Vehicle Detection Project

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
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- 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.
- Estimate a bounding box for vehicles detected.

Rubric Points

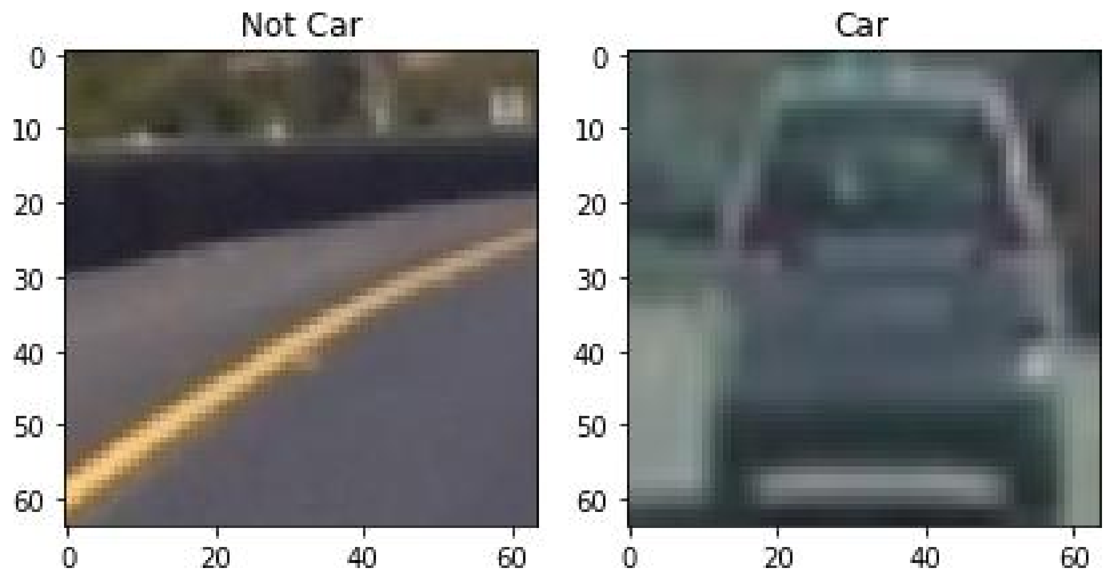
###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

###Histogram of Oriented Gradients (HOG)

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

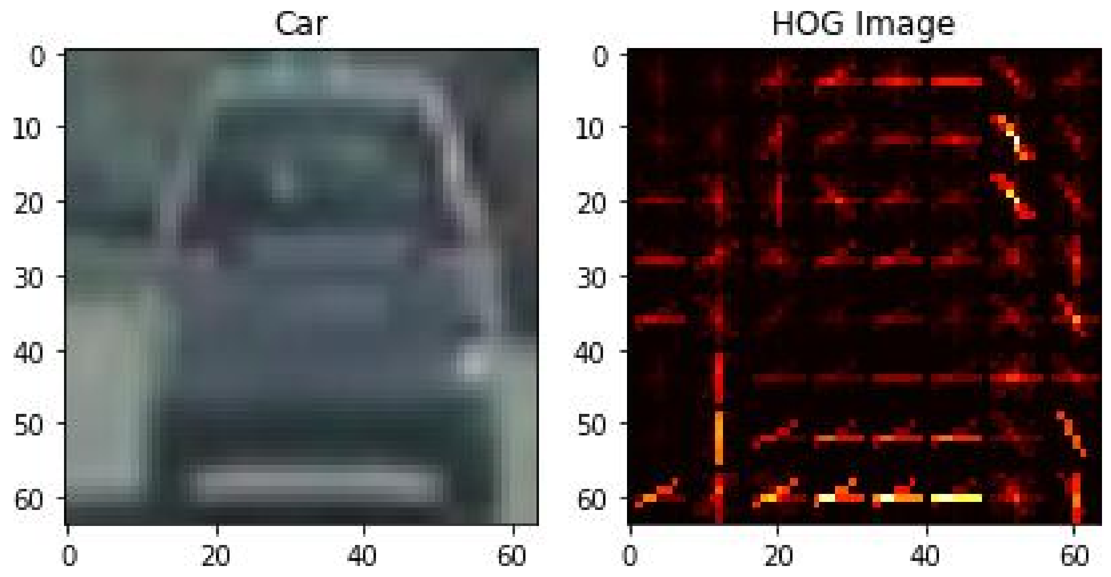
The code for this step is contained in the in lines #32 through #49 of the file called `Vehicle_Detection_and_lane_finding.py`).

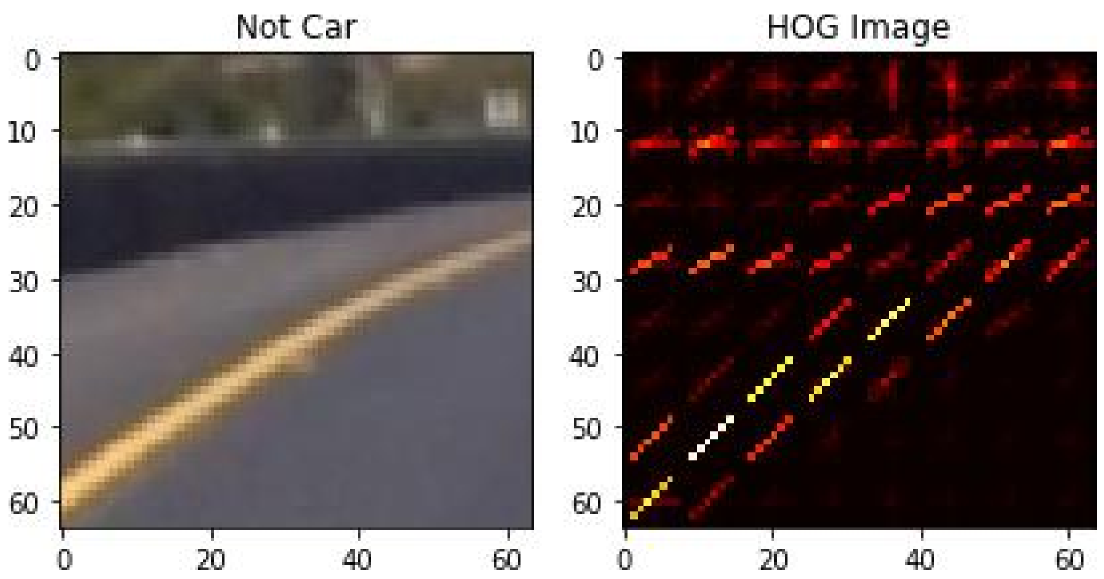
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



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 `LUV` color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:





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

I tried various combinations of parameters. I tried number of orientation bins that the gradient information will be split up into in the histogram from 6 to 12. For 8 orientation, I got better result. To define pixels_per_cell parameter, I choose the keep square shape with 8 x 8 pixels. For Block normalization, I selected cells_per_block parameter as 2 x 2 which helped produced a more robust feature set. To reduce the effects of shadows or other illumination variation, I also turned the transform_sqrt flag on.

####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 used vehicles /not vehicles (149MB) database to train the classifier. The code for this step is contained in the in lines #721 through #759 of the file called `Vehicle_Detection_and_lane_finding.py`).
For feature extraction, below techniques are used together:

1. Spatial Binning of Color: Raw pixel values of 16 X 16 resolution are added in feature vector. [code line # 52 to 56]
2. Color Histogram: To use the similar color distribution. [code line # 60 to 68]
3. Hog: Histogram of Oriented Gradients as explained above. [code line # 32 to 49]

The combined feature vector had the difference in magnitude of color and gradient based features. So I normalized the feature vector using StandardScaler() method. Also used transform() on the vector to perform standardization by centering and scaling. To avoid problems due to ordering of data, data is random shuffled. Then I split the data into a training and testing set to avoid overfitting and improve generalization.

SVM generally works well with HOG features so I selected SVM method for creating the classifier. I used Parameter tuning using GridSearchCV() to identify the best parameters. I selected Linear and rbf kernel method and 1, 10, 100, 1000 and 10000 as C values. The GridSearchCV() returned the best parameters as Linear Kernel and 1.0 as C value which are default values for the SVM method. So using these values classifier is fitted on the training data. The testing data accuracy with this option was 0.9803.

###Sliding Window Search

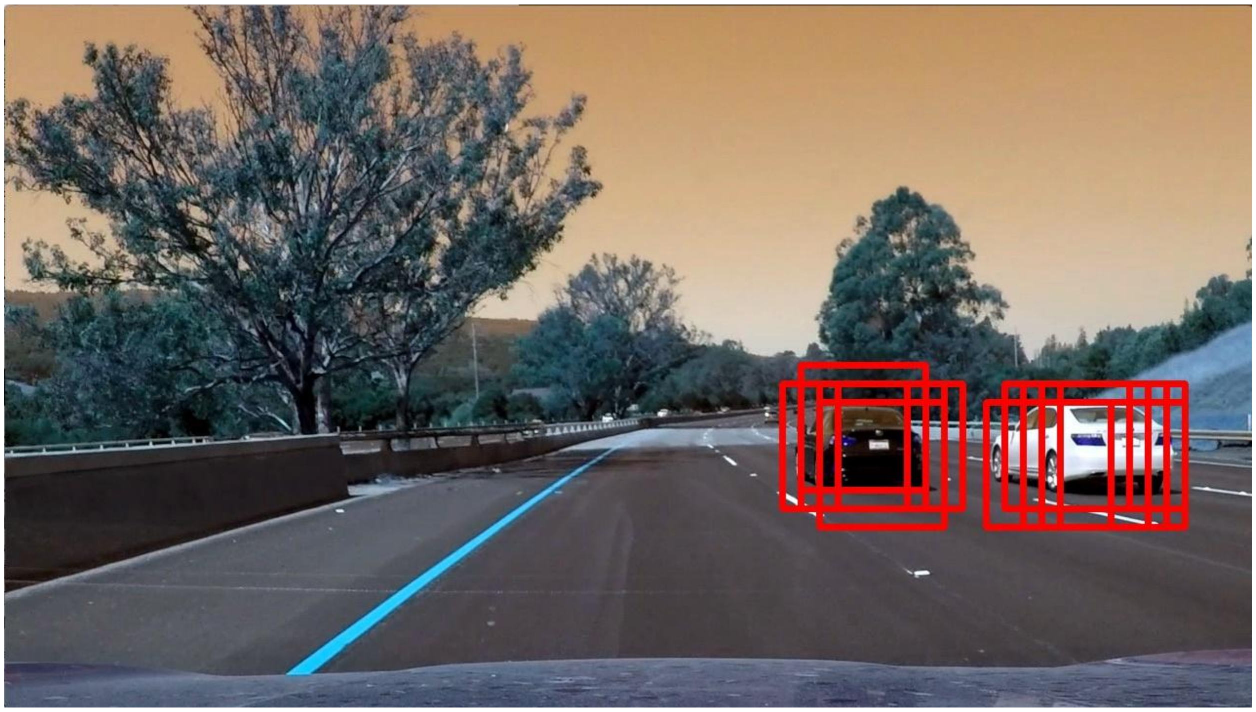
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for this step is contained in the lines #126 through #165 of the file called `Vehicle_Detection_and_lane_finding.py`). The call is made to the function from line #870.
I restricted the Y pixel value to 350 to 700 to avoid false positives in the sky and trees area. I started sliding window search with window size of 64 X 64 and overlap of 0.5. But these values were giving lot more false positives along with poor result of identifying the cars. With few experiments, I found the best value to be 128 X 128 window size and 0.85 as overlap.

####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 using LUV color space, HOG features (0 channel) plus spatially binned color (16 X 16 binning dimensions) and histograms of color (32 bins) in the feature vector, which provided a nice result. Here are some example images:

Image after running the classifier using sliding window approach:



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 output video is attached to the submission. Also it can be viewed at the link https://youtu.be/z-7IqEriAYc

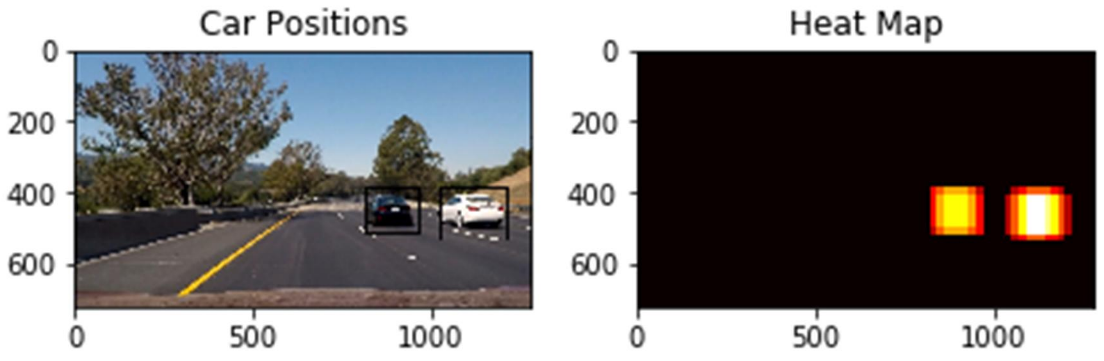
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

The code for this step is contained in the lines #260 through #289 of the file called `Vehicle_Detection_and_lane_finding.py`).

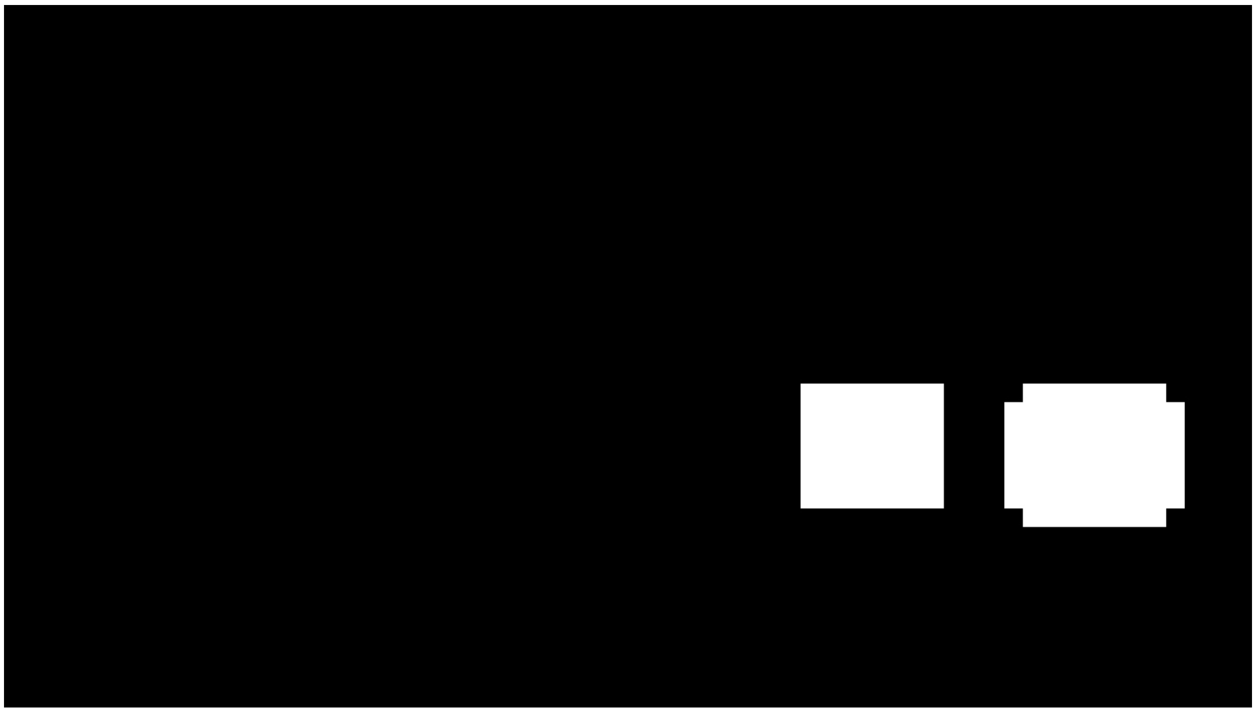
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in

the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.
Here's an example result showing the heatmap of the frame of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid frame of video.

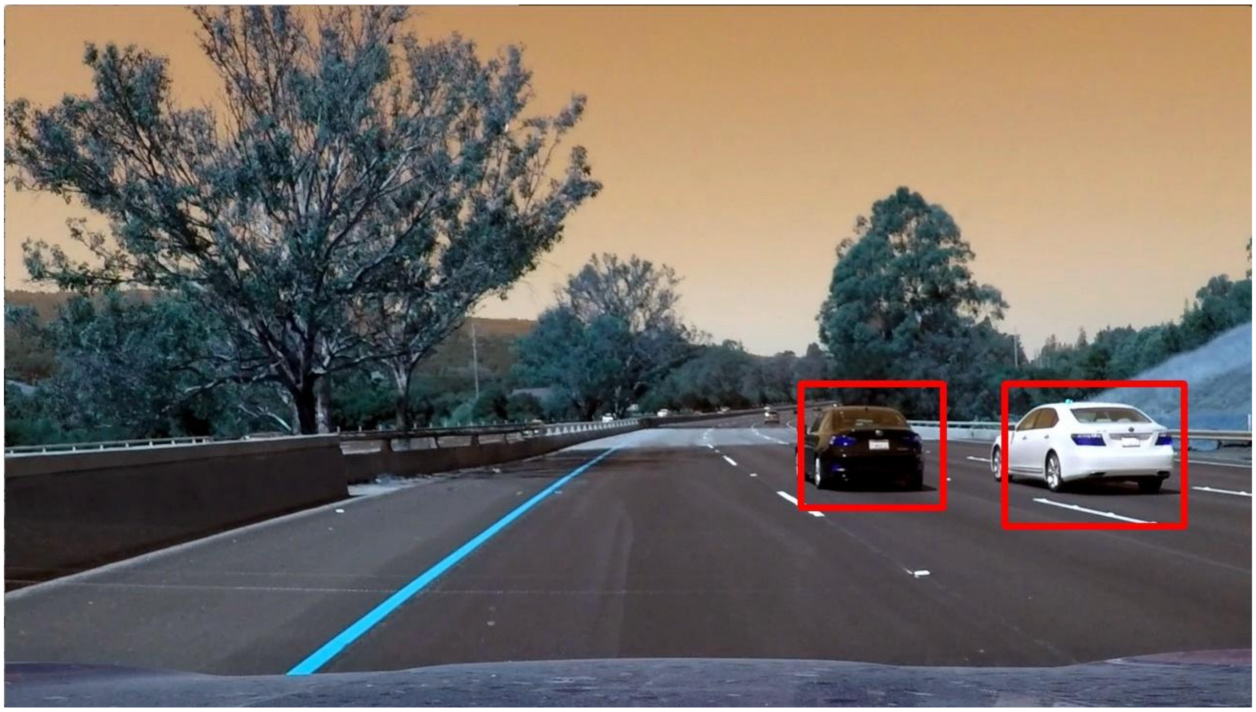
Here is the output of Heatmap for the same frame above:



Here is the output of scipy.ndimage.measurements.label() on the integrated heatmap:



Here the resulting bounding boxes drawn onto the frame finally:



###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 faced below problems in this project:

1. Training the classifier took multiple permutations to identify the correct car image. Even if the accuracy was more than 95% in all permutations, still the results were not as expected.
2. Sliding window technique took lot of permutations to come to the overlap and scaling parameters.
3. Heatmap and filtering was difficult to apply so that all the false positives are taken care off.

Pipeline may likely fail when there are multiple cars on the road. Also false positives may increase depending on the lighting conditions.

To make this implementation more robust, I will follow below pointers:

1. Identify any other ML technique to detect the cars.
2. Sliding window technique is inefficient and as overlap % increases, the execution gets much slower. This needs to be improved.
3. Even after using heatmap and filtering, false positives exist. They need to be reduced to 0.