UNIVERSITY, SCHOOL OR DEPARTMENT NAME

Project Vehicle Detection Additional Writeup

Lei Yang

January 11, 2018

1 Vehicle Detection

The purpose of this writeup is for answering the questions of reviewer and show what I had done based on the suggestions.
The code for most processing steps are contained in the IPython notebook vehicle_detection.ipynb file.

2 The answers to the reviewer's questions

2.1 Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.

The HOG feature is calculated directly using the skimage library. Since the hog function would require a number of argument. The choose of these arguments is critical. In my experiment, I set the parameters as following:

- I applied color transform from colorspace RGB to 'YUV'. This is selected by trial-and-error. I also tried using 'YCrCb' channel. Both of them achieved the better performance than other spaces. Fig 2.2 and 2.1 illustrated two windows detected by these two spaces.
- orient = 9. The document of skimage suggest typical values are between 6 and 12 bins.
- $pixel\_per\_cell$ = (8, 8). It specifies the cell size over which each gradient histogram is computed. This commonly chosen to be square.

- $cell\_per\_block$ = 2. This parameter specifies the local area over which the histogram counts in a given cell will be normalized.
- $hog\_channel$ = All.
- $spatial\_size$ = (32, 32)
- $hist\_bins$ = 32

Most of the above parameters are the default values. For these values I tried different combinations, such as pixel_per_cell as 10 or 12. There is no obvious improvement.
For the classifier, I used LinearSVC. The selection of classifier is a tradeoff between computation speed and performance.

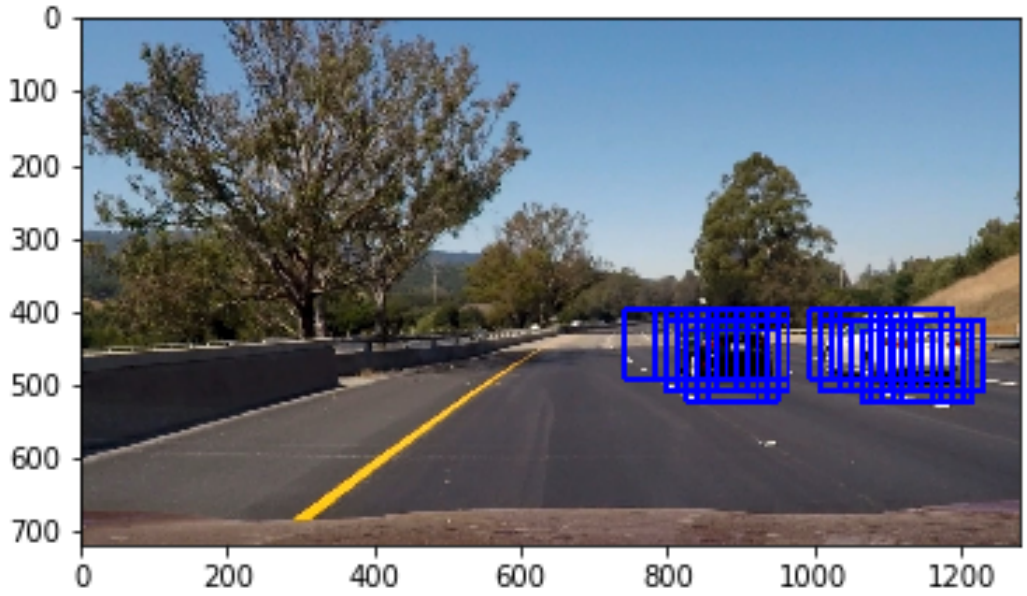

Figure 2.1: Sliding windows using YCrCb space

2.2 It seems that the "justification" part is missing from the writeup. Please, answer the following questions:.

- How many window sizes did you use? How did you choose the number of sizes? I choosed multiple scales of window sizes. In the begining, I used window scales as [1, 0.8, 0.6, 0.4, 0.2]. The reason of multiple scales is object generally has multiple scales

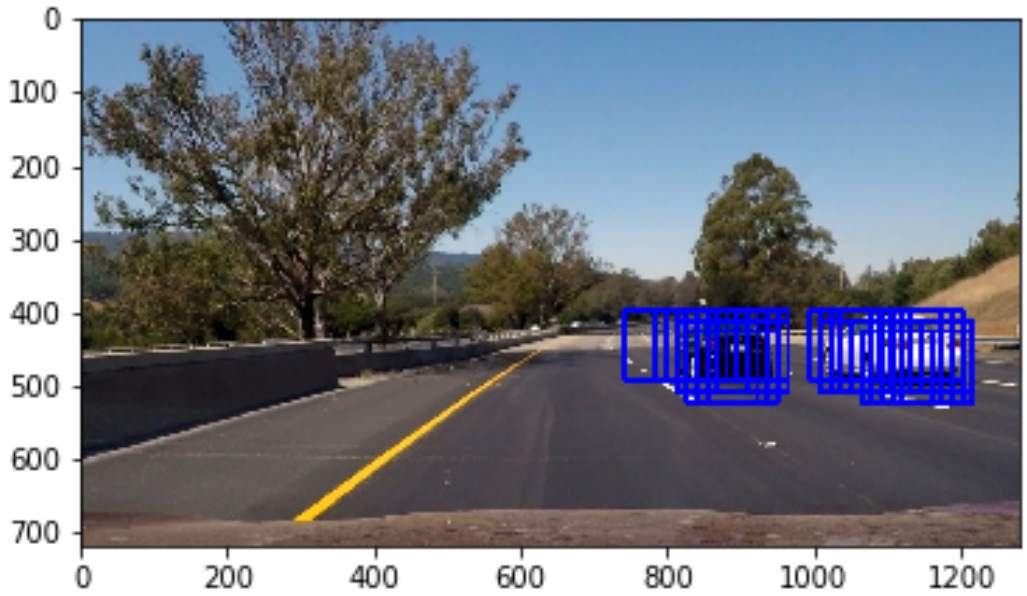

Figure 2.2: Sliding windows using YUV space

depends on the distance to the camera. It is a trade-off to choose the scale. With some researches about sliding windows, the scale variable may be limited. Since the x_start_stop also has variable scales. Therefore, I set the configuration values of different windows, these variables various depends on the distance between the car object and sensor.

- What were these sizes? Why did you choose those sizes you chose? Theoretically, the more the better. However, we have to make the testing time as small as possible. I finally choose the above scales based on multiple experiments.
- What overlap ration did you use (for which sizes)? How did you choose it/them? Overlap ratio I used 0.6 and 0.7 based on the distance in both x and y direction.
- How did you choose y-axis range(s)? I choose the y-axis as [400, 656] with longer distance and [400, 500] with shorter distance. Theoretically, there should be no difference if the detector is robust. To make the run-time more efficient, we set these to avoid detection the sky region.

There are frames when the white car is clearly visible and yet it is not detected (false negative detections). One reason for this can be that for certain window sizes the overlap ratio is not

big enough, try to increase the overlap ratio.

- I reset the value of both overlap_ratio in each direction as 0.6 and 0.7 rather than default value 0.5 to decrease the change of false positive.

2.3 PLEASE, PROVIDE MORE INFORMATION IN THE WRITEUP ABOUT THE INTUITION BEHIND, WHY THIS TECHNIQUE CAN REDUCE THE NUMBER OF FALSE POSITIVES.

- Heatmap is used to show which candidate has better probability to be classified as a car or not a car. Use the heatmap and a threshold, we can reject the candidate windows, since the predicted value will likely to be a small value.

2.4 TO GET EVEN BETTER CLASSIFICATION RESULTS YOU MIGHT WANT TO FINE TUNE THE C PARAMETER OF LINEARSVC CLASSIFIER WHICH CONTROLS BETWEEN THE TRAINING ACCURACY AND GENERALIZATION. AS THE TEST AND TRAIN IMAGES CAME FROM THE SAME SOURCE, THE HIGH TEST ACCURACY MIGHT IMPLY OVERFITTING. AGAINST IT YOU CAN USE C<1.0 VALUES. YOU CAN TRY 0.01 OR EVEN 0.0001.

- I tried different C (0.01, 0.001) values rather than the default 1. And there are some differences about the results. C=0.001 achieved the more smooth results. Some comparison can be shown as FIg 2.3, 2.4, 2.5, 2.6, 2.7, 2.8.

2.5 HOWEVER THERE ARE STILL FALSE POSITIVE DETECTIONS, AS WELL AS MANY FALSE NEGATIVE ONES IN THE VIDEO. TO OVERCOME THESE ISSUES YOU MIGHT CONSIDER APPLYING THE IDEAS GIVEN IN THE COMMENTS OF THE PREVIOUS AND NEXT RUBRIC POINTS.

- I applied the above items in the new ipynb file.

2.6 A METHOD, SUCH AS REQUIRING THAT A DETECTION BE FOUND AT OR NEAR THE SAME POSITION IN SEVERAL SUBSEQUENT FRAMES, (COULD BE A HEAT MAP SHOWING THE LOCATION OF REPEAT DETECTIONS) IS IMPLEMENTED AS A MEANS OF REJECTING FALSE POSITIVES, AND THIS DEMONSTRABLY REDUCES THE NUMBER OF FALSE POSITIVES.

Thanks for this helpful suggestion. I added the deque part in the last part of the code. And the video also generated with this method. I can see the big difference between the two results. The new video is saved as result_new.mp4.

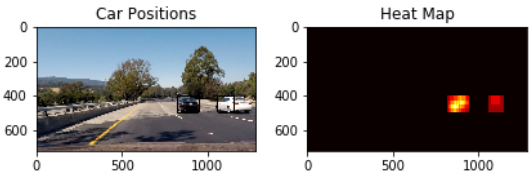

Figure 2.3: heatmap3 with c=0.001

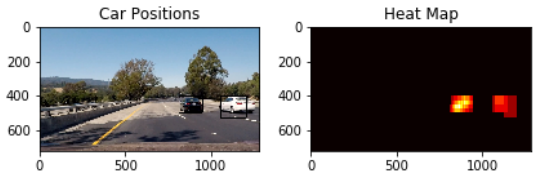

Figure 2.4: heatmap3 with c=0.01

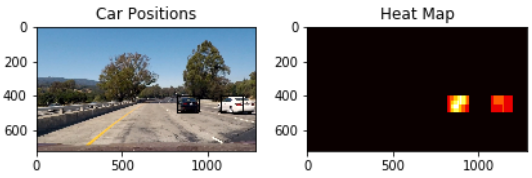

Figure 2.5: heatmap4 with c=0.001

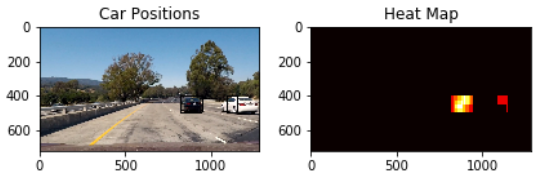

Figure 2.6: heatmap4 with c=0.01

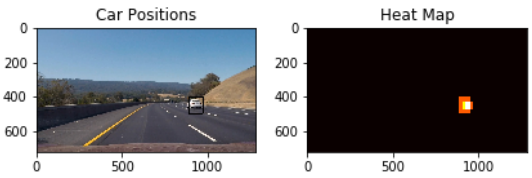

Figure 2.7: heatmap5 with c=0.001

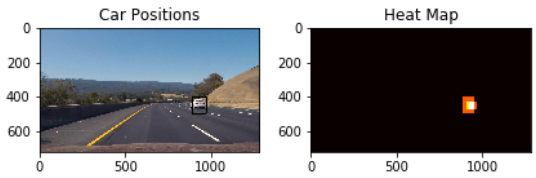

Figure 2.8: heatmap5 with c=0.01