Vehicle Detection and Tracking

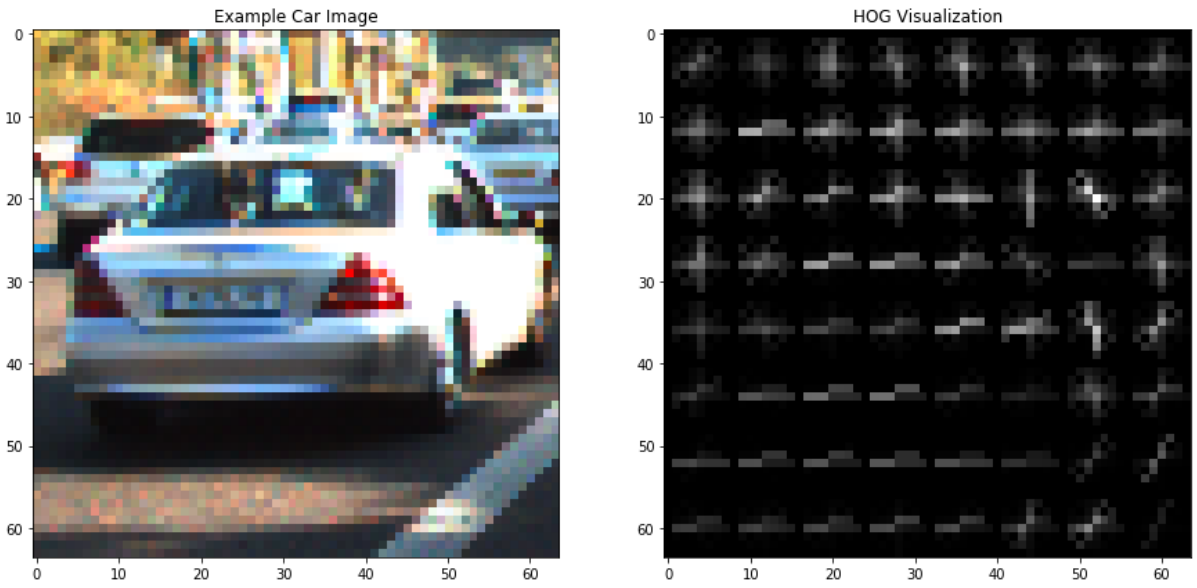
The goals / steps of this project are the following:

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
2. Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
3. Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
6. Estimate a bounding box for vehicles detected.

Histogram of Oriented Gradients (HOG)

1.Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

Function **get_hog_features** could extract HOG features from images. The main function is **hog** imported from skimage.feature.
Inspired by Udacity lesson “HOG Classify” program example, the parameters are set as:
orient = 10 (instead of 9, to get more gradient direction)
pix_per_cell = 8
cell_per_block = 2



2.Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

Function **color_hist** extract color histogram features in YCrCb color space and function **bin_spatial** computes spatial binning of color features.
In “T**rain and Test the Classifier**” part, HOG features, color histogram features from car images and not-car images separately have been extracted to train a linear Support Vector Machine(SVM) classifier. Classifier test accuracy is 98.90%.

3.Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

In part “**Test on test_images**”, function **slide_window** computes all the possible window locations in the image with one size, function **search_windows** find the windows matching classifier’s classification as “cars”.
Since the main vehicles need to be considered locate nearby, big size of 128*128 pixels window has been applied. There is a high chance of overlapping, so 0.7 is the overlap parameters here.

4.Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?

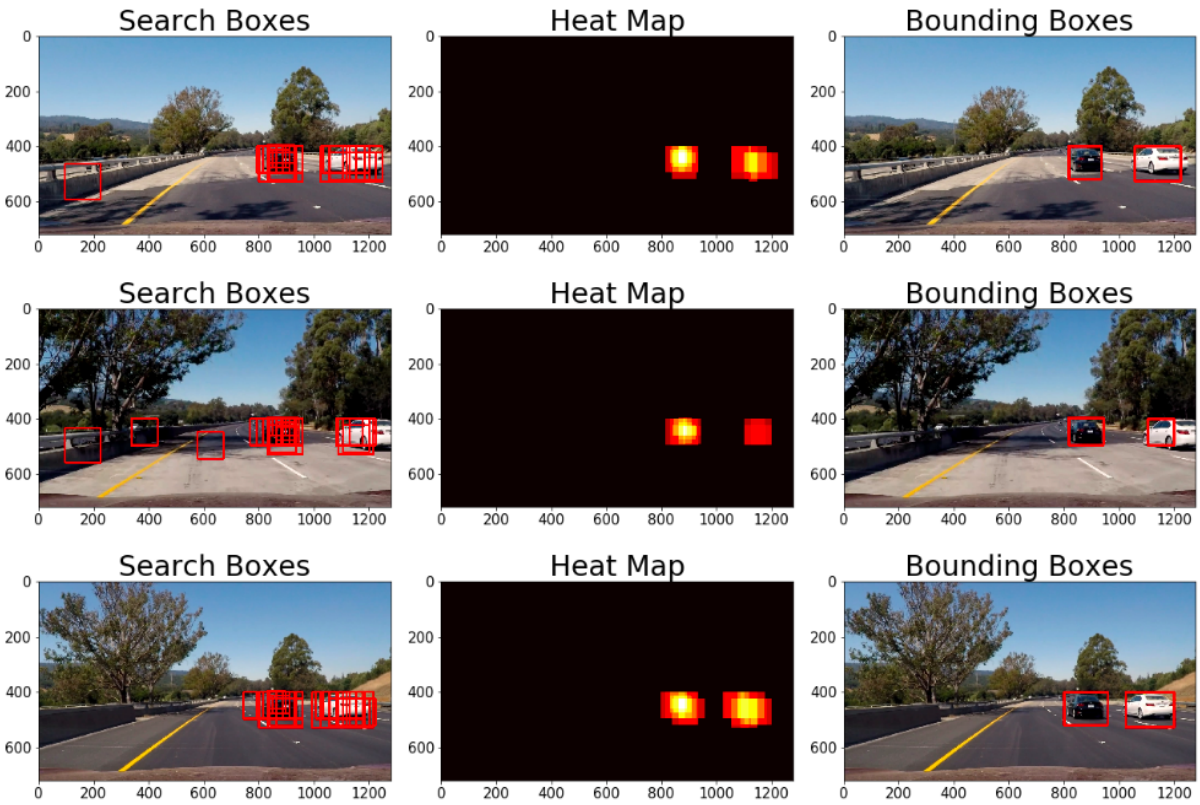
Two vehicles have been marked on above images during the test. Even though there are issues like multiple windows may detect the very same target.
Inspired by ideas online and Udacity's lesson, parameters to optimize classifier performances are:
color_space = 'YCrCb'
orient = 10
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimensions
hist_bins = 64 # Number of histogram bins

5. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Please check the attached "project_output.mp4" video.

6.Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

Inspired by Udacity's "Multiple Detections & False Positives", "Heat map" method have been applied.
For each vehicle detected window, "heat" flag value plus one from zero. Only windows meet thresholded heat values could be marked by function **draw_labeled_bboxes**.
This is the filter for false positives and combining overlapping bounding boxes.



7.Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

The biggest problem for me in this project is Hog Sub-sampling window search.
The **detect_vehicle** pipeline could detect vehicles in front and not far away.
But this pipeline is likely to fail to detect a vehicle when that vehicle is approaching from the left or right side instead of front to our controlled vehicle. I believe a wide-angle camera and more training data including rare conditions could help with this issue.