Vehicle Detection Project

Please note that to run my code you will have to download the vehicles and non-vehicles dataset because I am unable to upload that in my project directory. When I try to upload that it gives me error that number of files are large.

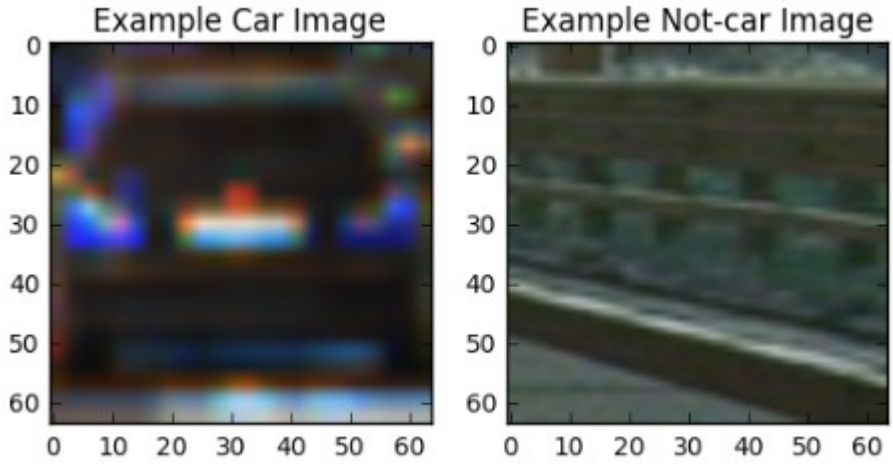
The goals / steps of this project are the following:

- •Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- •Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- •Run the 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.
- •Estimate a bounding box for vehicles detected.

Rubric Points

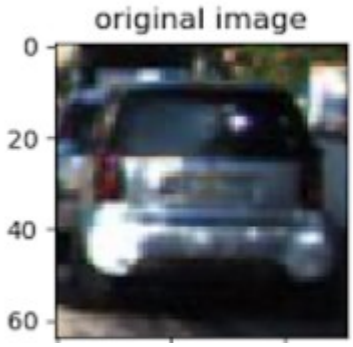
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

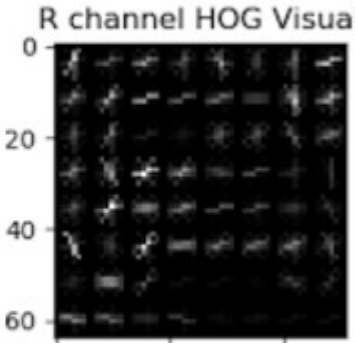
The code for this step is contained in the first code cell of the IPython notebook example.ipynb. This notebook has been used foe the entire code.
I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:

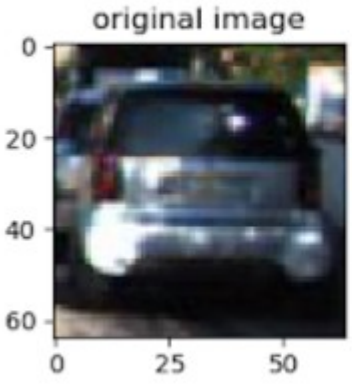


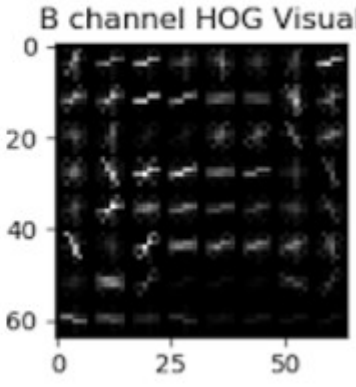
I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.

Here is an example using the RGB color space and HOG parameters of orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2). The RGB gave almost same features for each of the three channels, so I tried other color schemes.

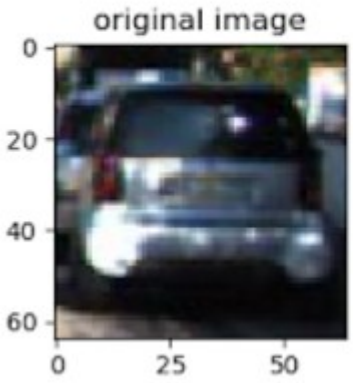


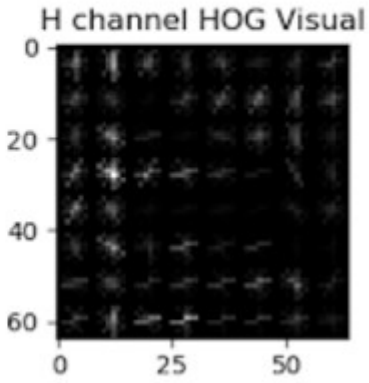


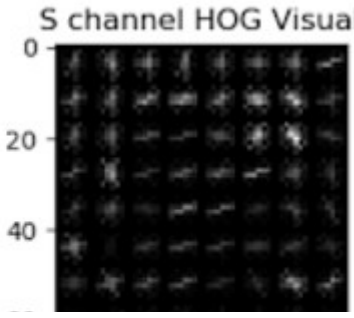


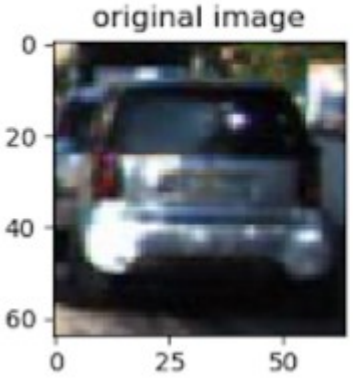


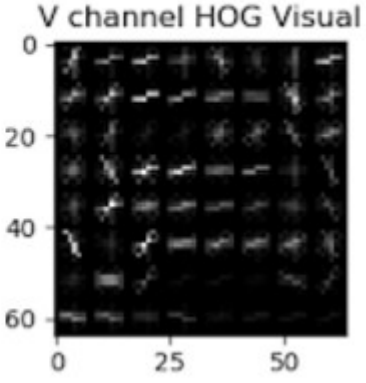
Here is an example using the HSV color space and HOG parameters of orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):











Like wise I plotted hog feature images for different color spaces, and it was observed that concatenating HOG features for each channel of the HSV colorspace gave sufficiently good results. YcrCb also gave equally good results. But I settled for HSV since the results were equally good.

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

I trained a linear SVM using HOG sub-sampling Window search by that concatenating HOG features for each channel of the HSV colorspace. This is done in cell 2 (In [2]).

The training and testing examples were taken from the vehicles and non-vehicles repository. These datasets are comprised of images taken from the **GTI vehicle image database**, the **KITTI vision benchmark suite**, and examples extracted from the project video itself.

I had split the training and testing example in 80:20 ratio.

The accuracy on the test set turned out to be 98.17.

The following was my result after training:

```
58.64 Seconds to extract HOG features...

(14208, 5292)
(3552, 5292)
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 5292
18.89 Seconds to train SVC...
Test Accuracy of SVC =  0.9817
My SVC predicts:  [ 1.  1.  1.  0.  1.  1.  1.  1.  0.  1.]
For these 10 labels:  [ 1.  1.  1.  0.  1.  1.  1.  1.  0.  1.]
0.00115 Seconds to predict 10 labels with SVC
```

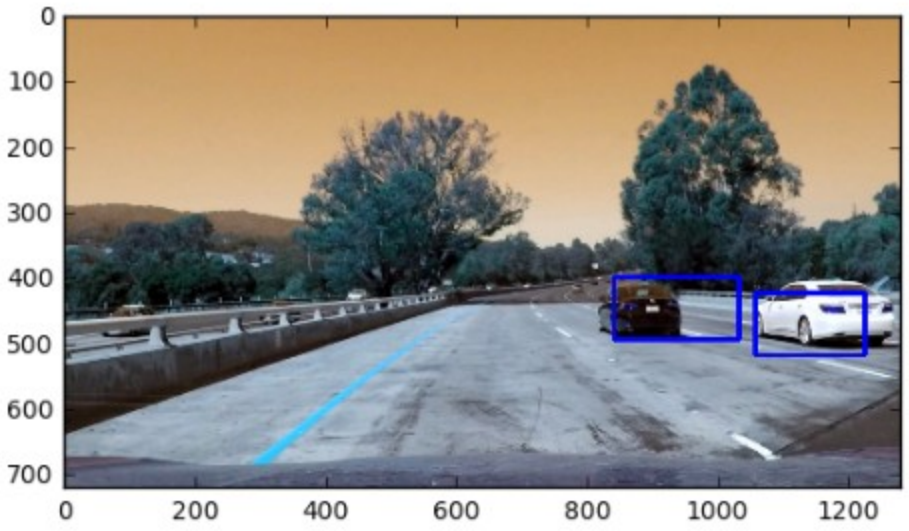
3. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

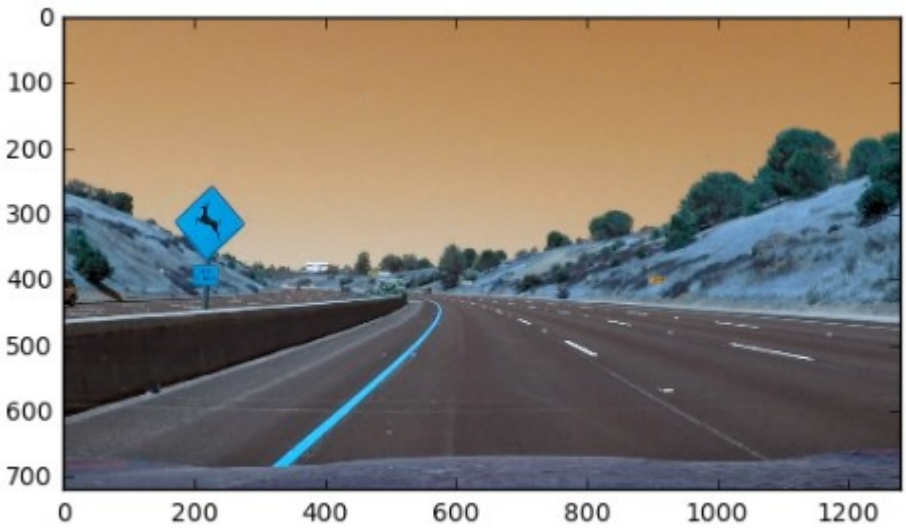
This has been implemented in cell 3 (In [11]). I used Hog Sub-sampling Window search (Please see find_cars function in cell 3).

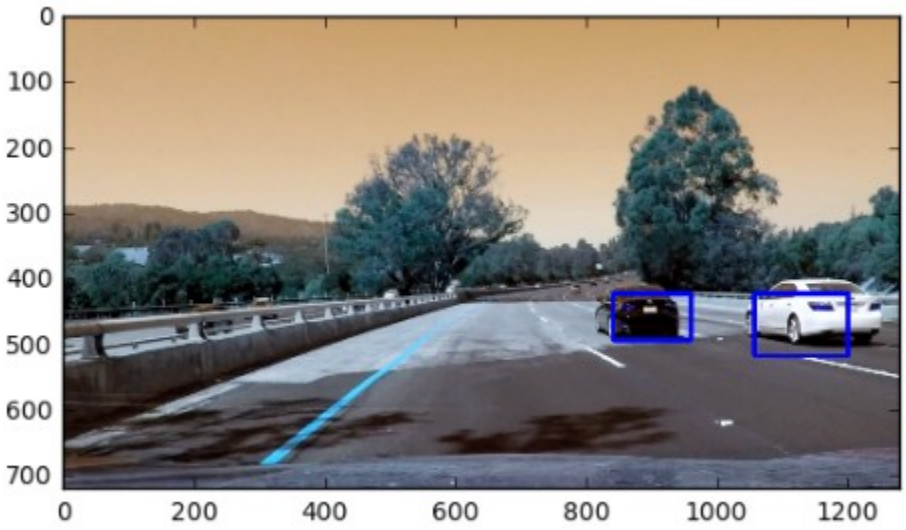
I chose ystart = 400, ystop = 656, scale = 1.5 since this optimized search time and searching well. I had set cells_per_step = 2 , which is analogous to overlapping windows. Instead of overlap, cells_per_step define how many cells to step

####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?

Ultimately I searched on two scales using HSV 3-channel HOG features, which provided a nice result. Here are some example images from the test_images directory:




0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

To optimize the results I had to use come heatmap counting as explained in the section below.

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.)

The video is by the name of project_video_output.mp4 in the project directory

####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.

This has been implemented inside the find_cars function (cell 3 In[11]).

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded (by 2)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. See function draw_labeled_bboxes() for this (cell 3 In[11].

Here are the results of my pipeline before and after applying heat map

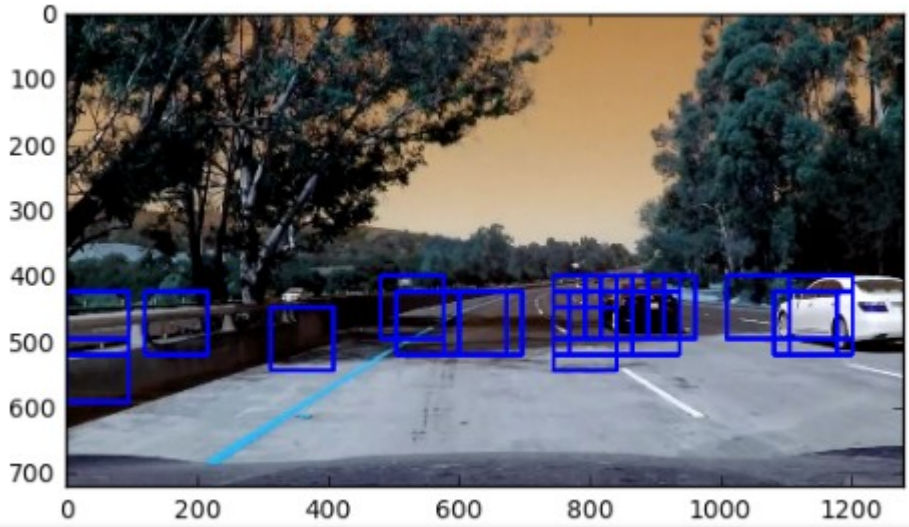


Illustration 1: Result after my Hog Sub-sampling Window searchHog Sub-sampling Window search

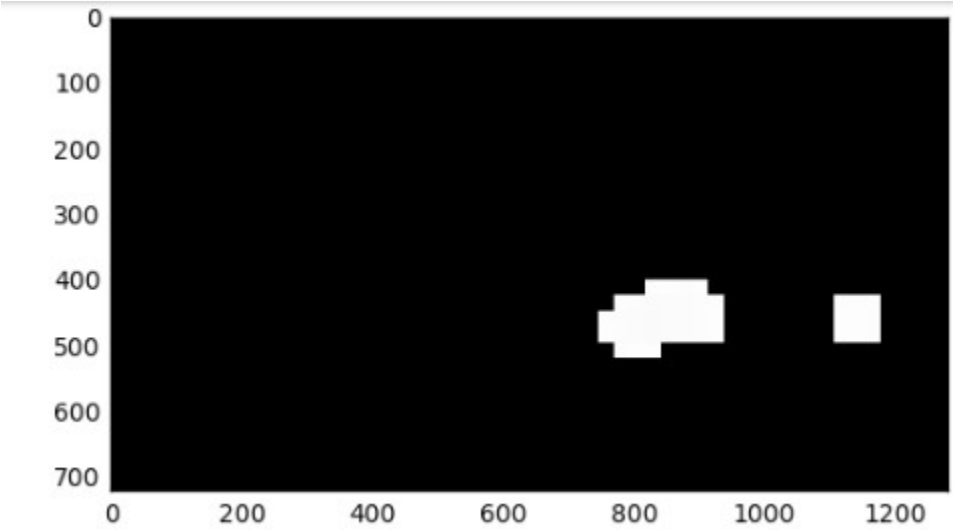


Illustration 2: Result after thresholding the heat map

Illustration 3: Drawing result of the labels function on the original image

###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?

I mainly faced problems in detecting false positives. There is no good threshold values that work perfectly for all the video frames.

My pipeline fails when the car is very far. I can avoid that by changing the scale of the sliding window but then it will increase the computation time.

Also, I have not incorporated color histogram features. I can do that as well.