# **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](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.

---

Image references.
[All images found in './output_images/']

IMG1 – Output of HOG feature function (orient 9, ppc 8, cpb 2) – final setting
IMG2 – Output of HOG with orient = 2
IMG3 – Output of HOG with orient = 15
IMG4 – Output of HOG with orient 9 but ppc 4 and cpb5
IMG5 – Testing image for  detection in image pipeline
IMG6 – Testing image with all the hot windows detected shown as bounding boxes
IMG7 – Testing image w. bounding boxes shown after applying threshold and label()
IMG8-13 – Consecutive frames from test_video and their individual heatmaps
IMG14 – Bounding boxes shown on the last frame of the test_video that are the result of combining all the 6 heatmaps and running them through threshold

## ###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 is located in several cells of the P5-vehicle-detection jupyter notebook:

1. [CELL 4] I started by defining the function 'hogify()' enabling me to extract HOG features from a single image. It allows to work with different color spaces, as well as different (or multiple channels).

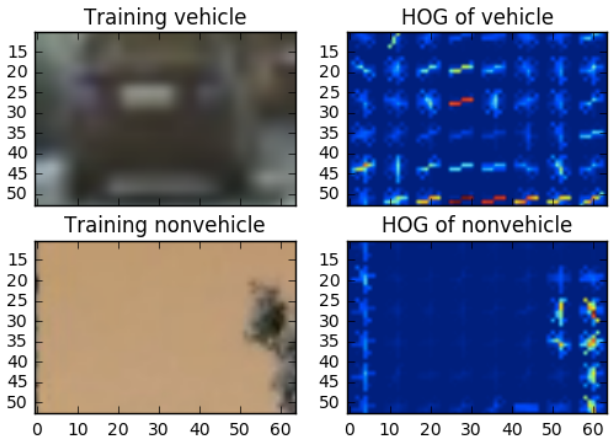
2. [CELL 7] I next defined a helper function 'get_features()' to allow me extracting and combining different types of features, e.g., HOG and color histogram.

3. I explored multiple color spaces and combination of features. I found that using color histogram does only marginally improve the model accuracy. Also, from different color spaces, I found YCrCb to be most consistent / reliable. I ended up using all 3 channels, although I found that this slowed my pipeline down significantly, it provided significant additional precision.

4. [CELL 8] As next step, I loaded all training images using glob (vehicles and non-vehicles separately), and then iterated through each, calling the get_features() function on them. This resulted in two sets of arrays, with vehicle and nonvehicle features respectively

5. I tested the HOG extraction process on sample images from the training set – below are the results:

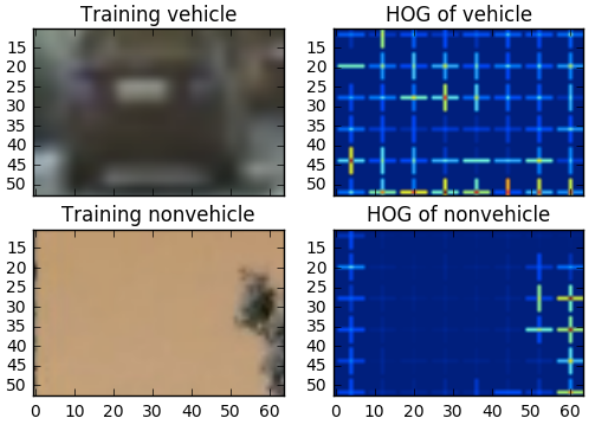
[IMG1]



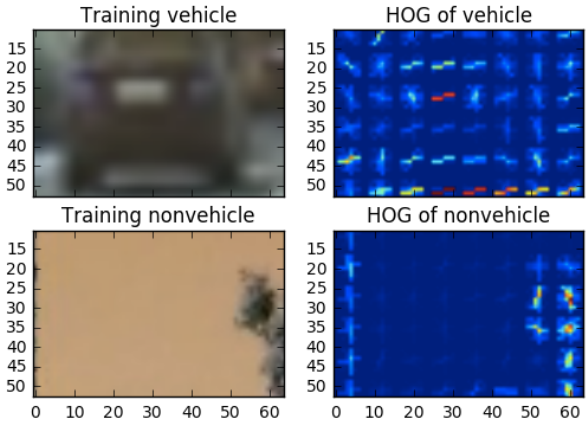

## ####2. Explain how you settled on your final choice of HOG parameters.

I tried various combinations of parameters and found the need to balance the number of resulting features to make sure the algo runs quickly and without overfitting, while being precise enough. For example, I've experimented with different number of orientations. At the low end, I tried 2 – with following result:

[IMG2]



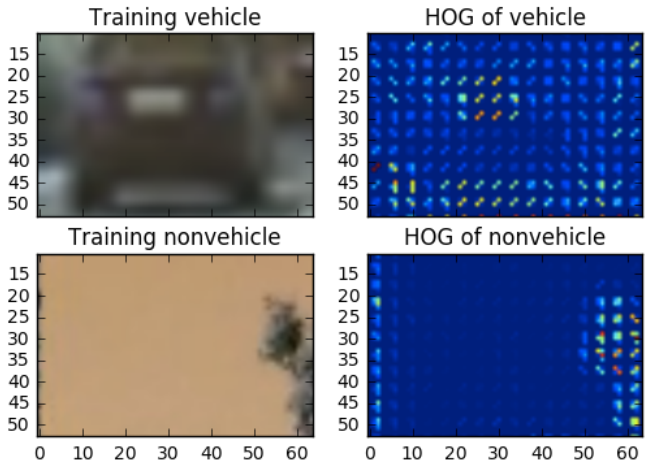

At the high end, I tried 15 orientations, with this result:
[IMG3]




I ended up using 9 orientations, because 2 clearly didn't provide enough degrees of freedom for the classifier (too many false positives) while 15 didn't bring any incremental precision. I found between 5 and 9 orientations to be the optimal number for this application.

Similarly, I experimented with different settings for pixel per cell and cells per block. For example, this was the image of HOG features using 4 ppc and 5 cpb:

[IMG4]




I ended up using the parameters 8 pixels per cell and 2 cells per block, which seems to yield good result.

**####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).**

1. [CELL 10] I constructed the X and y arrays required to feed the classifier by combining the vehicle and nonvehicle arrays, and creating a corresponding y array with 1s and 0s respectively.

2. [CELL 11] I then split my X and y into training and testing data sets

3. [CELL 13] I trained a linear SVC using the X_train and y_train arrays, and then tested its precision using the test X and test y arrays. My final configuration yielded ~97% accuracy.

## ###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?

Sliding search is implemented using two functions. First, in [CELL 14] a set of all windows is created which will be fed to the classifier using the. I found that using 50% overlap in x and y axis provided detailed enough coverage, without creating too many windows in total (thus slowing the algo down). I also ended up using window scale 64x64 which provided good detection rates both on far and near end of the screen. The second step involves using the trained classifier to make a prediction [CELL 15], function 'search_windows()'. Here I've replaced the .predict function with the .decision_function, which allowed me to implement a threshold and served as a first line of defense against false positives (lines 81-82).

Additional helper function was implemented in [CELL 18] to enable me to draw detected boxes on the images.

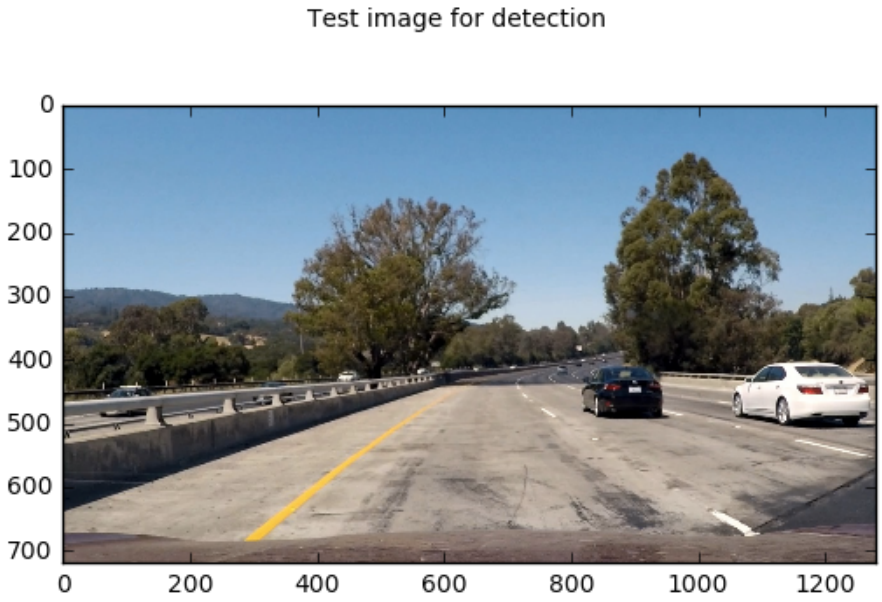
### ####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on single scale using YCrCb 3-channel HOG features, as I found that with precisely tuned parameters I can get good results with this setup, although it runs rather slowly (~1-2 frames / sec on a 2013 Macbook Air laptop).

To optimize the performance of my classifier, I iteratively explored tuning of various hyperparameters, and observed which yielded the best results. In particular I tweaked (mostly one or two at a time):
1. Color spaces used

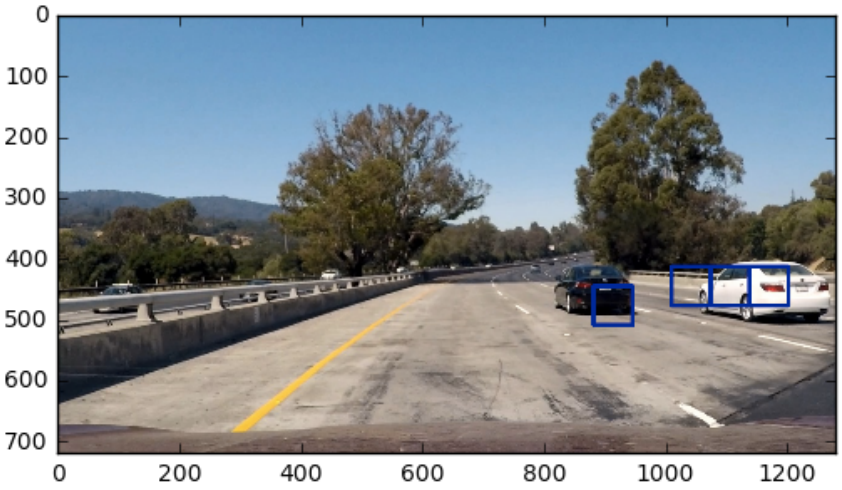
2. HOG parameters (orient, ppc, cpb)
3. Channels used for HOG detection
4. Threshold for the SVC's .decision_function
5. Heatmap threshold [for combining bounding boxes]
[6. Length of cache, i.e., how many frames were averaged, in the video pipeline]

[IMG5 ]



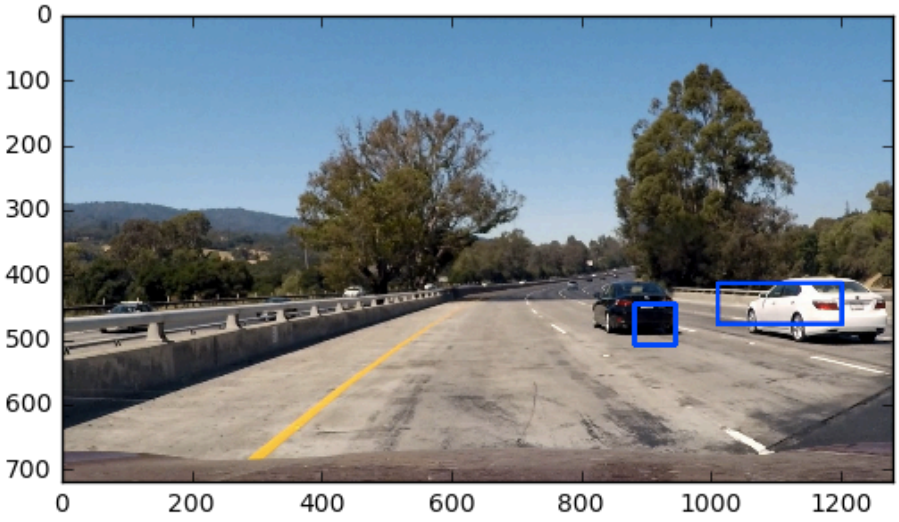

[IMG6]

Hot windows detected on a single video frame




[IMG7]

Bounding boxes on a single frame combined after applying heatmap and threshold



An additional step I took to enhance the performance of my classifier was to augment the training data set by extracting an extra 1000 images of vehicles and non-vehicles each from the video.

Finally, I limited the range where the algo looks for cars in the image to the area of the road where vehicles could be expected to appear, thus eliminating additional false positives from the sky and treetops which confused my classifier previously.

## ### 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.)

Here's a [link to my video result](./P5_v24.mp4)

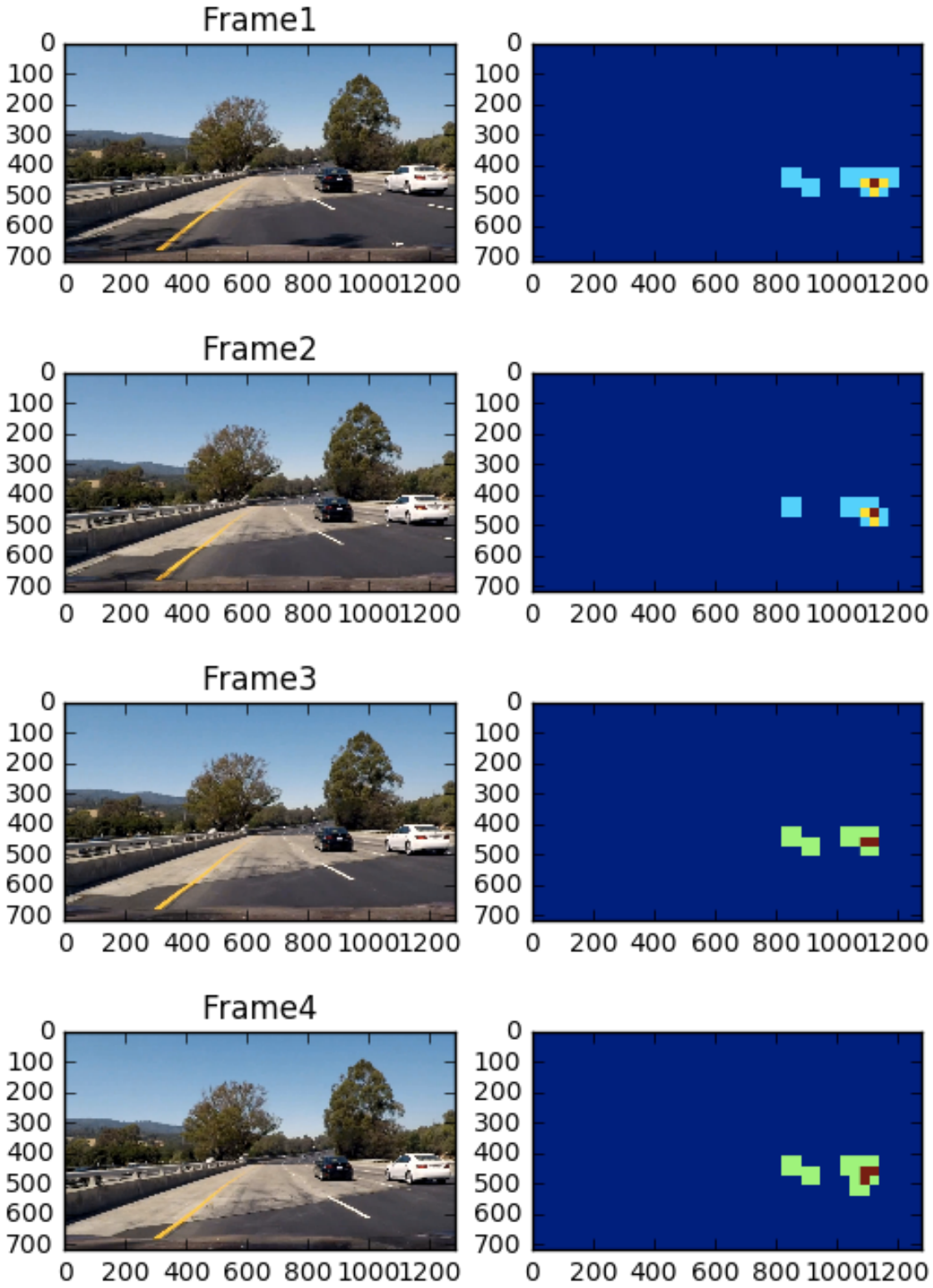
### ####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

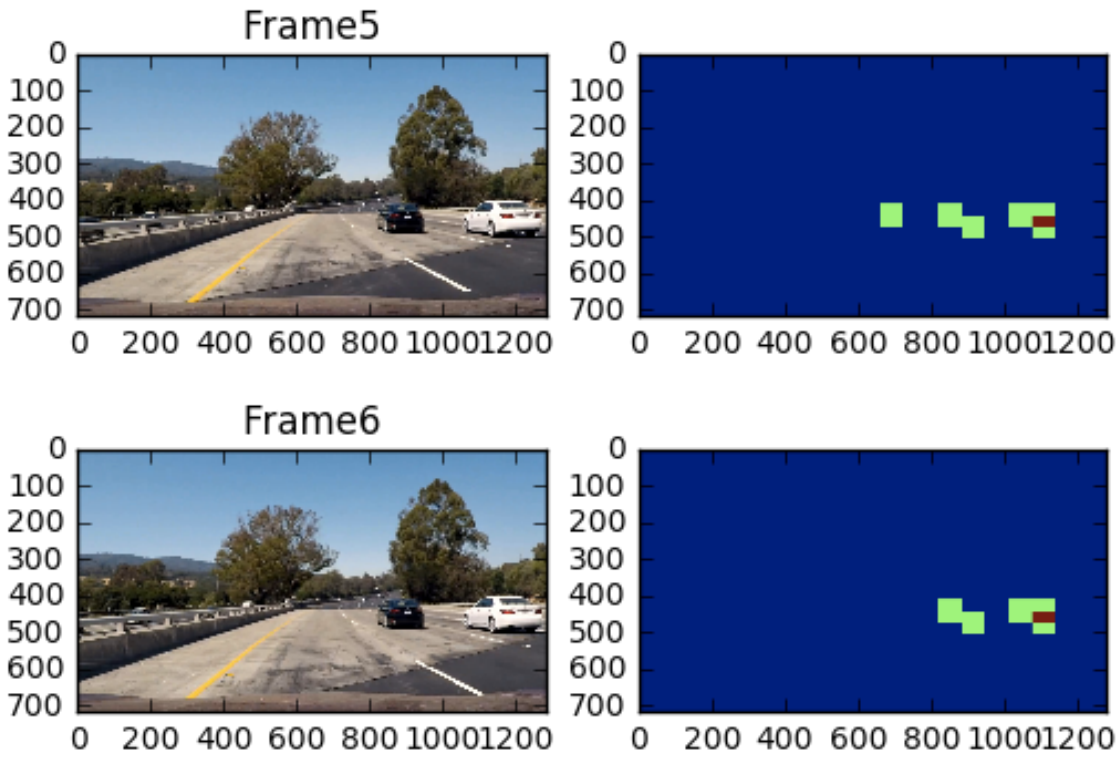
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.
For my final implementation, I looked at 30 consecutive frames, and set the threshold value of 7 required for positive detection. I discovered these values empirically, by tuning the parameters and comparing the results.
The helper functions I used to do this can be found in [CELL 20] and [CELL 24]

Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

### Here are six frames and their corresponding heatmaps:

[IMG 8-13]

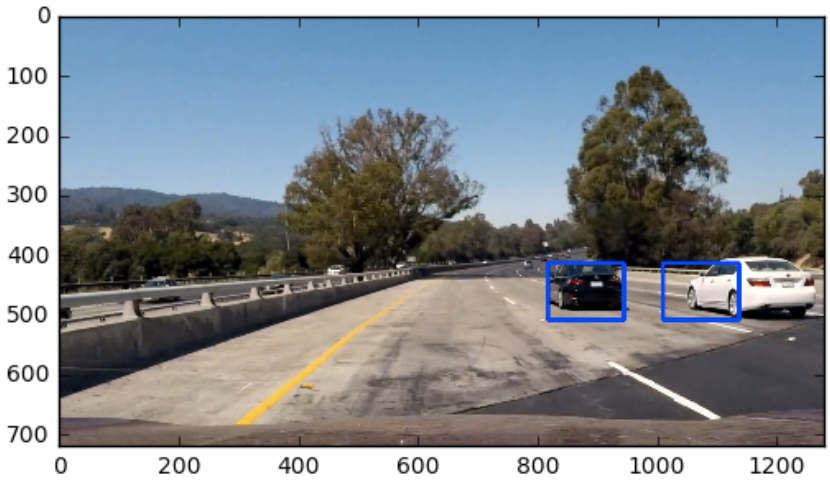

Frame1
Frame2
Frame3
Frame4
0
100
200
300
400
500
600
700
0 200 400 600 800 10001200




For these 6 frames  I combined the heatmaps [CELL 24] and then called the `scipy.ndimage.measurements.label()` on the integrated heatmap. Here, the resulting bounding boxes are drawn onto the last frame in the series:

[IMG14]

Bounding boxes as result of applying the label() func on combined 6 heatmaps




---

## ###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?

The biggest challenge I faced with the project was finding the optimal setting for all the possible parameters in the pipeline. In particular, I think of this challenge as two steps:
1. Finding the right parameters / color spaces / feature functions etc. to train the classifier to be reasonably robust and detect cars reliably but with reasonably small number of false positives
2. Tweak all the other parameters in the pipeline (e.g., heatmap threshold, number of frames to be averaged, sliding windows scale, number, and position, etc.) so that in the video pipeline most of the time, cars are correctly detected and most of the time, there are no false positives.
Getting these two steps right was a very iterative process for me, especially because one sometimes has to go through all the steps, and rerun the whole pipeline to find out if the end result is satisfactory (i.e., an improvement in some part of the pipeline does not necessarily mean better end product). It has taken me a while to develop an intuition for all the above mentioned parameters and get satisfactory result.

My pipeline is likely to fail under a number of conditions, e.g.,
- Strange weather conditions, in particular rain, fog, or strong shadows on the road
- Cars of unusual / reflective colors, not showing strong enough gradients
- Fast moving cars, esp. cars moving in opposite direction that move too fast to cross the threshold function
- Cars of unusual shape / not present in the training set (trailers, long-vehicles, motorbikes, etc.)

To improve this pipeline further I would:
- Gather more training data and/or augment existing training set – feels that to train a really robust classifier a much broader range of data would be needed
- Experiment with different classifiers – I feel that a properly implemented and trained CNN would do a better job at classifying than the simple LinearSVC I used, if for no other reason for speed of detection, and ability to control the parameters / inner piping of the classifier in a more precise way
- Streamline the pipeline to improve performance – at the moment it's quite slow, not running anywhere near real time (1-2 fps). A lot of the routines, e.g., HOG detection are not implemented optimally – the HOG detection is called for each window instead of the whole image at once, etc.