Vehicle Detection & Tracking

Term 1 – Project #5 - Self Driving Car Nano-Degree

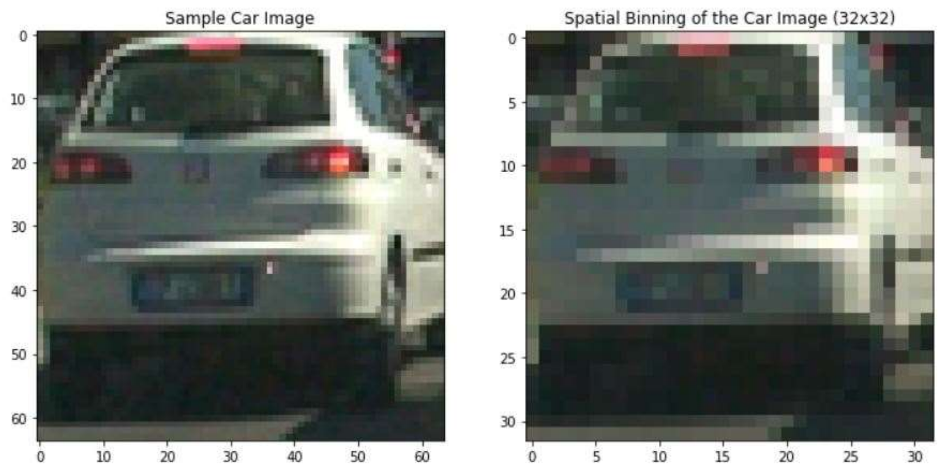
In this project we take a traditional 'Computer Vision' based approach to detect Vehicles in a video. The core areas of focus for this challenge is to:

1. Develop a classifier for Car vs Non Car Images
2. Develop a signature for identifying the Car Images
3. Scanning the Image for Cars
4. Tracking the Cars

In the interest of time, I used the code from tutorials to implement this project.

1. Features for identifying a Car:

- From the tutorials in the project, we needed data to identify the properties essential to determining whether the image given is that of a Car OR not.
- The robust features of a car are '**Shape**' and '**Color**'.
- To identify the shape we use a Histogram of Oriented Gradients (HOG).
- Next to identify colors, we use a Histogram of 3 Color Channels. The color space should be robust to variations in brightness in the images of cars. I experimented with RGB, HSV, YCrCb and YUV color spaces and found YCrCb & YUV to be very robust. I chose YUV as suggested in one of the forums.
- As an additional feature to identify Cars, we use a 32x32 Spatial Binning of an image.

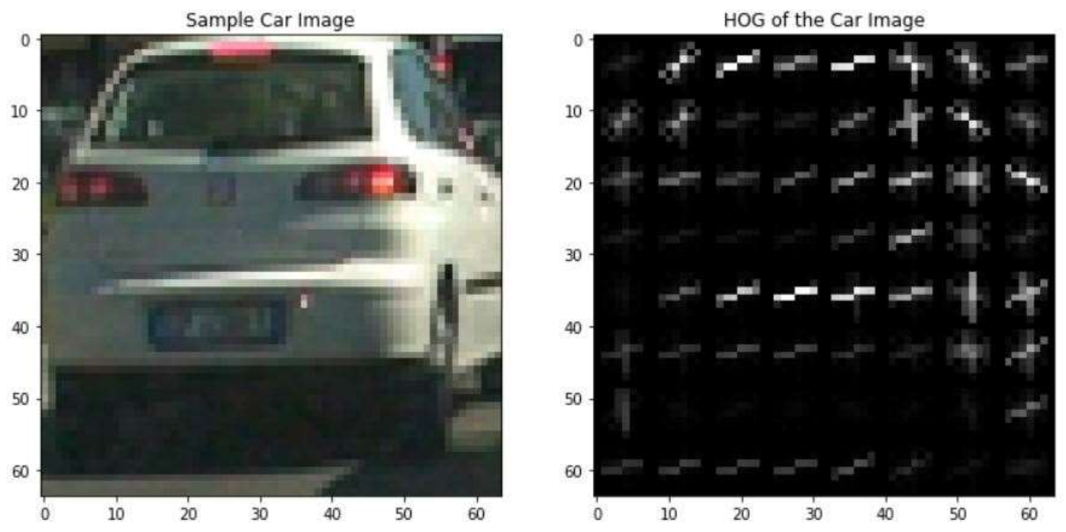


- So the final Feature Vector is in the order of :
  [**Spatial Binning + Histogram of Color + Histogram of Oriented Gradients**]

Histogram Of Oriented Gradients (HOG):

- As mentioned in the project lessons, I used SKImage's hog function (I read on the forums that OpenCV's hog function is faster).
- I used 11 Orientations bins, so that we capture maximum Gradient Information possible.

- I used 16 pixels per cell so that it runs faster and you have a bigger area to capture more gradient information.
- I used 2 cells per block, as I felt it is optimum to maintain the accuracy of the classifier. And we have already ensured performance by increasing the area of gradients with pixels per cell.
- I could not experiment with any Normalization methods due to paucity of time.



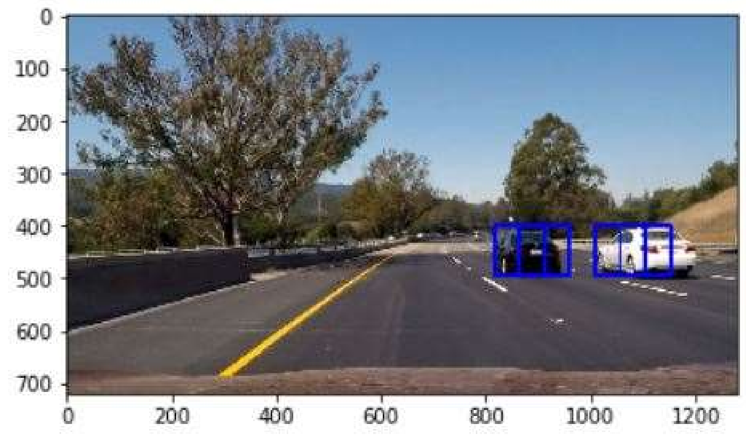
2. Classifier to identify Car vs. Not Car Images:

- I used a Linear Support Vector Machine as mentioned in the lessons, I could get an accuracy of 98.82% which I felt is sufficient for this project.
- I used the dataset from KITTI and GTI for this, 64x64 .png images.
- For paucity of time I did not experiment with any of the tuning parameters related to the classifier NOR did I include the Udacity dataset, its is in my todo list to work on them.

3. Scanning the image frame for Car Images:

- I used the a Sliding Window concept by sub-sampling the HOG data of an image.
- I focused on the lower half of the image, y=400 to 656 since those are the only areas where the cars can be found.
- I used a 64x64 pixels window size and an overlap of 50% i.e. set cells to shift per step to 2.
- I scaled the image by reducing the size of the region of interest to be 50% of the previous size, to speed up scanning.

```
Seconds to find cars a single image :  0.21
<matplotlib.image.AxesImage at 0x2cffb23ba20>
```



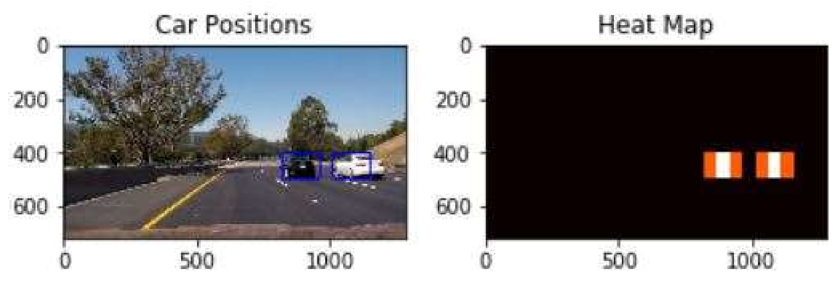
Failed Alternative approach of multi size scanning windows:

- I spent a lot of time trying to improve the performance of a multi size scanning window technique which I felt is the best way to scan the image for cars since the ones nearer to the camera will be bigger and the ones away will be smaller.
- I experimented with window sizes of 64x64, 128x128 and 256x256 from y=400 to 656, but I did run into problems of white car not being detected for a brief amount of time.
- I did find that increasing the overlap improved the accuracy of finding but did not solve the problem as it also slowed down the system.
- Even though it could do a decent job of finding and tracking the cars in the images, I did not continue further since it was eating up a lot of my time and it didn't match my expectations OR the results from the standard implementation.
- I will continue to work on finding where I went wrong later.

4. Tracking the Cars and eliminating the False Positive:

- Again could not focus much on this aspect, I used a Double Ended Queue and stored the heatmaps of the last 20 frames.
- Summed up the heatmaps and eliminated pixels with threshold of 10, i.e. 50% hit rate.

- Not happy with this implementation since it results in the loss of pixels in a new heatmap in the car detection as it moves with new frames (since their pixel strength will be less than the threshold).
- I would like to keep a track of the direction of the change of windows detected so that new windows which are persistent are tracked better and smoother.



Discussions:

- I believe a Deep Learning framework is best suited for this project but since the objective of this project was to experience the Computer Vision way of solving, I stuck to using the Computer Vision based solutions.
- ToDos:
    - Try using Canny Edge instead of HOG to identify shapes (doubting Canny Edge's ability to be robust to variations in Brightness)
    - Augment the Data set from (KITTI, GTI & Udacity) with Vertical & Horizontal Translations, rotations and brightness variations so that we can reduce the load on sliding window detection overlap. (May be a Neural Network will be better suited than a classifier)
    - Reduce the Feature Vectors and try to tune the Classifier for high accuracy.
    - Improve the detected Window Management by trying to map the detection in the new frame to the nearest previous ones.