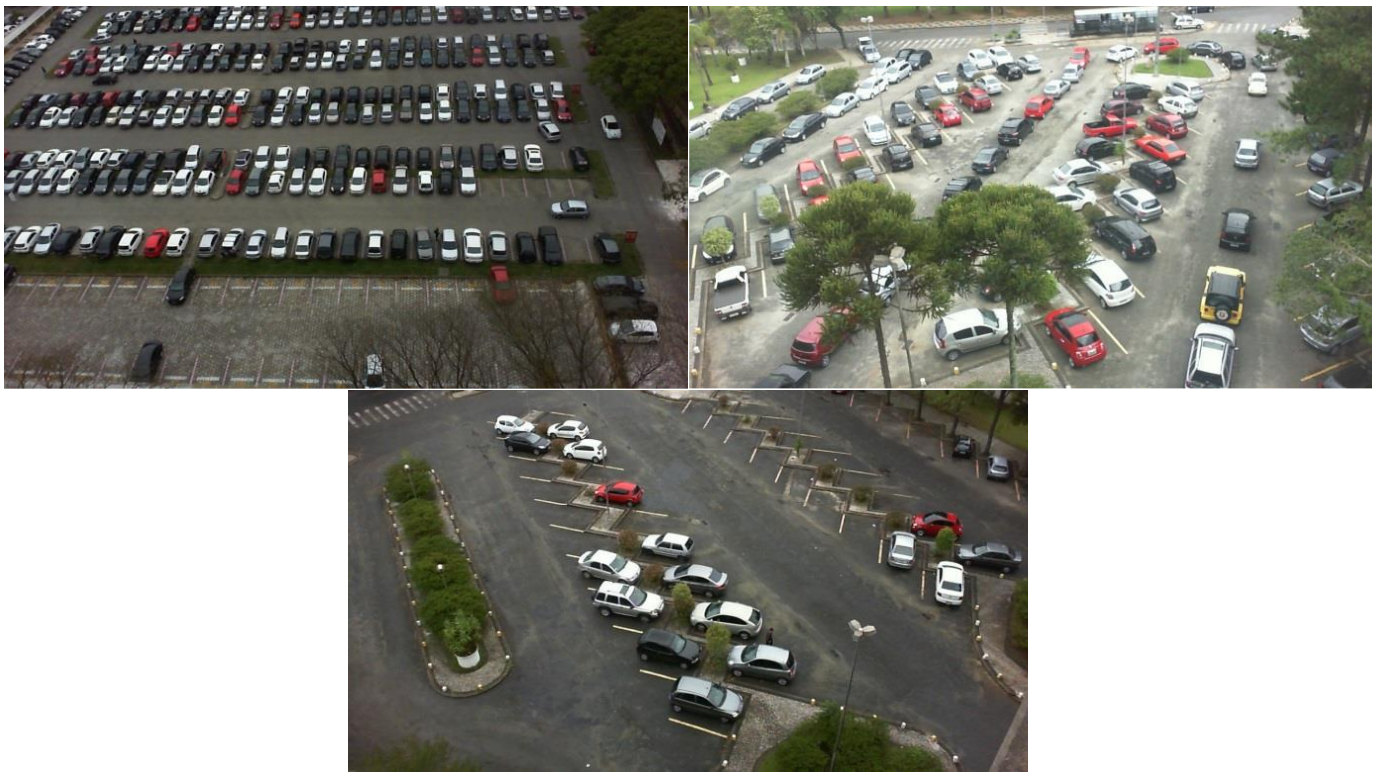


Fig 1.1 Examples of images from a Parking Lot CCTV

These parking lots are overlooked be surveillance cameras. The task is to leverage them to detect and count the empty parking spots.

The data set consists of images from three separate parking lots parking1a, parking1b and parking2. Each of the parking has data set sorted for three different scenarios cloudy, rainy and sunny. Each image in the parking2 has 100 annotated parking spaces (see Fig 1.2), along with the location, and occupancy information of the parking spot (Every image is supplemented with a ground truth file).

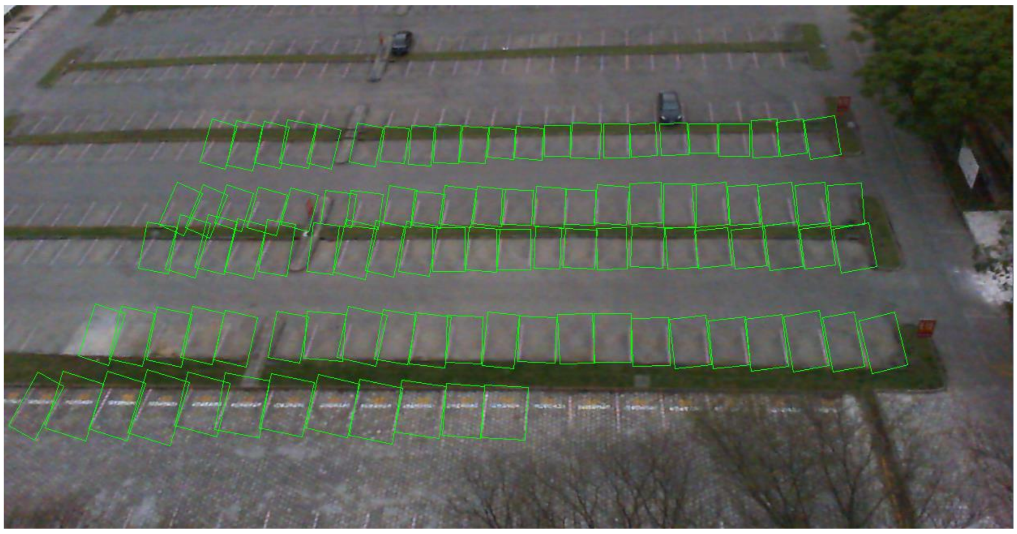
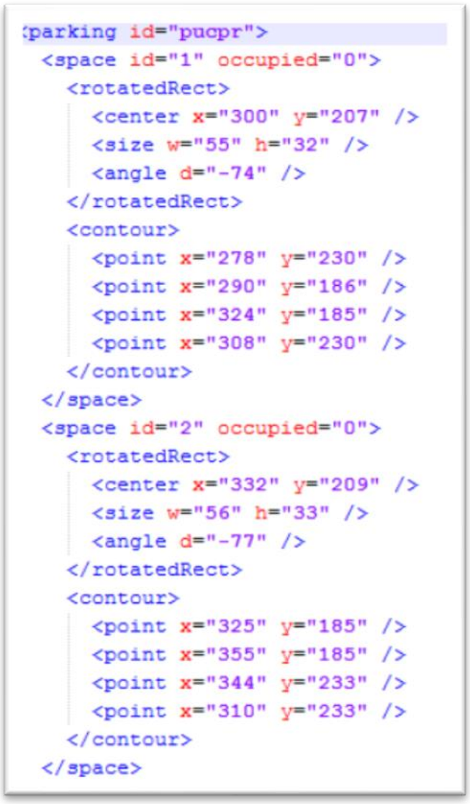


Fig 1.2 Example of a Ground Truth File.

Procedure:

Step 1: Creating a Training Set

- The program parse.py reads all the XML and corresponding JPG files in the "sunny" sub-directories of our main dataset "PKLot" to extract cropped car images and populate our positive images for training.
- To achieve this, the program uses the python library "glob". To parse the XML file, it uses xml.etree.ElementTree.
- For each XML file (an example of which is shown in Fig 3.1), it extracts the four "point" attributes under the child "contour" to get an estimate of where a car is present in the corresponding image.



```
<parking id="pucpr">
  <space id="1" occupied="0">
    <rotatedRect>
      <center x="300" y="207" />
      <size w="55" h="32" />
      <angle d="-74" />
    </rotatedRect>
    <contour>
      <point x="278" y="230" />
      <point x="290" y="186" />
      <point x="324" y="185" />
      <point x="308" y="230" />
    </contour>
  </space>
  <space id="2" occupied="0">
    <rotatedRect>
      <center x="332" y="209" />
      <size w="56" h="33" />
      <angle d="-77" />
    </rotatedRect>
    <contour>
      <point x="325" y="185" />
      <point x="355" y="185" />
      <point x="344" y="233" />
      <point x="310" y="233" />
    </contour>
  </space>
```

Fig. 1.3 Ground Truth XML File snippet

- We assume that (min(x), min(y)) and (max(x), max(y)) to be the opposite vertices of the bounding – box of the car. To ensure that the complete car is present inside the bounding – box, we expand the edges by 10 pixels on each side (as shown in Fig. 3.2).

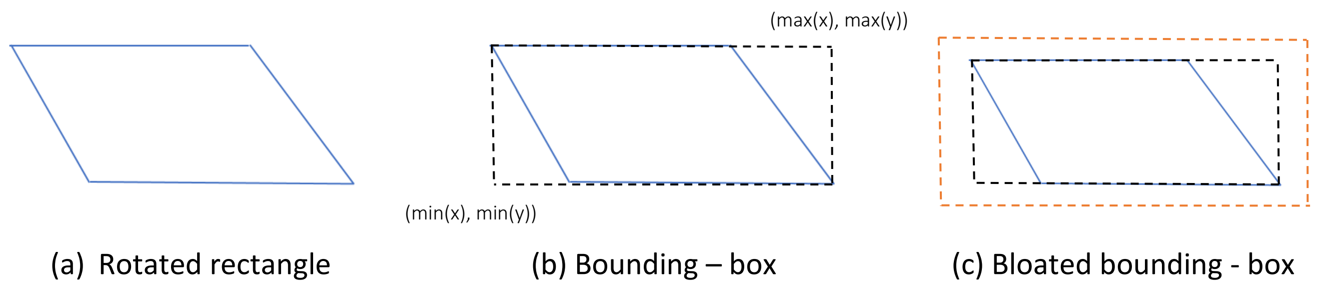


Fig 1.4 Bloated bounding – box logic

- Using the coordinates obtained from the bloated bounding box, each car is cropped from the image and resized to 50x50, properly indexed and saved into a directory "positive_samples".

Step 2: Training a Cascade Classifier

- The program parse.py creates a total of 184,501 positive samples. Fig 3.3 shows a few examples of the positive samples generated.

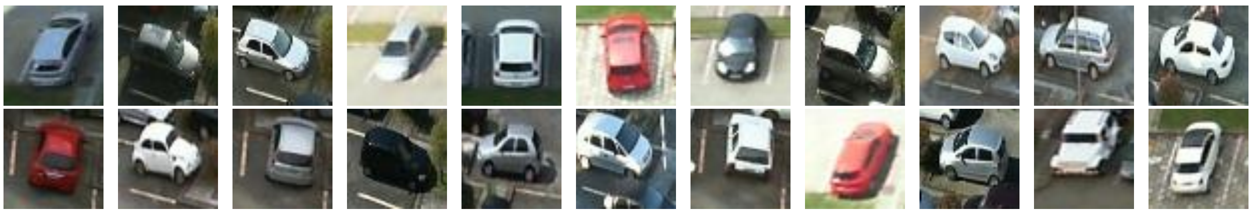


Fig. 1.5 Examples of positive images chosen for training

- Negative samples are collected manually and resized to 100x100. A negative image can be anything other than a car. So, I used the following image datasets available for public use from the internet:
  - The Street View House Numbers (SVHN) Dataset (http://ufldl.stanford.edu/housenumbers/)
  - CalTech 101 Dataset (http://www.vision.caltech.edu/Image_Datasets/Caltech101/) – excluding the car_side and motorbikes categories
  - Labeled faces in the Wild Dataset (http://vis-www.cs.umass.edu/lfw/)

  Fig 1.6 shows a few examples of images chosen as negatives.

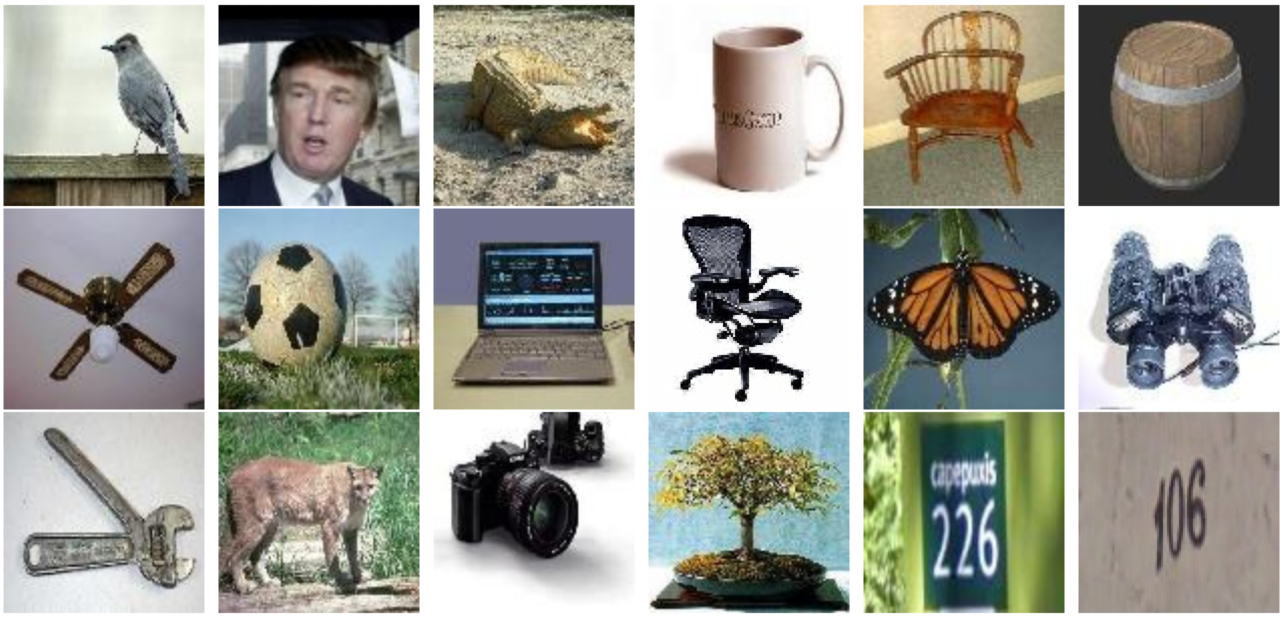


Fig. 1.6 Examples of negative images chosen for training

- Files required for training such as info.dat and bg.txt were created as per the instructions present in the OpenCV documentation for cascade classifier training. (https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html)
- **Training the LBP classifier –** An LBP classifier can be trained in a reasonably short time. I trained one with 50k positive samples and 25k negative samples for 15 cycles. It is a good practice in supervised learning to avoid using the entire dataset available for training the classifier. One possible reason for this is overfitting. When we use the entire dataset available (in this case all 184,501 positive images), the trained classifier might perform extremely well in the training dataset but will fare poorly on the test dataset.

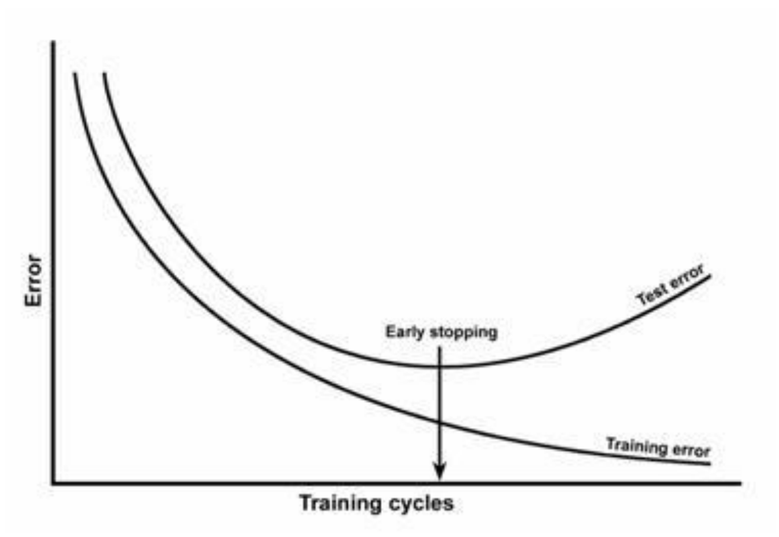


Fig. 1.7 Overfitting in Supervised Learning

- While creating the vector file using the opencv_createsamples command, we use 50k as the value for the argument -num.
- Since cascade training is a boosting process, the value for -numPos argument in the opencv_traincascade command must always be less than the number of samples in the created vector file. The vector file must contain >= (numPos + (numStages-1) * (1 - minHitRate) * numPos) + S number of samples where S is the number of samples in the vector file that can be recognized as background right away. numPos = 0.9 * number of samples in the vector file is an accurate estimation of this coefficient. Therefore, the value of numPos in my case is 45k.
- Minimum Hit Rate and Maximum False Alarm Rate were left at their default values. The number of possible features a classifier needs to consider and as an extension, the time required to train the classifier is directly proportional to the width and height of the window chosen. I chose a window size of 24x24.
- I chose numStages as 15, since the number of weak classifiers that were boosted above Maximum False Alarm Rate would be greater and the accuracy is directly proportional to this number.
- Hence, my final training command was:
  opencv_traincascade -data lbp50000 -vec lbp50000.vec -bg bg.txt -numPos 45000 -numNeg 25000 -numStages 15 -featureType LBP -w 24 -h 24
  The training took 4 hours 42 minutes and 40 seconds to complete. The performance of the classifier is present in the file lbp50000.txt in this repository.
- **Training the HAAR classifier** – All the points raised in the previous section hold true for HAAR classifier training as well. A HAAR cascade training takes a very long time. To account for time constraints, I trained a HAAR classifier with 10k positive samples, 5k negative samples for 10 stages.
- Hence the command used was:
  opencv_traincascade -data haar10000 -vec haar10000.vec -bg bg.txt -numPos 9000 -numNeg 5000 -numStages 10 -featureType HAAR -w 24 -h 24
  The training took 22 hours 06 minutes and 44 seconds to complete. The performance of the classifier is present in the file haar10000.txt in this repository.

Step 3: Car Detection

- The program “test.py” takes a classifier (LBP or HAAR) and an input test image and performs car detection.

- There are two important parameters in the detectMultiScale command that determine the accuracy of the classifier:
    - scaleFactor – OpenCV creates an image pyramid of our test image by shrinking it by the value given at each layer. If the scaleFactor is too large, then we'll only evaluate a few layers of the image pyramid, potentially missing cars at scales that fall in between the pyramid layers. On the other hand, if we set scaleFactor too low, then we will detect more objects in the test image, but it makes the detection process slower and substantially increases the number of false-positives
    - minNeighbors – This parameter controls the minimum number of detected bounding boxes in a given area for the region to be considered a car. It is very helpful in pruning false-positive detections.

  One disadvantage of cascade classifiers is that both these parameters need to be fine-tuned on an image-to-image basis to get the best result possible. "test.py" accepts these two parameters as command line arguments.
- Fig. 1.8 shows sample outputs for both HAAR and LBP classifiers.

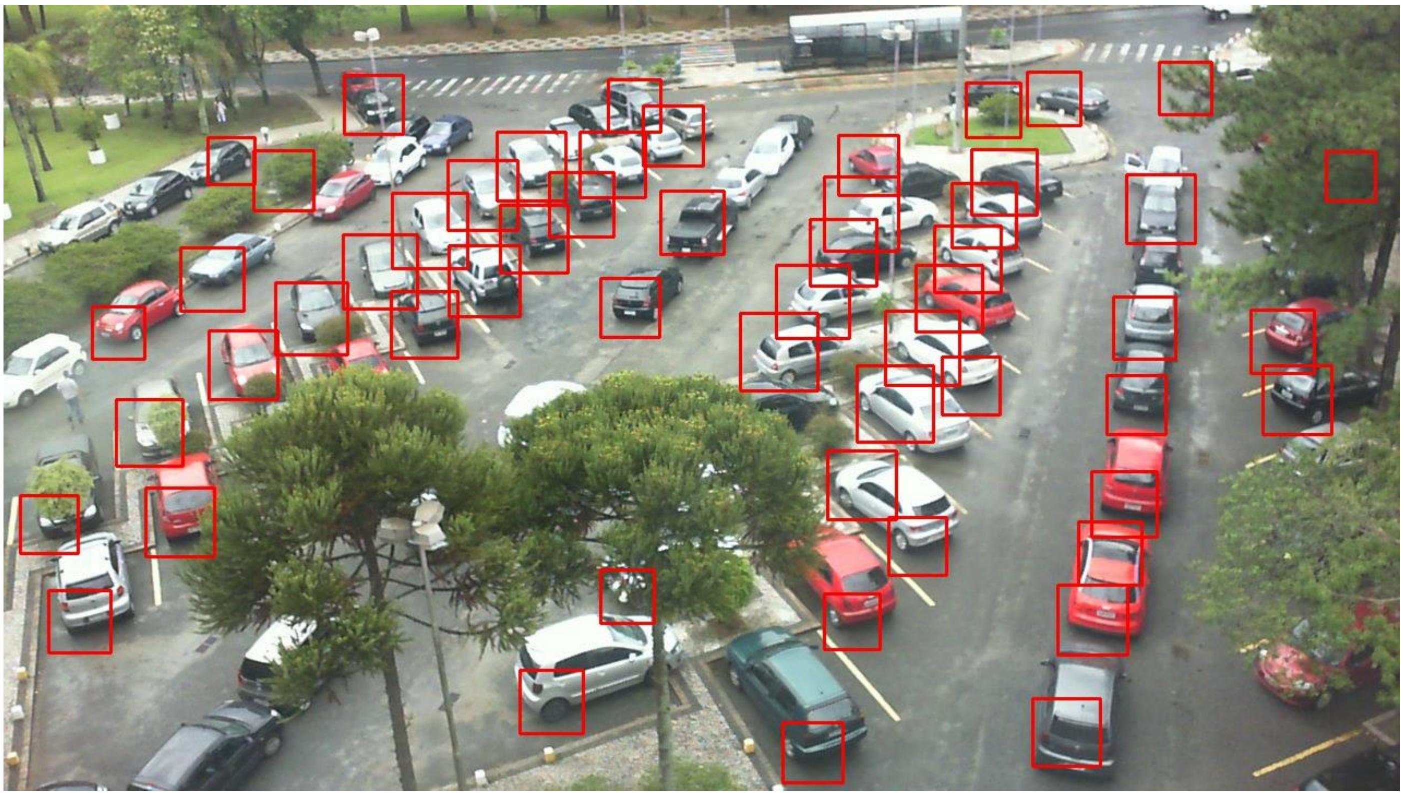


Fig. 1.8(a) LBP scaleFactor = 1.03 minNeighbors = 33

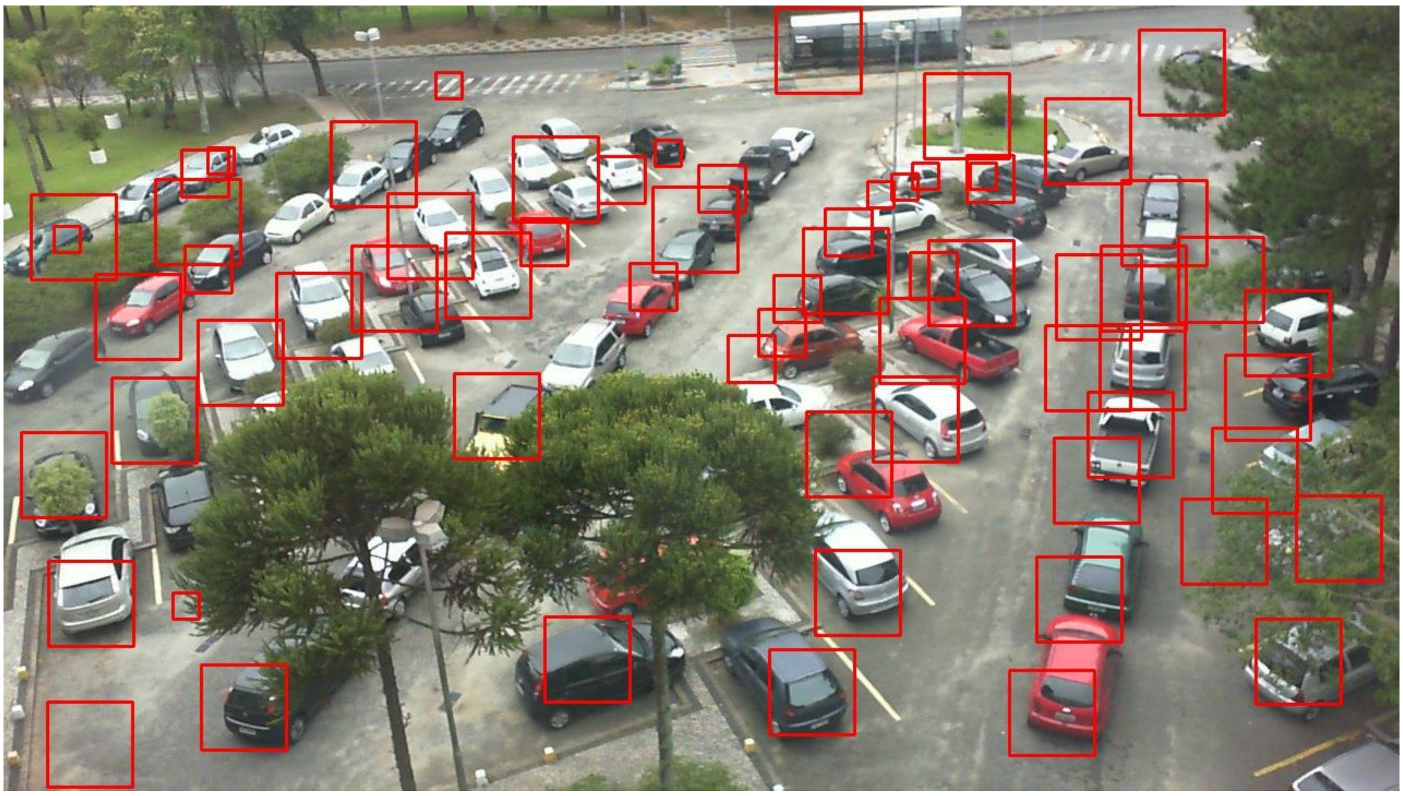


Fig. 1.8(b) HAAR scaleFactor = 1.8 minNeighbors = 10

- Implementation details for “test.py” can be found in “Readme.txt”

Step 4: Parking Lot Analysis

- Using the results from car detection, we determine whether a parking lot is empty or not.
- To do this, we check whether the midpoint of a parking lot’s bounding box lies within a rectangle detected by the classifier.
- Following the same logic as proposed in Step 2, we can compute the midpoint of the parking lot’s bounding box to be ((min(x) + max(x)) / 2, (min(y) + max(y)) / 2). Fig. 1.9 is a visual explanation of this logic.

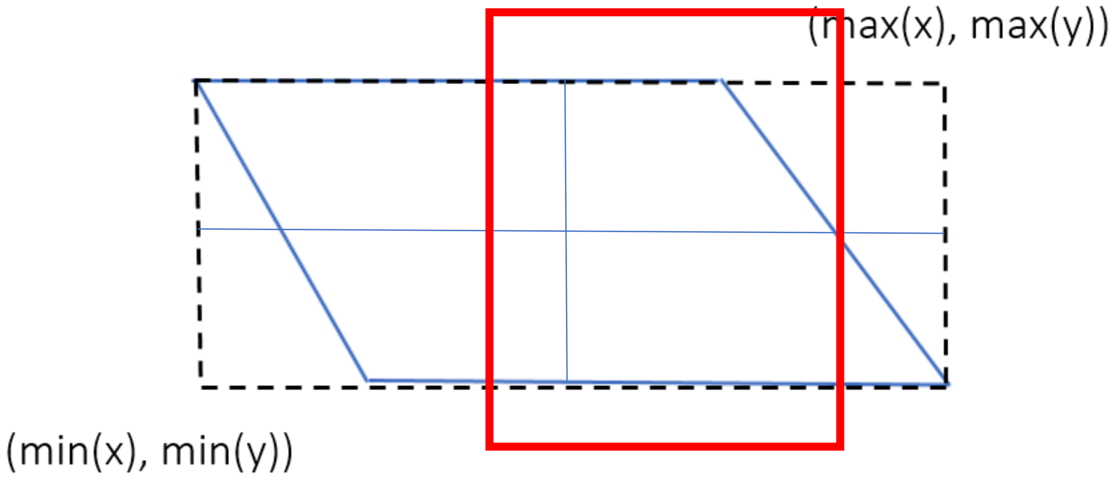


Fig. 1.9 When the mid-point of the parking lot’s bounding box lies within the detected rectangle, the classifier determines the parking lot to be occupied.

- Fig. 1.10 shows a confusion matrix which we use to compute the accuracy of our classifier.

		Obtained from Ground Truth XML	
		Car Present	**Car Absent**
Result from classifier	**Detected**	**True Positive**	**False Positive**
	Undetected	**False Negative**	**True Negative**

Fig. 1.10 Confusion Matrix for Car Detection

- All our calculations are only with respect to the annotated parking lots in the ground truth XML.
- Using the logic in Fig. 1.10 our classifier can calculate the number of true positives and false positives for the test image.
- By parsing the corresponding XML file, we can learn the actual number of parking spots that are occupied and empty.
- Using these values, we can easily compute the true negatives and false negatives by taking advantage of the confusion matrix given in Fig. 1.10. They turn out to be:
  # of True Negatives = # of empty parking spots - # of False Positives, and
  # of False Negatives = # of occupied parking spots - # of True Positives
- Accuracy of a classifier is defined as the number of correct classifications done by the classifier divided by its total number of classifications i.e.

  Accuracy = (TP + TN) / (TP + TN + FP + FN)

- The following table shows the performance of the classifier tested on 50 images collected from rainy and cloudy datasets.

Serial #	Test Image	Classifier Feature	Training Stages	# of Positives	# of Negatives	TP	FP	Accuracy (%)
1	2012-09-12_07_44_29.jpg	LBP	15	50,000	25,000	71	0	89
2	2012-09-12_10_16_27.jpg	HAAR	10	10,000	5,000	57	0	62
3	2012-09-16_08_17_59.jpg	HAAR	10	10,000	5,000	1	0	99

4	2012-09-21_10_25_21.jpg	LBP	15	50,000	25,000	83	0	93
5	2012-09-21_11_30_24.jpg	LBP	15	50,000	25,000	78	2	91
6	2012-09-21_12_20_26.jpg	HAAR	10	10,000	5,000	34	6	75
7	2012-09-21_14_00_30.jpg	HAAR	10	10,000	5,000	38	10	74
8	2012-09-28_09_21_06.jpg	LBP	15	50,000	25,000	76	0	81
9	2012-10-11_09_41_36.jpg	LBP	15	50,000	25,000	88	2	95
10	2012-10-11_10_31_38.jpg	HAAR	10	10,000	5,000	56	0	63
11	2012-10-16_07_31_46.jpg	HAAR	10	10,000	5,000	17	5	83
12	2012-10-16_10_18_32.jpg	LBP	15	50,000	25,000	85	1	92
13	2012-10-16_12_13_38.jpg	LBP	15	50,000	25,000	56	3	94
14	2012-10-16_14_18_44.jpg	HAAR	10	10,000	5,000	42	1	70
15	2012-12-11_16_01_08.jpg	HAAR	10	10,000	5,000	22	1	82.14
16	2012-12-11_16_16_08.jpg	LBP	15	50,000	25,000	22	0	78.57
17	2012-12-12_10_55_06.jpg	LBP	15	50,000	25,000	17	0	64.28
18	2012-12-12_11_45_07.jpg	HAAR	10	10,000	5,000	16	0	60.71
19	2012-12-14_08_20_03.jpg	HAAR	10	10,000	5,000	7	6	64.29
20	2012-12-14_08_25_03.jpg	LBP	15	50,000	25,000	14	6	64.29
21	2012-12-14_09_10_04.jpg	LBP	15	50,000	25,000	18	0	64.29
22	2012-12-14_09_55_05.jpg	HAAR	10	10,000	5,000	17	0	60.71
23	2012-12-17_10_35_06.jpg	HAAR	10	10,000	5,000	16	0	57.14
24	2012-12-17_12_15_08.jpg	LBP	15	50,000	25,000	18	2	71.43
25	2012-12-19_20_20_17.jpg	LBP	15	50,000	25,000	17	3	64.29
26	2012-12-21_18_20_14.jpg	HAAR	10	10,000	5,000	12	5	75
27	2012-12-21_18_30_15.jpg	HAAR	10	10,000	5,000	8	1	85.71
28	2013-01-22_12_45_08.jpg	LBP	15	50,000	25,000	17	1	78.57
29	2013-01-22_21_00_16.jpg	LBP	15	50,000	25,000	16	1	71.43
30	2013-02-22_17_10_11.jpg	HAAR	10	10,000	5,000	12	1	57.50
31	2013-02-22_18_55_13.jpg	HAAR	10	10,000	5,000	11	4	57.50
32	2013-02-26_13_04_33.jpg	LBP	15	50,000	25,000	25	0	62.50
33	2013-02-26_13_24_34.jpg	LBP	15	50,000	25,000	24	0	60
34	2013-03-05_11_40_06.jpg	HAAR	10	10,000	5,000	16	3	60
35	2013-03-05_12_55_07.jpg	HAAR	10	10,000	5,000	32	5	80
36	2013-03-05_14_20_09.jpg	LBP	15	50,000	25,000	29	0	72.50
37	2013-03-05_16_15_11.jpg	LBP	15	50,000	25,000	28	1	70
38	2013-03-09_12_25_07.jpg	HAAR	10	10,000	5,000	4	4	80
39	2013-03-09_17_50_13.jpg	HAAR	10	10,000	5,000	1	3	90
40	2013-03-12_07_15_01.jpg	LBP	15	50,000	25,000	9	0	82.50
41	2013-03-12_09_55_04.jpg	LBP	15	50,000	25,000	29	1	75
42	2013-03-13_13_05_08.jpg	HAAR	10	10,000	5,000	19	0	52.50
43	2013-03-13_17_00_12.jpg	HAAR	10	10,000	5,000	21	0	55
44	2013-03-13_18_25_14.jpg	LBP	15	50,000	25,000	27	3	72.50
45	2013-03-16_16_40_12.jpg	LBP	15	50,000	25,000	4	0	100
46	2013-03-19_07_30_01.jpg	HAAR	10	10,000	5,000	15	2	62.50
47	2013-03-19_12_45_07.jpg	HAAR	10	10,000	5,000	22	0	57.50
48	2013-03-20_17_00_12.jpg	LBP	15	50,000	25,000	24	2	75
49	2012-10-23_08_15_50.jpg	LBP	15	50,000	25,000	81	1	89
50	2012-10-23_08_20_50.jpg	LBP	15	50,000	25,000	84	3	90