# **Vehicle Detection Project**

## ## Writeup

**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.

### 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 second code cell of the IPython notebook.

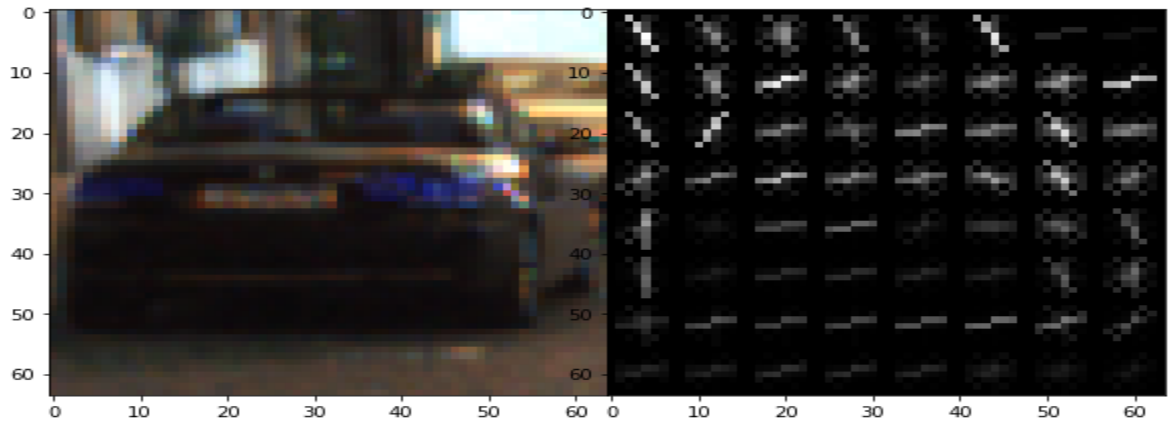
I have used a function get_hog_features, which will fetch the hog features and will return me the same.

I started by reading in all the `vehicle` and `non-vehicle` images.

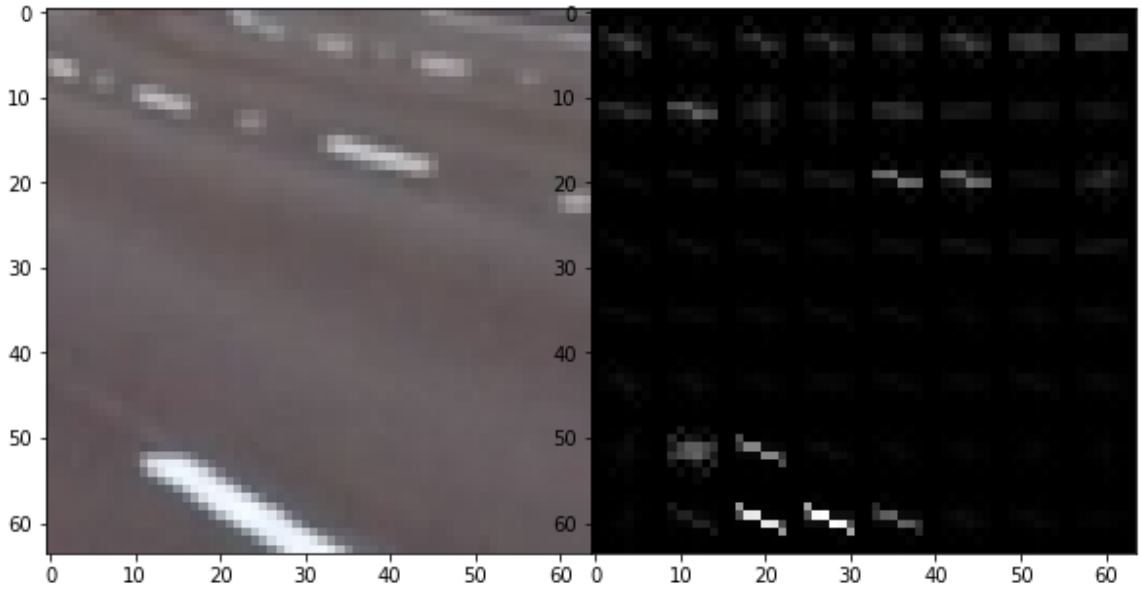
I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `LUV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` of the `vehicle` and `non-vehicle` classes with its hog features:

Car Image:



Non Car Image:




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

I tried various combinations of parameters and finally decided to use the following as it produced me good results from the beginning. When I was doing Udacity exercises in the classroom, I used to play with all the color spaces to get a feel of what's giving better result. At that time, I felt using YUV or LUV looks better. And using LUV makes the car data skewed almost based on the L channel. So I stick on to the L channel of the LUV color channel. And for the other parameters like orientation I chose 9, and pixels per cell 8, 8 and cells per block as 2, 2. Eventually it worked well for me.

#### 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).

Initially I trained my data using a linear SVM classifier. It gave good results but it was giving too much false positives. So I used **GridSearchCV** of scikit-learn to find the bet model. I gave Linear, rbf kernels and gave C values of 1, 5 and 10. I got the best model as rbf kernel with C value as 5.

From then on I used those best found values alone. It gave me good results.

I also used spatial binning to reduce the pixel of the training images to 16*16 pixels and used color histogram with 16 bins to get the color features of the training images.

The final feature vector is the combination of all the above. I got the feature vector length: 2580.

I have used 8500 images of cars and 8000 images of non-cars. I shuffled the data and made 20% as test set and trained the data with SVM classifier and got an accuracy of 99.38% which is good to have.

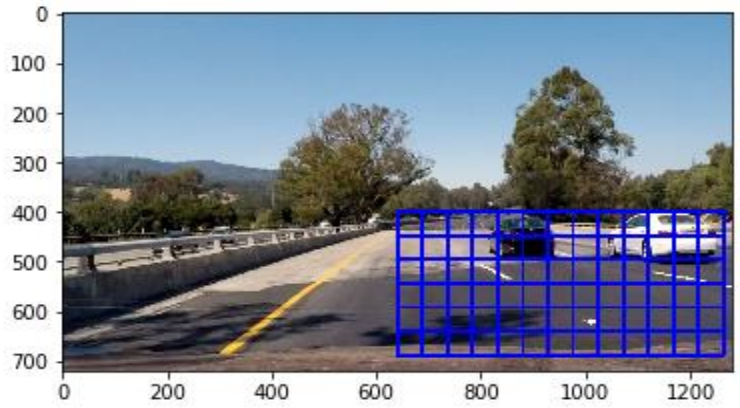
## ### Sliding Window Search

#### 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?

I decided to try with the one window that was provided in 34$^{th}$ lesson of classroom to start with. It was 96*96 window with 50% overlapping. It gave pretty good results with minimum false positives.

I have also limited my searching area of the y axis from 400 pixels row till the bottom of the image and skipped rest for the window.

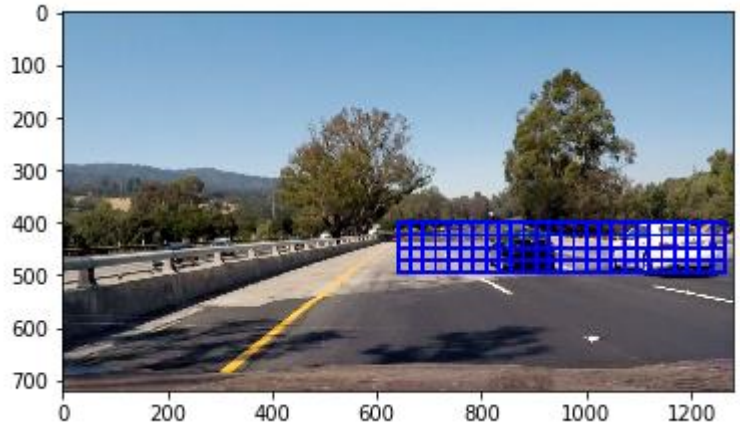
Here is the sliding window on an image:




The problem with that approach is that, it didn't capture certain cars. So I had to add one more window of 48*48 pixels with 50% overlapping.

I have also limited my searching area of the y axis from 400 pixels row till the 500 pixels row of the image and skipped rest for the window.

Here is the sliding window on an image:




So it resolved that problems. And I had a new problem of having too many false positives.

I am explaining how I resolved in the last before part of this write up.

And also note that I have also limited my searching area of the x axis from 640 pixels row till the right end of the image and skipped rest for the window as it reduces my processing time a lot. In our problem, we are not very much concerned with the image that are coming in the opposite lane. In a real life scenario, we have to include a few more of x. Processing this model itself took me around 30 Minutes for the video.

#### 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 sliding windows using LUV L-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 with the car found on them:




## ### Video Implementation

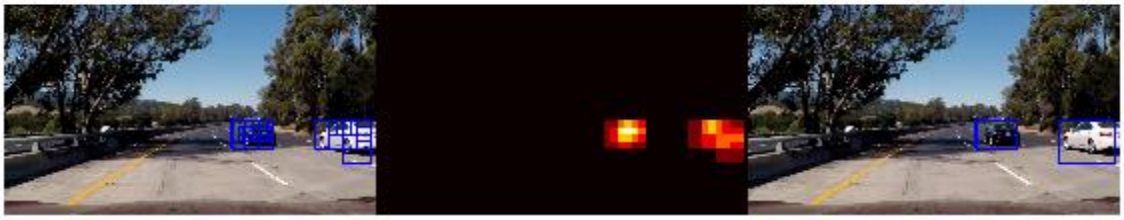
#### 1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

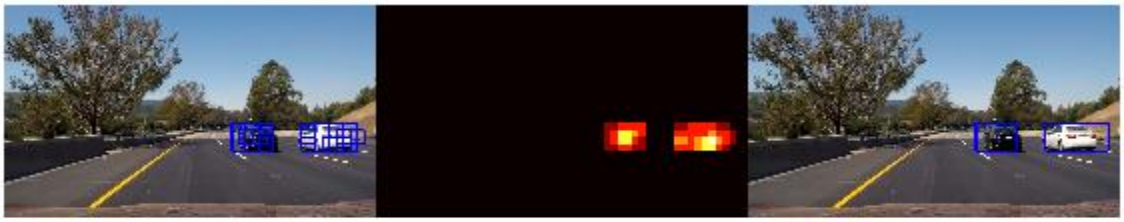
I have submitted the video with this write up.

#### 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.

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 detections of the vehicles multiple times with duplicates, the heatmap and the bounding boxes then correctly predicted for the test images.





This does not seems to solve the issue as false positives and the car are detected only once in certain cases as my model is pretty robust. Unfortunately it detects false positives in certain areas.

So given this is the case. I have recorded the heat for the previous 10 frames and taking the sum of all the heat and applying the threshold to the added sum and drawing the heat map to get the final bounding box of the frames. It avoided all the false positives which just appears once or twice in 10 frame and does not affect cars which is detected throughout.

All these are implemented in the function pipeline_vehicle_detection().

## ### 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?

As I said already, I chose the above said model after doing a lot of research. It seems to work pretty fine. I have also tried the optional challenge of adding my lane-finding algorithm from the last project to do simultaneous lane-finding and vehicle detection. It is working almost fine. It is having few false positives. As I have crossed the deadline, I am submitting the project. I guess it needs some more tuning to remove one or two false positives that I have in the optional challenge. Thus after which the model will be robust and can be used in real-time. But yeah!, I know it's a day dream. Since we use deep neural networks approach instead of this traditional method of classification. So, I also want to try out the convolutional neural network way of training and classifying the objects. So a long way to go before finding a robust model which can really be used in a self-driving car.