Vehicle Detection Project

Image References

[image1]: ./output_images/Example_Carimage.png

[image2]: ./output_images/Example_Noncarimage.png

[image3]: ./output_images/Test_example_box.png

[image4]: ./output_images/Test_example_heatmap.png

[image5]: ./output_images/Test1_labeledbox.png

[image6]: ./output_images/Test2_labeledbox.png

[image7]: ./output_images/Test3_labeledbox.png

[image8]: ./output_images/Test4_labeledbox.png

[image9]: ./output_images/Test5_labeledbox.png

[image10]: ./output_images/Test6_labeledbox.png

[video1]: ./project_video_output_Final.mp4

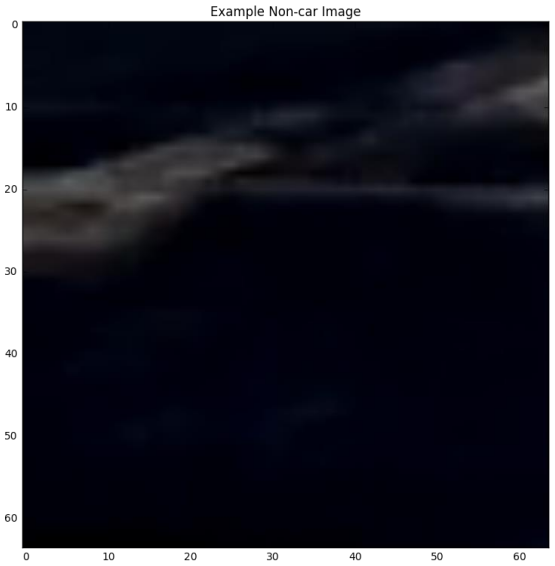
###Histogram of Oriented Gradients (HOG)

####1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the “Section#2.Feature Extraction” of the IPython notebook. Before extracting features, I prepared data(Section#1.Data Preparation). I tried to use extra (Udacity) data in addition to GTI, KITTI data. I opened the CVS file and cropped the vehicle image based on the location of the vehicle in the CVS file. When it comes to non-vehicle image, I randomly cropped the image which region does not have the vehicle (0~400pixel in y-axis). I acquired total about 80,000 vehicle data and 80,000 non-vehicle data. However, I could not include this data in this project because it causes memory error.

I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` images:

[Image#1, #2]



I then explored different color spaces (RGB, HSV, LUV, HLS, YUV, YCrCb) and different parameters (Orientations, Pixels_per_cell, Cells_per_block and Hog_Channel).

####2. Explain how you settled on your final choice of HOG parameters.

On top of that, I also tried to use binary spatial features, color histogram features in addition to HOG features. Then, I tried various combinations of parameters. In terms of parameter combination, I will explain it below section in detail.

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

I trained data using linear SVM model. Linear SVM was the simplest and fastest way to train data and provided high accuracy as well. I trained data under a variety of parameter combination. In addition, I also checked whether binary spatial features and color histogram features would help to increase the accuracy. Below table is my parameter combination and accuracy result:

Color	Spatial	Hist_bin	Orient	Pix	Cell	Channel	Spatial	Hist	HOG	Accracy
RGB	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	98.31
HSV	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	99.27
LUV	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	99.21
HLS	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	99.18
YUV	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	99.18
YCrCb	32	32	9	8	2	ALL	TRUE	TRUE	TRUE	99.55
YCrCb	32	32	9	8	2	ALL	FALSE	TRUE	TRUE	99.1
YCrCb	32	32	9	8	2	ALL	TRUE	FALSE	TRUE	99.18
YCrCb	32	32	9	8	2	ALL	FALSE	FALSE	TRUE	99.01
YCrCb	32	32	9	8	2	0	TRUE	TRUE	TRUE	98.11
YCrCb	32	32	9	8	2	2	TRUE	TRUE	TRUE	97.07
YCrCb	32	32	9	12	2	ALL	TRUE	TRUE	TRUE	99.27
YCrCb	32	32	12	12	2	ALL	TRUE	TRUE	TRUE	99.27
YCrCb	16	16	9	8	2	ALL	TRUE	TRUE	TRUE	99.18

My conclusion is 1) For the color channel, YCrCb provides the best accuracy. 2) Spatial and Color features is helpful to increase the accuracy of the training model 3) ALL Hog channel should be used for the accuracy. Conclusion#2, #3 is relatively obvious because additional information might be helpful in general case. For the conclusion #1, I compared HOG images in different color channel to see why YCrCb provides the best result. However, I could not find much difference on the image. My final model shows accuracy of 99.55 which is enough to use.

###Sliding Window Search

####1. 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?

####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Sliding window search is implemented at “Section#4. Sliding window search”. Basically, I used and modified “find_car” function which is introduced in the class. Because it calls HOG features only one time, I thought it is much efficient way. I used 8 pixels per cell, 2 cells per block and 2 cells per step which means 75% of window overlap. I tried to use 1 cell per step (87% of window overlap), but it cause too much false positive. In terms of the window size & location, I used 3 different size of windows with following manner. 1) X_axis: I set only right half a region of interest because my car is on the first lane in project image & Video 2) Y_axis : (From 400~500 pixel) I used 1.0 scale (64 x64) of window. (From 400~550 pixel) I used 1.5 scale (96 x 96) of window. (From 500~700 pixel) I used 2.0 scale (128 x 128) of window. This is the example image of the window search:

[Image#3]

In order to merge overlapping boxes and to reduce the false positive (Not in above image, but a lot in video stream), I used heatmap with threshold and labeled box strategy which are introduced in the class. I set threshold value 2 for these test images, but I used different approach to video stream (I will explain this next section in detail). Below images are heatmap example for 1 test image and labeled box examples for 6 test images

[Image#4, #5, #6, #7, #8, #9, #10]

Video Implementation

####1. 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.)

I uploaded my video file to Github with other materials. I also added this project video to "advanced lane detection video (Project#4)". My pipeline for video is in "Section#5. Pipeline for image & video"

####2. 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.

At first, I applied same pipeline for image test to video. With accuracy of 99.5%, there was too much false positive and unstable vehicle detection box!!!! So, as I did at project#4, I implemented a sort of smoothing strategy. I define the Smooth() class which save vehicle detected boxes of n-number (I used 15 number in this project) of previous frame. Then, I hand over these boxes to the input of heatmap. Then I applied high threshold to remove the false positives. If we used this strategy, 1) We can eliminate them easily because false positive does not appear consistently. 2) We can get smoother boxes for vehicle detection box. In my final result, I used threshold of 36 which means "if pixels of each box are not overlapped over 36 times on the 15 frame images, the pixels should be removed". Even though my video still has to be improved because of a couple of false positive and unstable vehicle detection box, this was the optimal value for me. Threshold value is always trade-off. If I set it too high, it misses the vehicle occasionally in spite of minimum false positive. Otherwise, if I set it too low, it causes too many false positive.

###Discussion

####1. 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?

1) It is worth to try different training model such as CNN, Decision tree beside of SVM. Even though it shows great accuracy, there were a lot of false positive in the video stream. Another model would be a solution for this.

2) I felt that using previous frame information is required for all the image processing. 0.5% of fail is not small number even though 99.5% accuracy looks great. If we assume to use about 50,000 sliding window, 250 times would be misclassification. To make it robust, we have to think about more how to use each frame efficiently.