UDACITY

PROJECT

Vehicle Detection and Tracking

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

You put a lot of effort into this project and I think it was worth it because you must have learnt a lot from it. Congratulations!

Keep up the good work for the next term. 😉

Writeup / README

The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

You provided a writeup with the necessary information in it. Well done.

Histogram of Oriented Gradients (HOG)

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.

You correctly used the function `skimage.feature.hog(...)` to get the HOG feautres. Also you tried out different color spaces and HOG parameters and found that the combination below works well for you. In Machine Learning the trial and error approach is an often used technique. Nice work.

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
```

The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.

You extracted spatial binning and color histogram features beside HOG features. You trained a Support Vector Classifier based on the extracted and normalized features and you achieved ~99% accuracy on the test set. Awesome.

Sliding Window Search

A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.

You implemented the sliding window approach here with three different sizes of widows (scales). For each size you carefully chose an appropriate overlap ratio and an y-axis range to narrow the search. Nice work.

However there are a longer section in the video, just before the black car passing the white one, when the white car is clearly visible and yet it is not detected (i.e. false negative detections). Two possible reason can be:

- The smallest size is not small enough to capture the white car when it is farther away. I suggest trying scale=1, too.
- The overlap ratio is too small. I suggest trying to use bigger overlap ratio (i.e. smaller `cells_per_step`)

Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)

To eliminate false positives you created a heatmap from the pixels of classified images. The intuition behind is that those pixels belong to cars where there is many positive classification which means hot pixels in the heatmap. Well done.

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.

Also you can threshold the result of the method `decision_function(...)` of `LinearSVC` to avoid false positives.

Video Implementation

The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

The sliding window search with SVC classifier has been used correctly to identify the vehicles. Great job.

However there are false negative detections in the video: there is a longer section when the white car is not detected at all, though it is not covered yet and clearly visible (in a real life application loosing the track of a vehicle for more than a second would be very dangerous). To overcome this issue you might consider applying the ideas given in the comments of the previous two rubric points.

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. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

If you found a car in the previous frame, you search for cars in the proximity of that find. Great idea. Moreover you use smooth the detections by summing older heatmaps. Nice work.

To store previous heatmaps I suggest using `collections.deque`, in this way you do not need to delete the oldest heatmap:

```
from collections import deque
history = deque(maxlen = 8)
...
history.append(current_heat_map) #No deletion or pop needed
...
```

Discussion

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

You provided a reflection on your work. Well done.

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ Reviewer Agreement