**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

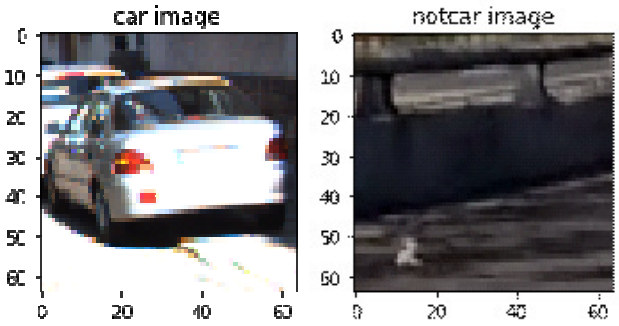
## Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

---

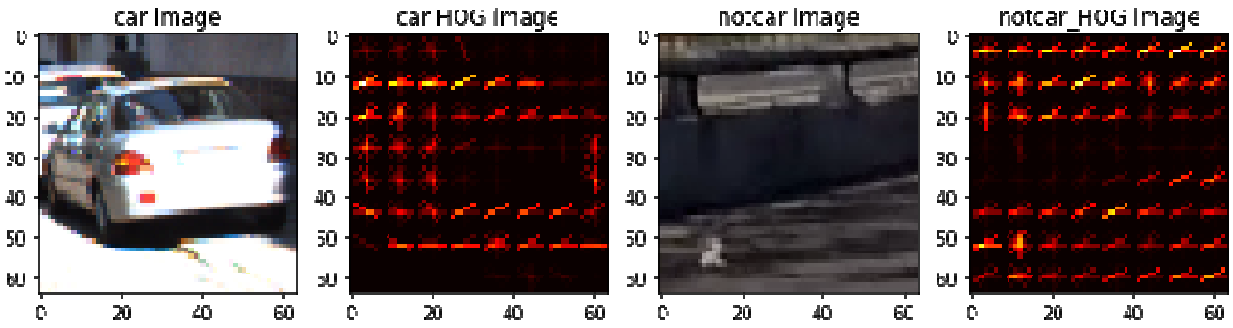
## Histogram of Oriented Gradients (HOG)

### 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 CELL #1 of the IPython notebook “Vehicle Detection.ipynb“.  The code cell imports all the required libraries and imports the vehicle and non-vehicle images in preparation to analyze the HOG features.  Below is one random example of a car vs a non-car image.




I explored several color spaces (such as YUV and YCrCb) and hog parameters (such as orientations, pixels per cell). I viewed several HOG transform outputs which appeared as follows for the RGB color space, 6 orientations, 8 pixels per cell, 2 cells per block, hog_channel 0, and 16x16 histogram bins:




The features appeared to capture the essence of car vs. not-car pretty well.

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

In CELL #5, I utilized some of the default parameters as discussed in the walkthrough video, namely the color space YCrCb because it tended to yield better results. Orientations were set to 9, which was mentioned in a whitepaper as being an optimal number of orientations. Pixels per cell were set to 4, and cells per block were set to 2 so that rendering times were reasonable while also limiting false positives. Finally HOG channels were set to ALL to better capture all three channels of the color space.

**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).**

CELL #6 is where I trained a linear SVM using the LinearSVCclass.The data sent to the classifier was already removing the mean and scaling to unit variance using StandardScaler.  Using the labeled images of car and not-car images, the SVC can be trained to recognize the difference between the two.  Approximately 8000-9000 images were used to train the SVM to recognize the difference between cars and not-cars. After the SVM was trained on 90% of the data, it was tested against 10% of the remaining data and given a score >99% after some mild tweaking of parameters.

```
18.970264673233032 sec to compute features
Using: 9 orientations, 8 pixels per cell, 2 cells per block, 32 histogram bins, and (32, 32) spatial sampling
Feature vector length: 8460
18.32 Seconds to train SVC...
Test Accuracy of SVC =  0.995
```

## 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?

In CELL #3, a sliding window function takes the existing window and utilizing a predefined window size, a matrix of windows is calculated for searching for car features. A 96x96 pixel window was selected for performance purposes since my laptop is fairly low-powered.  Overlap was set to 50% so that no area would go unchecked.

### 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?

Using the sliding window function, I extracted features using all 3 channels of YCrCb to identify when a car was present.  I found that I had to bump up the number of pixels per bin slightly to keep false positives down. Below are some examples of the classifier working properly on static images with some false positives.

## Video Implementation

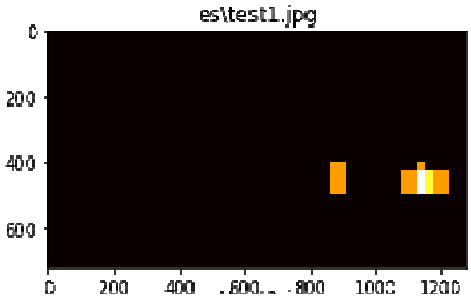
**1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)**

My final output is under output_images/final.mp4.

**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.**

In CELL #10 is where I handled false positives. Every time there was a positive detection, the position of the detection was recorded. Next, overlapping detections increased the heat associated with each vehicle. To help filter out false positives, thresholding above a level of 2 was used to eliminate single frames where detections occurred. In addition, I averaged across approximately 10 frames to try eliminate single frames where false positives were unavoidable. This eliminated pretty much all false positives while still keeping decent sensitivity on the real vehicles in the scene.

Below is an example of a frame from the video, in addition to its heatmap.


## 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?**

One of the key areas I made tuning adjustments was using the scale factor. When it was lowered, it tended to finely tune the sensitivity of the classifier to cars further away, but also caused more false positives, especially in traffic coming from the other direction. However, raising the scale factor tended to cause the classifier to ignore nearby cars in adjacent lanes that should be detected. The pipeline would likely fail more if the traffic divider was not present, as cars in the opposite direction were definitely a problem for me. Masking that part of the image would not have been practical.

To make it more robust, perhaps additional training examples would have been helpful. I could also perhaps test more tuning parameters to try to find a better “sweet spot” where false detections were minimized but true positives were more reliable with stronger heat on the heat map. I found this project challenging to complete successfully compared with some of the other projects, but I sure learned a lot!