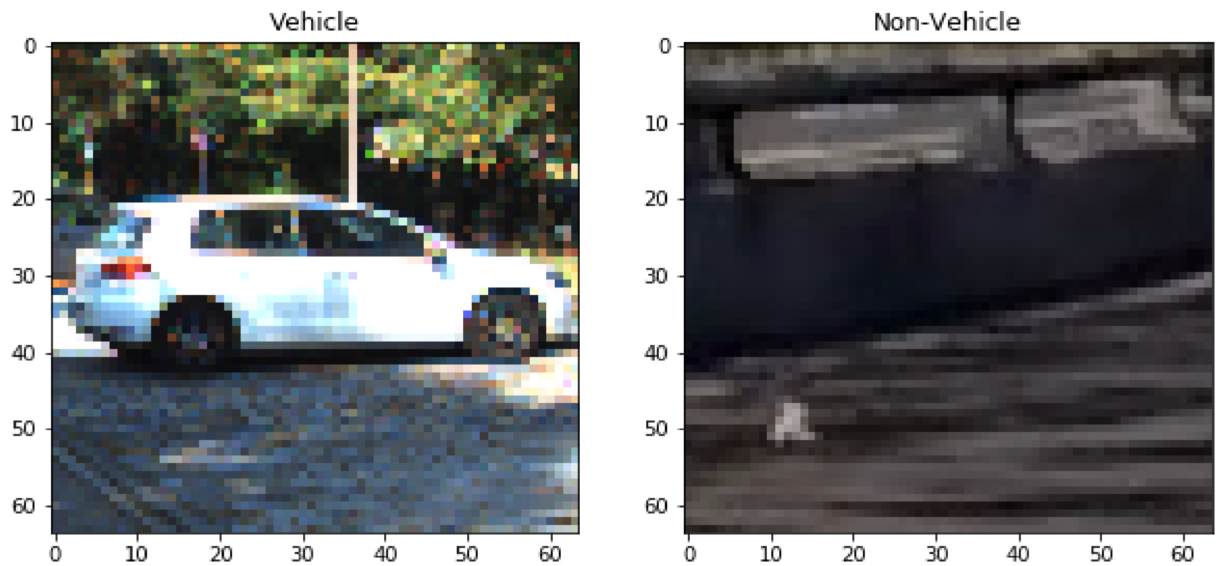
Vehicle Detection Project

The goals / steps of this project are the following:

Reading dataset:

The code for this step is contained in the cells(1-3) of the IPython notebook

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` classes:



Extracting Features

The code for this step is contained in the cells(4-6) of the IPython notebook

To identify vehicles in an image, we need "signature." for it. To create the vehicle signature, I extracted three different feature-sets:

- Spatial Features
- Color Histogram Features
- HOG (Histogram of Oriented Gradients) Features

Spatial Features :

Spatial features are the image pixel values after resizing and flattening the image.

Color Histogram Features :

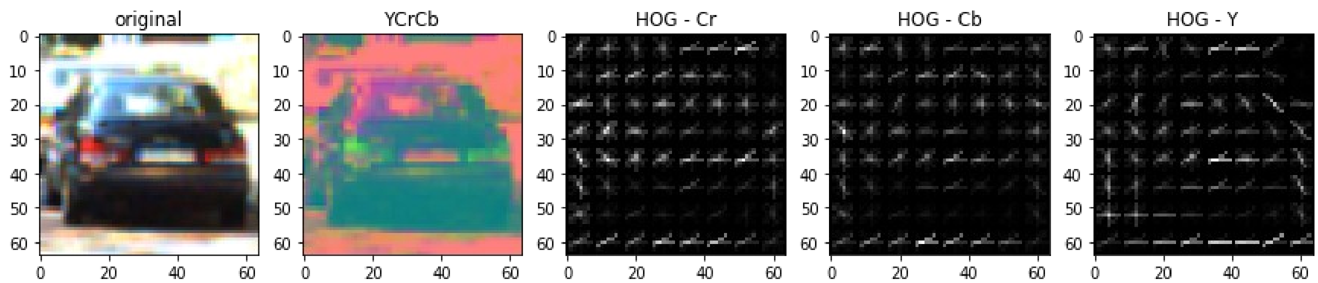
A color histogram totals the number of pixel values that fall into evenly-distributed bins for each color channel. I chose to use 32 color bins parameter .

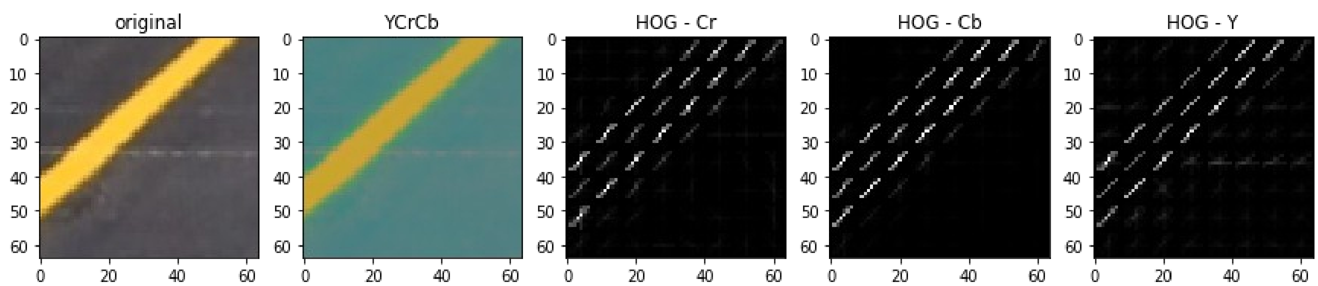
HOG (Histogram of Oriented Gradients) Features

I extracted HOG features from the training images with Udacity course function using the following parameters:

- orient_param=8
- pix_per_cell_param=8
- cells_per_block_param=2

And this was the result for a vehicle and non-vehicle example.





Training a Classifier

The code for this step is contained in the cells(8-9) of the IPython notebook

I trained a linear SVM using the provided data set of vehicle – non-vehicle images. For each image we wish to train our machine learning classifier on, these features will be extracted and concatenated together to form the image's vehicle "signature." I trained a Linear SVM classifier on 8792 images . The features were normalized before being passed in to the classifier by StandardScaler. The test results showed over 99% accuracy.

Feature extraction parameters

This step is in the cells(7,10) of the IPython notebook

I tried various combinations of parameters

i	orient	Pix /cell	Cell/ block	Color Space	Spatial Size	Hist Bins	Hog Channel	spatial	hist	hog
1	8	8	2	RGB	(16, 16)	32	ALL	False	True	True
2	8	8	2	RGB	(16, 16)	32	ALL	False	False	True
3	8	8	2	RGB	(16, 16)	32	ALL	False	True	False
4	8	8	2	RGB	(16, 16)	32	ALL	True	False	False
5	8	8	2	YCrCb	(16, 16)	32	ALL	True	False	True
6	9	8	2	YCrCb	(32, 32)	32	ALL	True	True	True
7	8	8	2	HSV	(16, 16)	32	ALL	True	False	True
8	8	8	2	RGB	(16, 16)	32	ALL	True	False	True
9	8	8	2	RGB	(16, 16)	32	0	True	False	True
10	8	8	2	RGB	(16, 16)	32	1	True	False	True
11	8	8	2	RGB	(16, 16)	32	2	True	False	True

i	Feature vector length	Test Accuracy
1	4800	0.964
2	4704	0.9623
3	96	0.505
4	768	0.9279
5	5472	0.9885
6	8460	0.9918
7	8792	0.989
8	5472	0.9772
9	2336	0.9645
10	2336	0.9744
11	2336	0.9696

And found that the best set of parameters are:

i	orient	Pix /cell	Cell/ block	Color Space	Spatial Size	Hist Bins	Hog Channel	spatial	hist	hog
6	9	8	2	YCrCb	(32, 32)	32	ALL	True	True	True

i	Feature vector length	Test Accuracy
6	8460	0.9918

The main parameters that highly make effects are YcrCb Color Space , using all feature extraction methods.

Sliding Window Search

The code for this step is contained in the cells(11-12) of the IPython notebook

I created windows in our area of interest (ystart = 350, ystop = 656)
I used Udacity function that combines HOG feature extraction with sliding windows.

This function extract HOG features for the entire image and then these features are sub-sampled according to which window we are in and then fed to the classifier.
I did some changes of this function to output both images and windows that I use later.
I used different window size at first to do detect both near and far cars.
But I realized that heatmap that I use later ensures that near vehicles is windowed well.
Ultimately I searched using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.
Here are some example images:

Improving Sliding Window Search

The code for this step is contained in the cells(13) of the IPython notebook

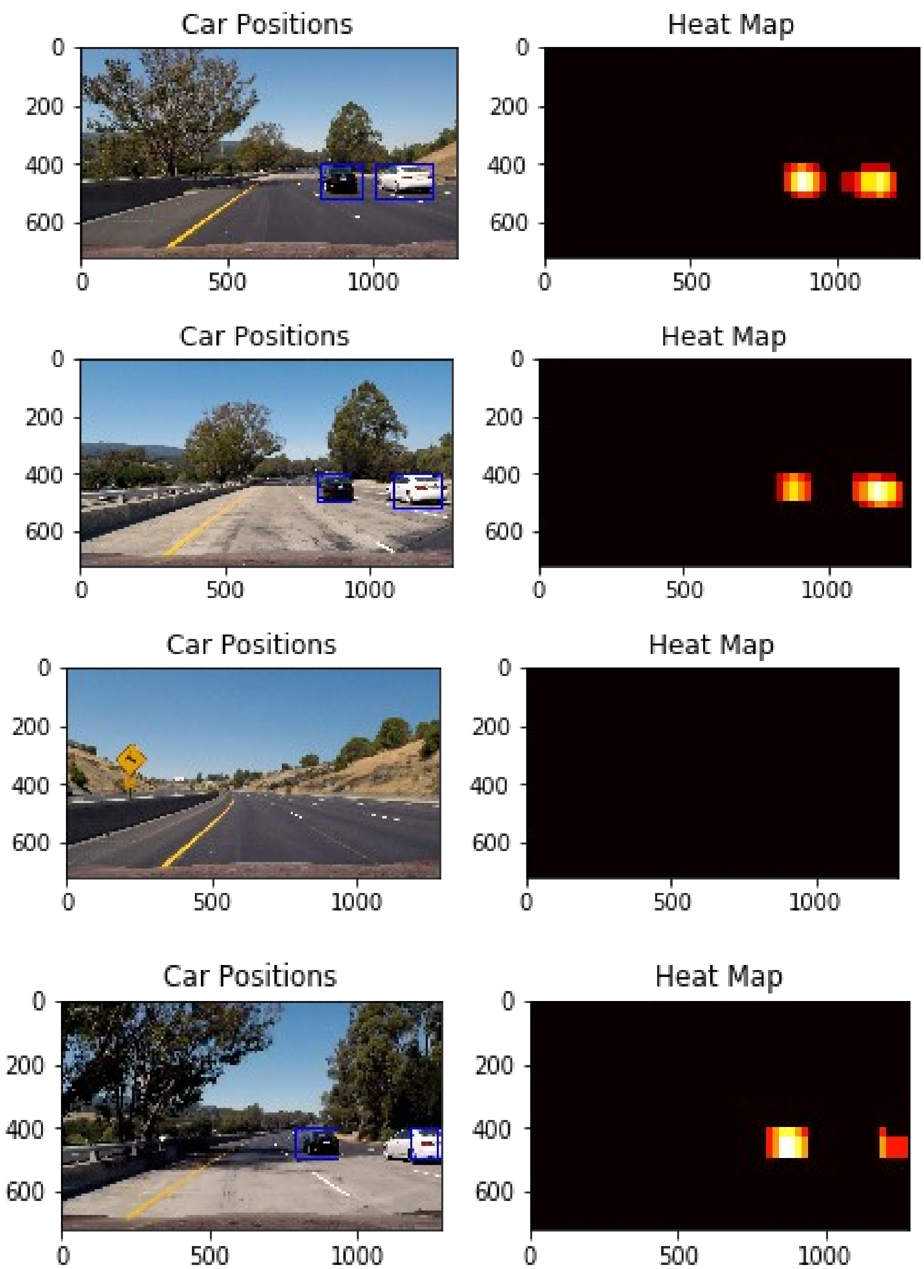
Since there are overlapping and probably false positive in the images. We should overcome that .

I overcame that using these techniques:

- Combined heatmap and thresholding:

  Since true positive window number is large and false positive is so low, The function adds 1 to any pixel in a window so that it most probably lie above another window in case of true positive so there are at least two windows in the same place. But , in case of false positive , For sure there aren't any other window so there are less than 2 windows in the same place. If we took 2 as a threshold for no. of windows to consider a vehicle , it works.

Here are some test images after using heatmap:



Video Implementation

The code for this step is contained in the cells(13) of the IPython notebook
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Discussion

There are still few false positives that may make the car apply brakes or change its road.
One improvement that can be done is to increase training data (flip images and different sizes) so that the classifier do better .
Also we can restrict the area of interest not to take false positives from the other road .
Do more feature extraction using combination of more techniques.
Finally we can fuse camera output with another range finder sensors to double check on any detected element.