# Vehicle Detection

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
**Done**

* Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
**Done**

* Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
**Done**

* Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
**Done**

* 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.
**Done**

* Estimate a bounding box for vehicles detected.
**Done**

[//]: # (Image References)
[image1]: ./examples/car_not_car.png
[image2]: ./examples/HOG_example.jpg
[image3]: ./examples/sliding_windows.jpg
[image4]: ./examples/sliding_window.jpg
[image5]: ./examples/bboxes_and_heat.png
[image6]: ./examples/labels_map.png
[image7]: ./examples/output_bboxes.png
[video1]: ./project_video.mp4

## [Rubric](https://review.udacity.com/#!/rubrics/513/view) Points
###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

---
###Writeup / README

**####1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.  You can submit your writeup as markdown or pdf.  [Here](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.**

*You're reading it!*

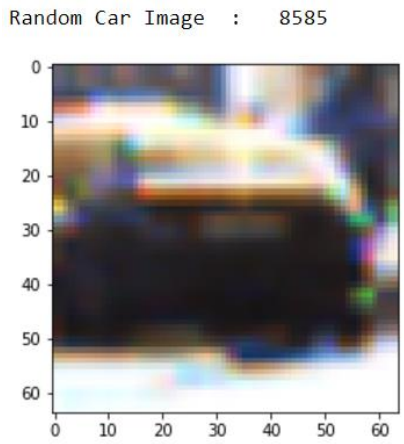
###Histogram of Oriented Gradients (HOG)

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

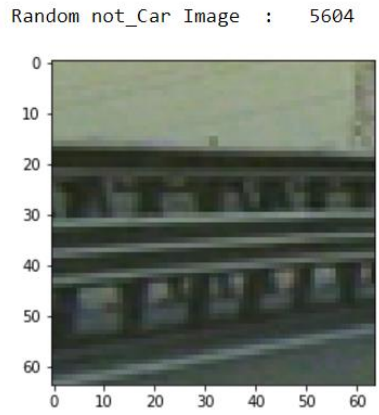
*The code for this step is contained in the second and third code cell of the IPython notebook.*

*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:*

*Vehicle*




*Non- Vehicle*




After setting up the code (where credit is due to Udacity Classes), I set it up to take input of parameters,
Code cells 7, 8 and 9 to view individual feature on a random image, and then Code cell 10 to run through multiple parameters while calculating accuracy.

Then I set code cell 12 to show me which selection of parameters were the best to use.

After multiple runs, I found that Spatial Binning added no extra qualities to me Classifier.

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

*Answered Above*

**####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).**

*After The above notes, I finally re-instantiate the Linear SVM Classifier, using the best selected parameters in Code Cell 13.*

I attempted to use Canny features as well to calculate HOG values but it did seemed the same. This can be seen in the test_wihtCanny and test_withoutCanny images in the github repo

**###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?**

Utilizing the code from the Udacity class, I set up the search windows
Also, I set up a region of interest to assume that horizon starts at 44 on the y axis and eliminate anything above that point

**####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 YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a sufficient result. The Classifier noted a higher accuracy with use of all three features, but I feel it visually performed better without Spatial Binning.  Example images of the classifier are shown after* ***Code Cell 20***

**### 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 result is in this GitHub Repo under the name
**Main_Project_Video_Processed.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.**

*After using the normal states of sliding through windows and searching in windows, I checked the decision function of the classifier(which provides a confidence score on its prediction and I filtered out those with confidence lower than 0.8 thus removing false positives.*
*On the Video, I also did an extra filter where I noticed that if any Not_Car objects slipped by the confidence filter, they were never present in more than two frames. Thus I setup a cache that only added hot windows that were present for more than 3 Frames.*

You can see the results of this test procedure in the folder Video_Frames

---

**###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 also, came up with another process to track objects but did not have time to implement it. I will discuss this method to further improve this.*

*To start of define an Abstract class Object with the following attributes:*

*Initial frame in which it was detected.*

*Initial Position where it was detected.*

*During the pipeline:*

*Setup a cache to see if it is present in more than three frames and filter out Not Car objects.*

*Define one further inherited classes "Car" which has the following methods:*

*Instantiate Car object with the initial cache and Frame Number.*

*Calculate the midpoint of the bounding box.*

*Assign label.*

*Calculate distance traveled over cache*

*Calculate Velocity over cache*

*Calculate projected position for next Frame.*

*% In Main pipeline, iterate through bounding boxes and send to next method*

*Check if position is within range of projected position form previous function and assign to correct label of Car.*

*If Car not found, send projected position to cache for a max of 5 Frames, else Assume Car not present.*

*Then Send last position and label number in the cache to draw boundoing boxes function.*