Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

Writeup / README

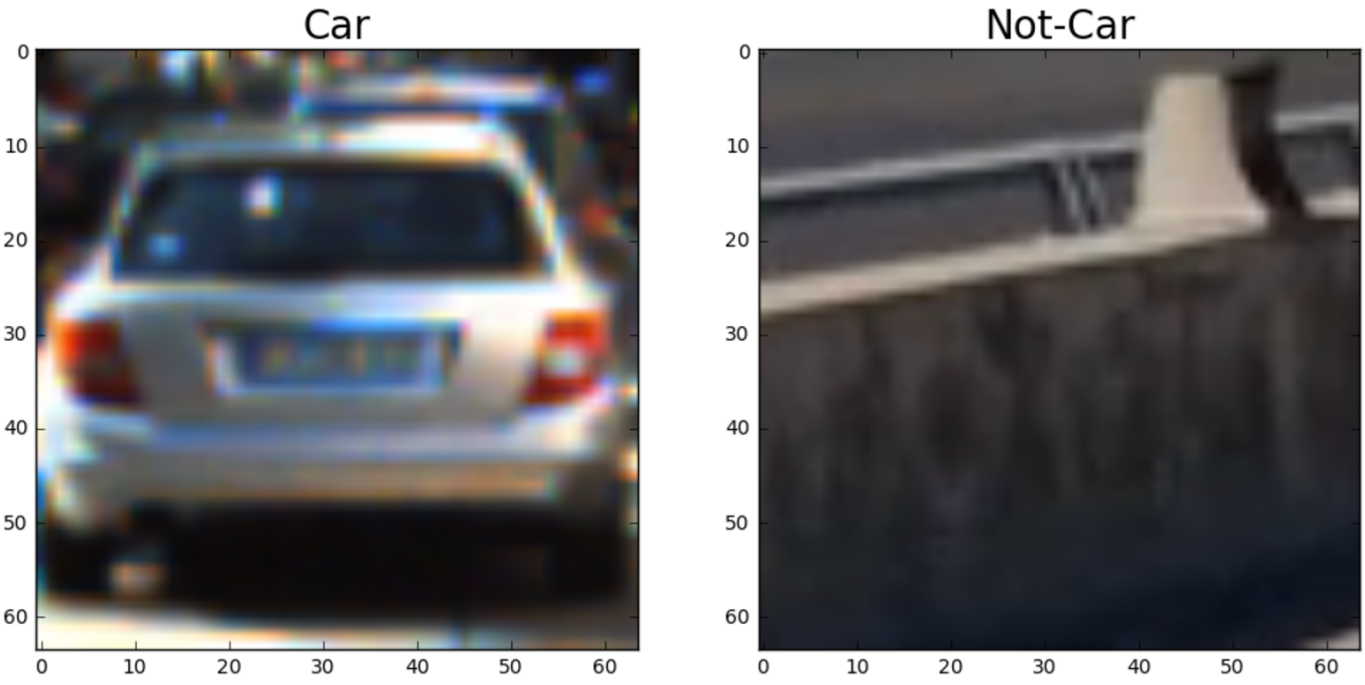
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here (https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and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

You're reading it!

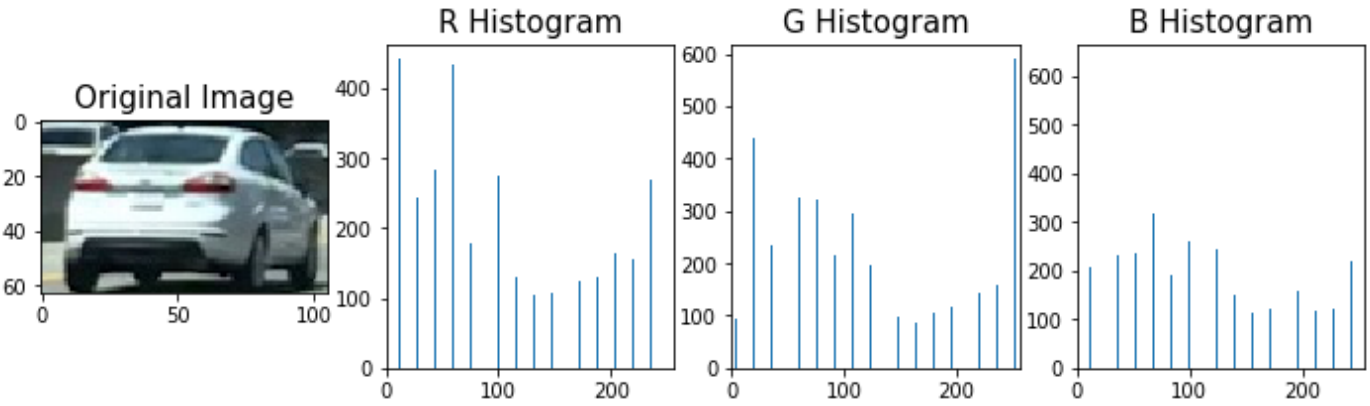
Histogram of Oriented Gradients (HOG)

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

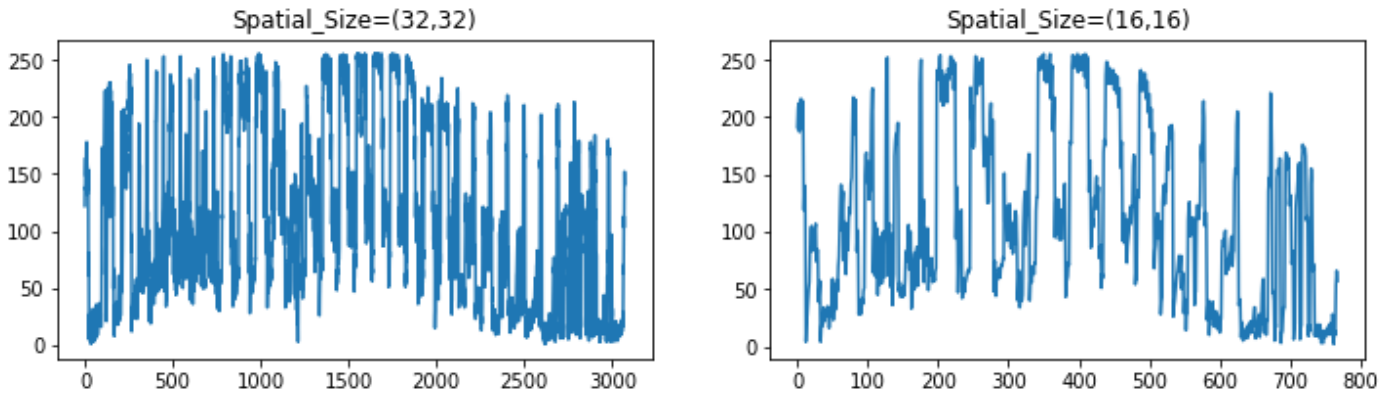
I started by reading(code cell [2]) in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



I have defined a function `color_hist()` to extract(code cell [3]) color histogram features. This function takes in color image and number of histogram bins and returns the concatenated color histogram features. Below figure visualizes the color histogram features of an RGB image.



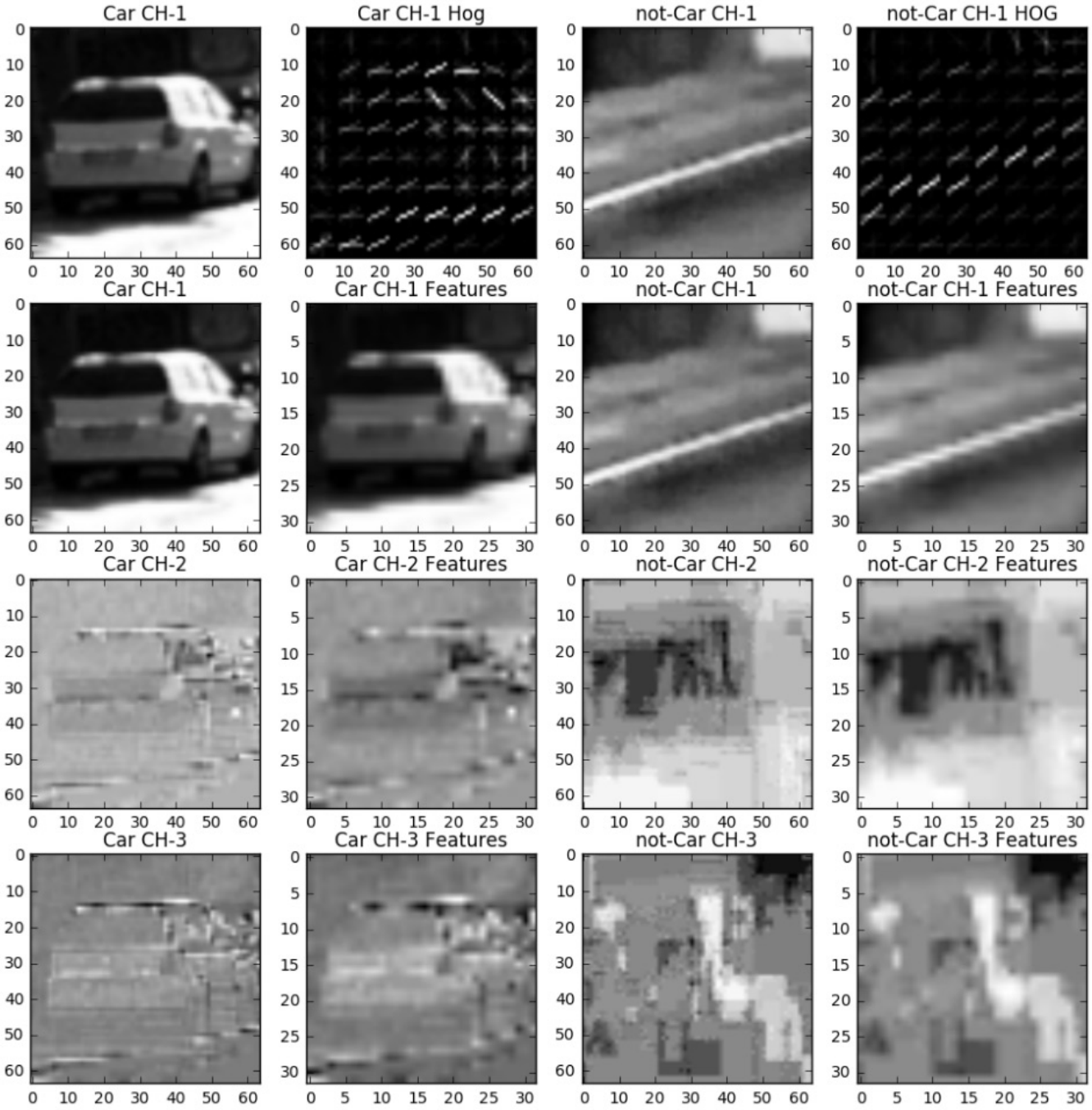
The function `bin_spatial()` takes in a image and a new image size(spatial_size) and returns a feature vector. The below figure visualizes the feature vectors after resizing the image.



Extract HOG Features

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. The code to extract HOG features is given in code cell [6] and [7].

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



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

I tried various combinations of HOG parameters, Spatial sizes and histogram bins. I settled on the final choice of parameters based upon the accuracy of the SVM classifier and the time it taken to extract and train the classifier. Below table shows the various combinations of parameters used and accuracy I got from SVM classifier.

First I tried using only spatially binned color and color histograms. Below table shows the various parameters used and the accuracy I got from SVM classifier.

Bins	spatial size	Color	Accuracy	Time to train SVC
8	(8,8)	RGB	0.8925	6.24
16	(8,8)	RGB	0.9057	6.24
32	(8,8)	RGB	0.9046	5.99
8	(16,16)	RGB	0.9237	15.14
10	(16,16)	RGB	0.9206	15.36
16	(16,16)	RGB	0.922	15.16
32	(16,16)	RGB	0.9113	15.46
8	(32,32)	RGB	0.9167	43.6
16	(32,32)	RGB	0.9093	39.55
16	(8,8)	HSV	0.8899	6.28
8	(16,16)	HSV	0.9113	15.21
16	(16,16)	HSV	0.9217	14.48
8	(32,32)	HSv	0.8849	30.57
16	(8,8)	LUV	0.9398	4.26
8	(16,16)	LUV	0.9406	9.95
8	(32,32)	LUV	0.9161	28.44
16	(16,16)	LUV	0.9448	9.13
16	(8,8)	HLS	0.8913	5.93
8	(16,16)	HLS	0.9065	15.36
8	(32,32)	HLS	0.8789	28.36
16	(16,16)	HLS	0.9099	14.62
16	(8,8)	YUV	0.9048	5.6
8	(16,16)	YUV	0.9234	14.28
8	(32,32)	YUV	0.9032	30.88
16	(16,16)	YUV	0.9184	14.19
16	(16,16)	YCrCb	0.9243	14.95
16	(8,8)	YCrCb	0.9057	6.18
8	(16,16)	YCrCb	0.9203	14.69
8	(32,32)	YCrCb	0.895	33.56

Later, I tried using the HOG Features alone and Below figure shows the combination of Different HOG parameters and the accuracy I got.

Colorspace	Orient	pix_per_cell	cell_per_block	hog_channel	Accuracy	Time to train
RGB	9	8	2	2	0.9462	8.21
RGB	9	8	2	1	0.9465	8.45
RGB	9	8	2	0	0.9445	8.57
RGB	9	8	2	ALL	0.9648	21.02
HSV	9	8	2	2	0.9499	7.67
HSV	9	8	2	1	0.8916	21.4
HSV	9	8	2	0	0.9127	15.39
HSV	9	8	2	ALL	0.9766	18.35
LUV	9	8	2	0	0.9488	8.88
HSL	9	8	2	2	0.8939	25.28
HSL	9	8	2	1	0.9476	8.52
HSL	9	8	2	0	0.9023	16.14
HSL	9	8	2	ALL	0.9797	19.1
YUV	9	8	2	1	0.9164	14.06
YUV	9	8	2	0	0.9417	8.6
YCrCb	9	8	2	2	0.9015	18.11
YCrCb	9	8	2	1	0.9175	16.71
YCrCb	9	8	2	0	0.9502	8.14
YCrCb	9	8	2	ALL	0.982	16.69
YCrCb	8	8	2	ALL	0.9814	14.85
YCrCb	9	16	2	ALL	0.9823	2.09
RGB	9	16	2	ALL	0.969	2.98
HSV	9	16	2	ALL	0.978	1.77
HLS	9	16	2	ALL	0.9789	1.96
YCrCb	9	16	1	ALL	0.9673	3.69

I decided to combine spatially binned color and color histograms and HOG features. Below table summarizes the parameters used and the Accuracy I got.

ColorSpace	Spatial Size	Hist_bins	Orient	Pix_per_cell	Cell_per_Block	Hog_Channel	Accuracy	Traintime
RGB	(8,8)	8	9	8	2	ALL	98.03	17.65
HSV	(8,8)	8	9	8	2	ALL	98.51	14.99
HLS	(8,8)	8	9	8	2	ALL	98.82	14.90

ColorSpace	Spatial Size	Hist_bins	Orient	Pix_per_cell	Cell_per_Block	Hog_Channel	Accuracy	Traintime
YCrCb	(8,8)	8	9	8	2	ALL	98.68	16.07
RGB	(8,8)	8	8	8	2	ALL	96.48	21.02
HSV	(8,8)	8	8	8	2	ALL	98.23	16.69
HLS	(8,8)	8	8	8	2	ALL	98.88	15.39
YCrCb	(8,8)	8	8	8	2	ALL	98.68	16.07
HSV	(16,16)	16	9	8	2	ALL	99.01	4.74
HLS	(16,16)	16	9	8	2	ALL	98.82	14.64
YCrCb	(16,16)	16	9	8	2	ALL	99.01	16.07
YCrCb	(16,16)	16	8	8	2	ALL	99.35	3.05

So, I have decided to use the following parameters to train the classifier.

Parameter	Value
Colorspace	YCrCb
Spatial size	(16,16)
hist_bins	16
Orient	8
Pix_per_Cell	8
Cell_per_Block	2
hog_Channel	ALL

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).

The code can be found in code cell [9] of the Ipython notebook. I have extracted the features from car images and Non car images and created an array stack of feature vectors as shown below:

```
X = np.vstack((car_features, notcar_features)).astype(np.float64)
```

Our labels vector y in this case will just be a binary vector indicating whether each feature vector in our dataset corresponds to a car or non-car (1's for cars, 0's for non-cars). Given lists of car and non-car features (the output of extract_features()) we can define a labels vector like this:

```
y = np.hstack((np.ones(len(car_features)),np.zeros(len(notcar_features))))
```

Now I have splitted the data into training and testing sets using Scikit-Learn `train_test_split()` function as below :

```
# Split up data into randomized training and test sets
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=rand_state)
```

Now I have scaled the feature vectors to have zero mean and unit variance.It's important to do the scaling after splitting the data, otherwise you are allowing the scaler to peer into your test data!

```
from sklearn.preprocessing import StandardScaler
# Fit a per-column scaler only on the training data
X_scaler = StandardScaler().fit(X_train)
# Apply the scaler to both X_train and X_test
scaled_X_train = X_scaler.transform(X_train)
scaled_X_test = X_scaler.transform(X_test)
```

Now I have defined a classifer and trained it on Training dataset. I have used Linear SVM.

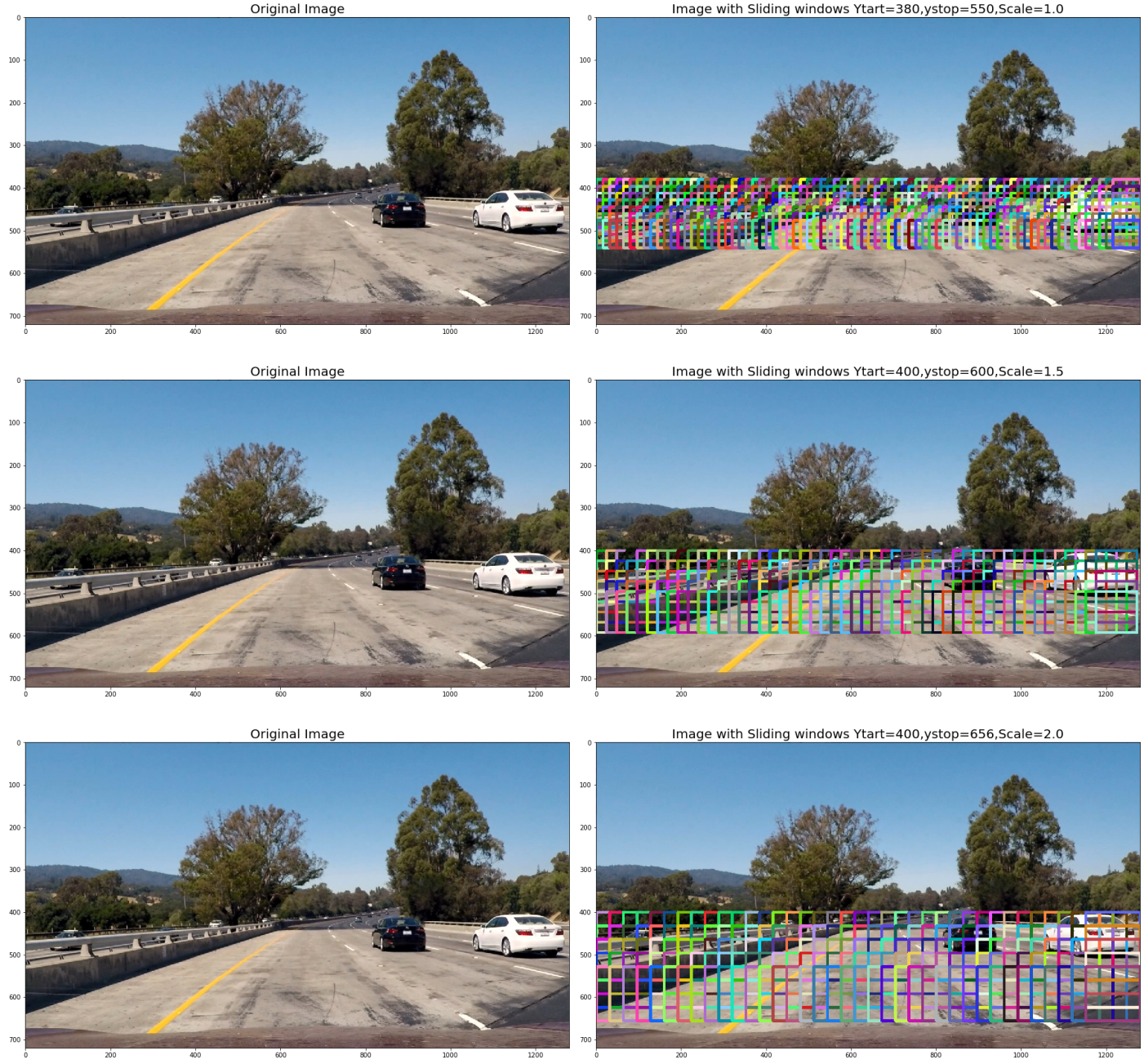
```
from sklearn.svm import LinearSVC
# Use a linear SVC (support vector classifier)
svc = LinearSVC()
# Train the SVC
svc.fit(scaled_X_train, y_train)
```

By using above mentioned parameters I have got an accuracy of around 99.35%.

Sliding Window Search

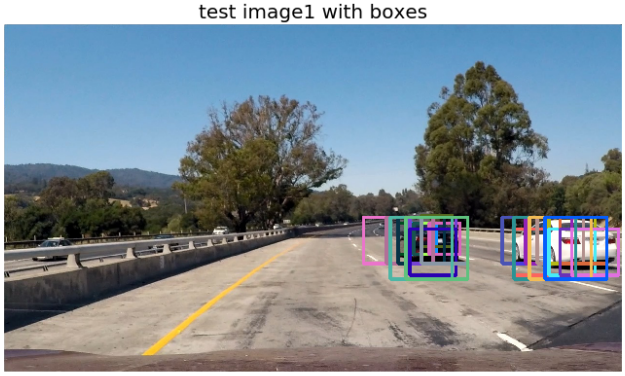
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I have adapted the find_Cars() function(code cell [10]) from the Udacity lesson which can extract features and make predictions. It extracts hog features only once for each of the predetemined window sizes(defined bt scale argument). Each window is defined by a scaling factor that impacts the window size. The scale factor can be set on different regions of the image (small near the horizon, larger in the center. For our example are using a 64 x 64 base window. If we define pixels per cell as 8 x 8, then a scale of 1 would retain a window that's 8 x 8 cells (8 cells to cover 64 pixels in either direction). An overlap of each window can be defined in terms of the cell distance, using cells_per_step. This means that a cells_per_step = 2 would result in a search window overlap of 75% (2 is 25% of 8, so we move 25% each time, leaving 75% overlap with the previous window). Any value of scale that is larger or smaller than one will scale the base image accordingly, resulting in corresponding change in the number of cells per window. Its possible to run this same function multiple times for different scale values to generate multiple-scaled search windows. (This is taken from Udacity lesson Hog Sub-sampling Window search) Below images shows search windows with different scales.



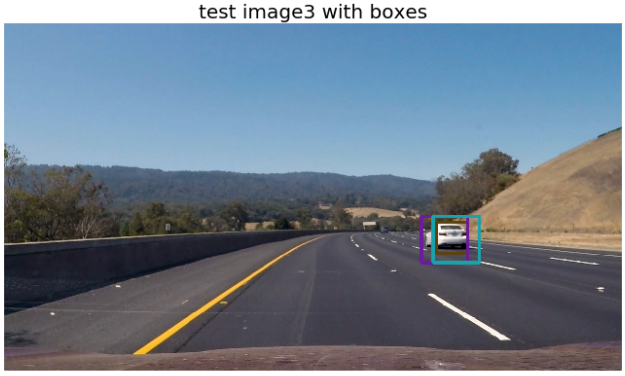
The below image shows the rectangles returned by the find_cars() function after combining the above three scaled windows.

test image1 with boxes

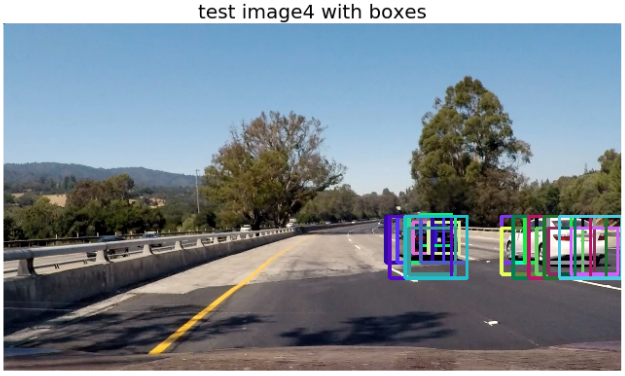


test image2 with boxes

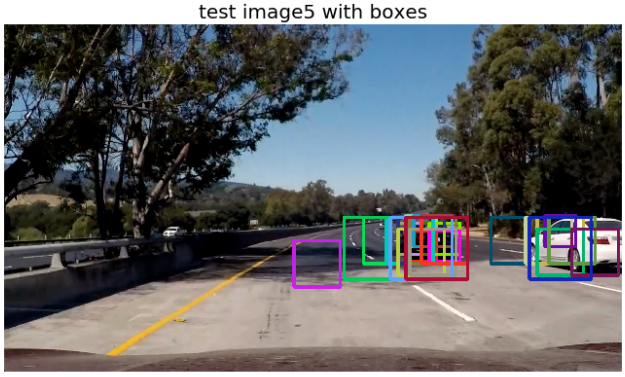
test image3 with boxes



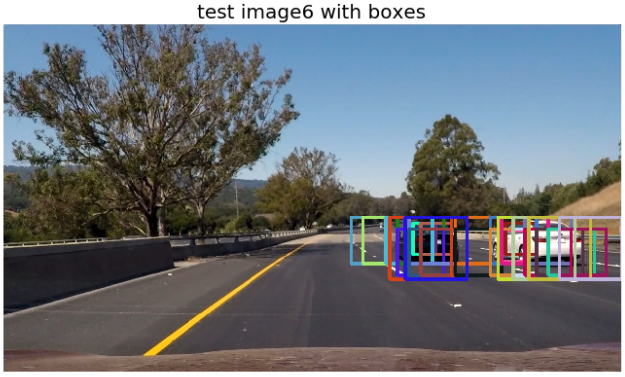
test image4 with boxes



test image5 with boxes



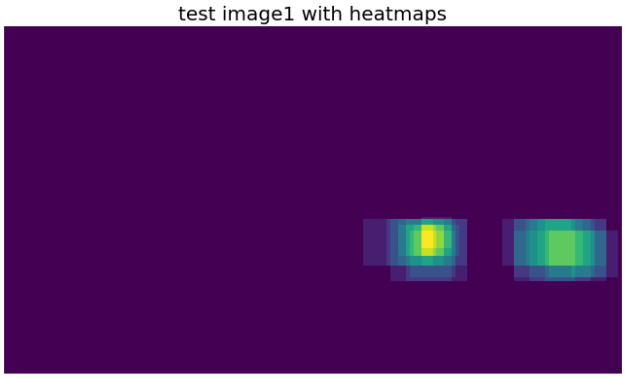
test image6 with boxes

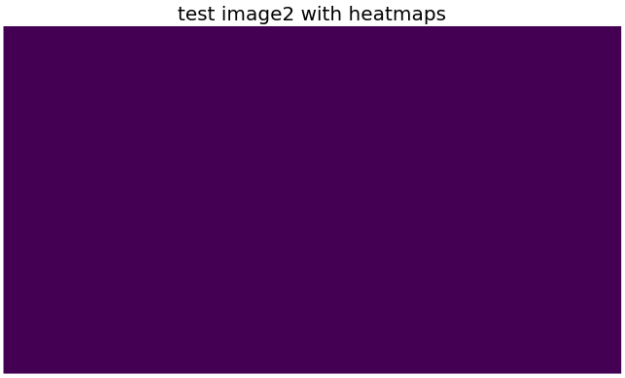


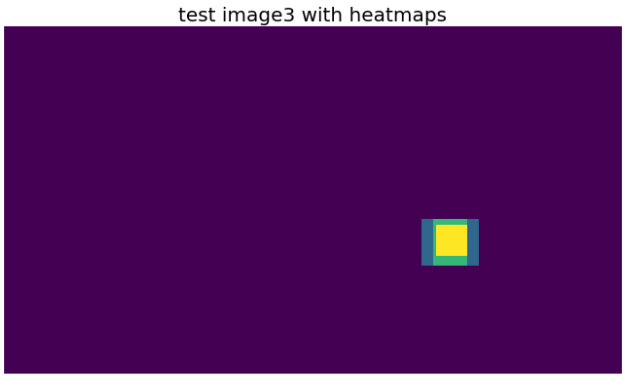
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

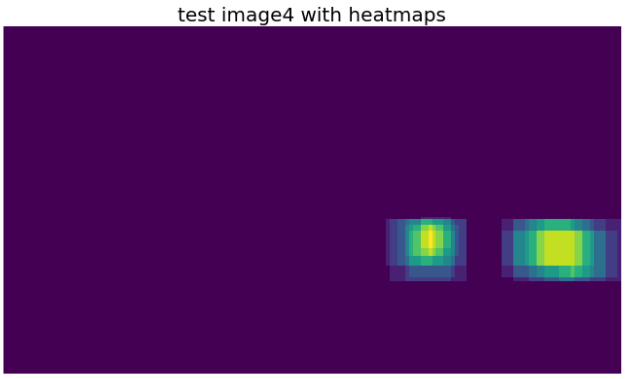
The code can be found in cell [11] I recorded the positions of positive detections. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

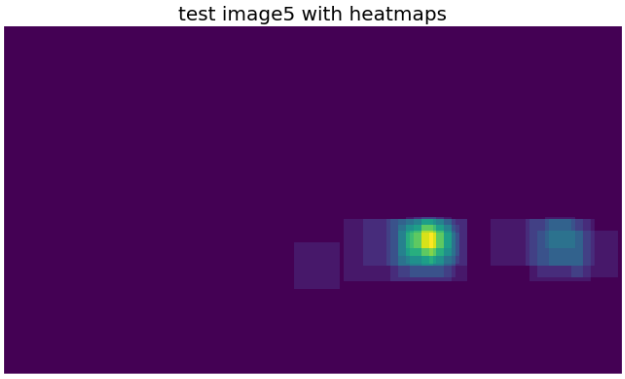
Here are six frames and their corresponding heatmaps:

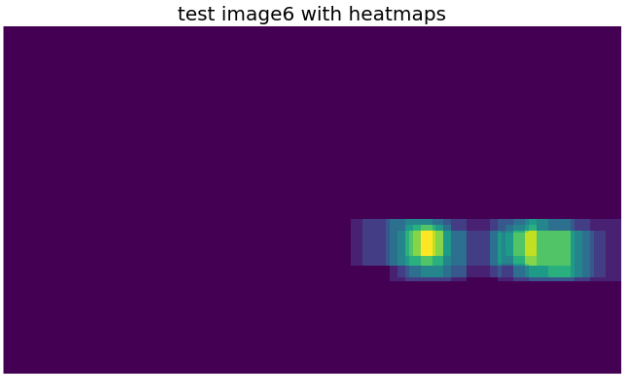




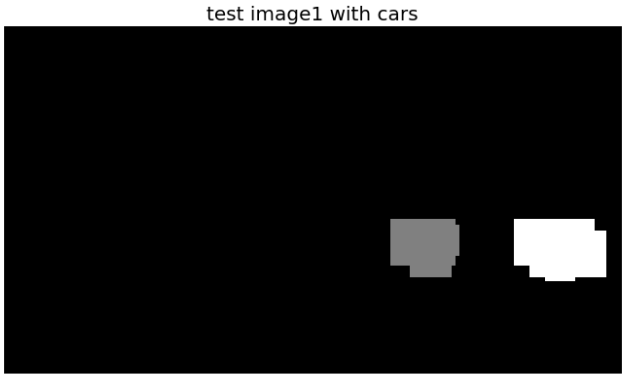


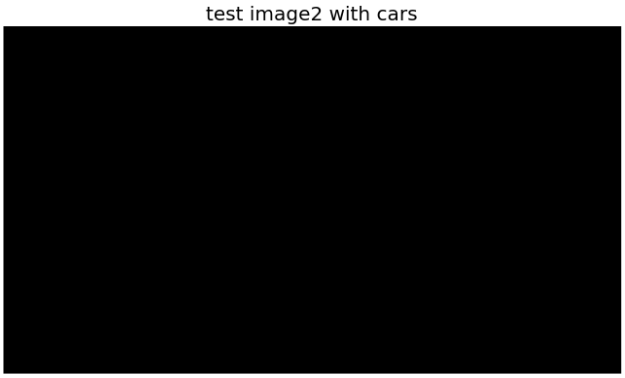


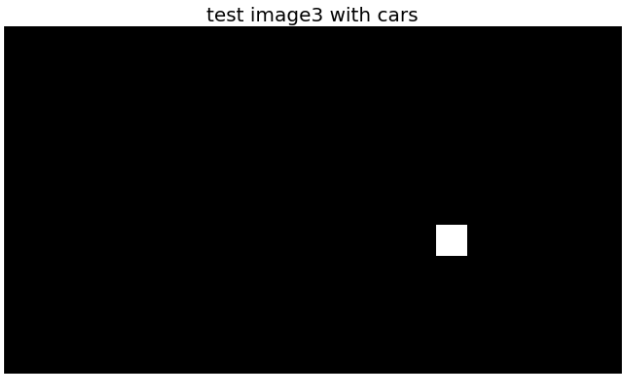


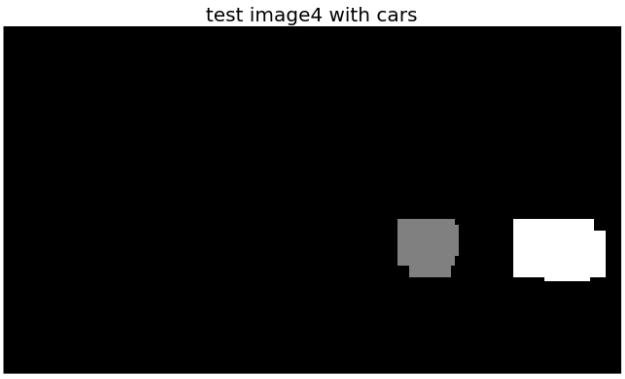


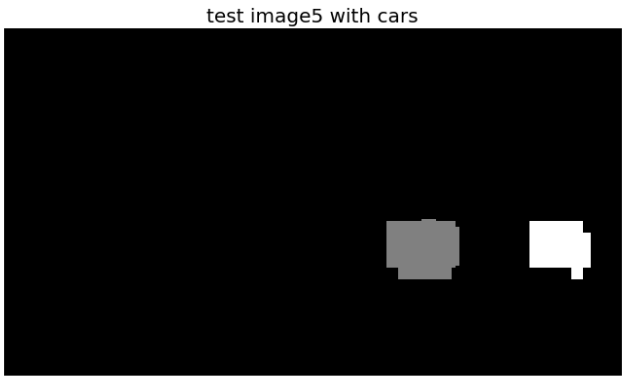
Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:

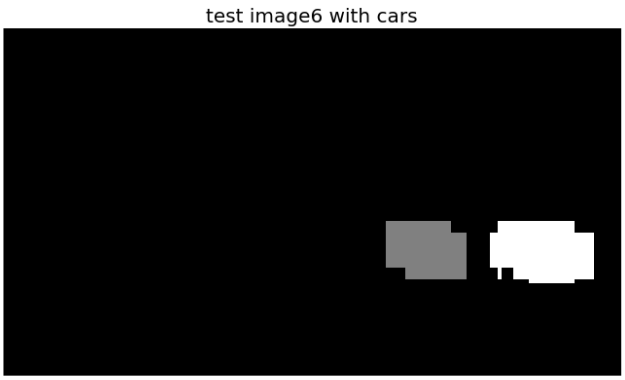




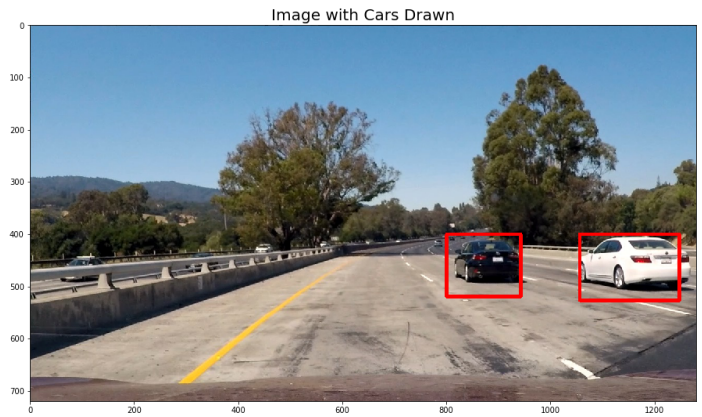








Here the resulting bounding boxes are drawn onto the last frame in the series:



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?

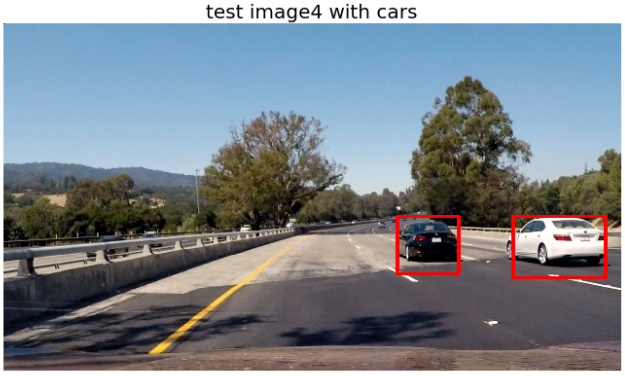
Ultimately I searched on three scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:

test image1 with cars

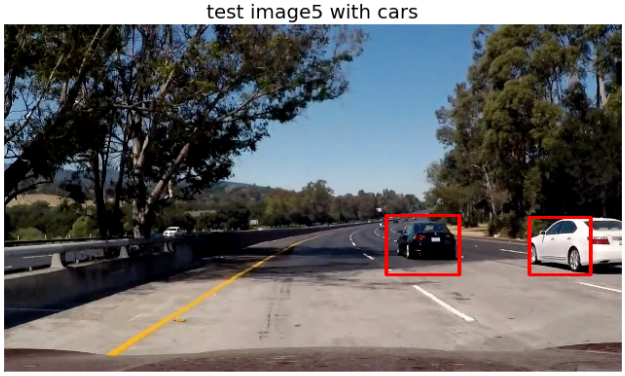
test image2 with cars

test image3 with cars

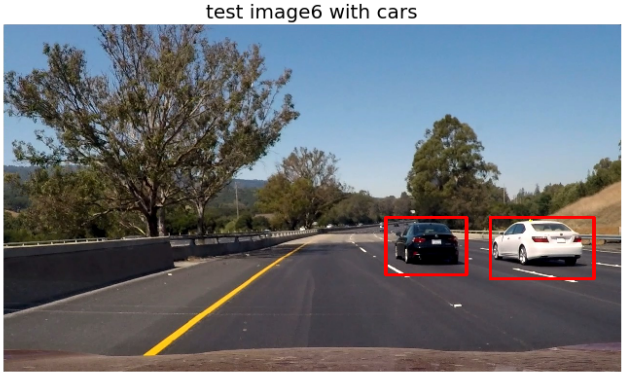
test image4 with cars



test image5 with cars



test image6 with cars



Video Implementation

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 (./project_video_output.mp4)

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?

For this project I have spent a lot of time on selecting the best combination of HOG parameters. After selecting the best combination I have worked on selecting different scales and overlapping values for windows. There are lot of false positives detected on the left side of the car where shadows and different lighting conditions other than the ones from training set. This issue can be resolved by adding more dark images to the training data set. I