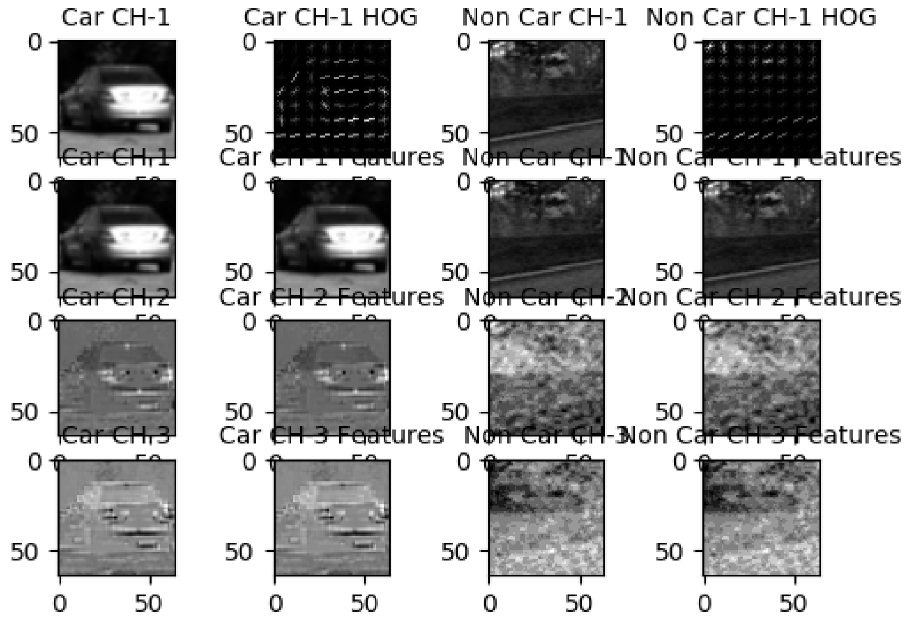
Vehicle Detection And Tracking

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 is a template writeup for this project you can use as a guide and a starting point.**
   This writeup will describe the techniques used to perform vehicle detection on a series of test images and two videos. A total of 17760 samples of feature size 8460 were used for training.

2. **Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.**
   HOG (Histogram of Gradients) was utilized as one of the feature sets used to generate the complete feature vector for each training image. The training images used were provided by Udacity. The function to perform feature extractions is called extract_features, which, in turn, calls the function get_hog_features to extract HOG features, and is present in the file lesson_functions.py. HOG features specifically were extracted using the skimage.feature.hog function, the following parameters. These parameters were selected as they were used in the course, and worked well directly.

Parameter Name	Parameter Value
orientations	9
pixels_per_cell	(8, 8)
cells_per_block	(2, 2)

HOG features are useful in obtaining features for classes of objects that may vary significantly in color, but not very significantly in shape. It essentially divides up an image into blocks of four 8X8 cells each, and computes a histogram of the gradient values in each block. As seen below, car and non car images are very different, and can be easily used by a classifier. The YCrCb olorspace has been used here.

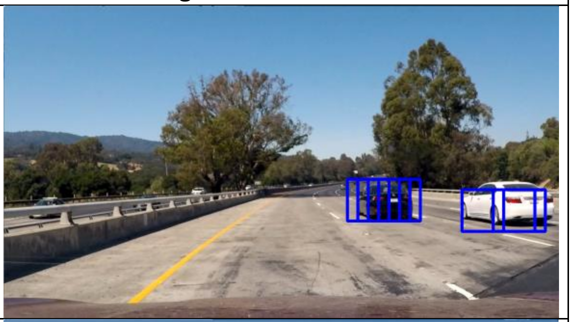
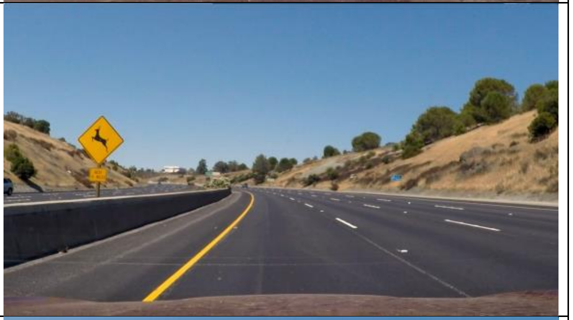
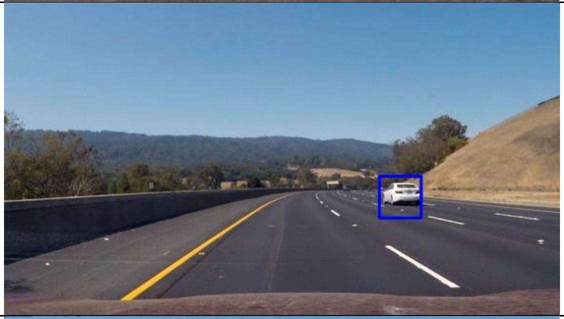
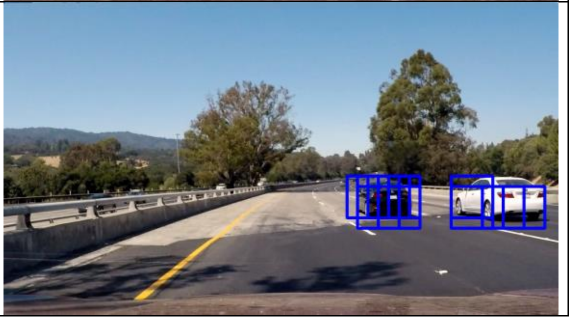


3. **Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).**
   The classifier used was the standard Support Vector Classifier (SVC) available in sklearn.svm. This step is covered in lines 82 to 145 of main.py. The features used were a combination of HOG features, spatial color features, and color histogram features. The kernel used was 'rbf', which is the default kernel, and a cache size of 600 MB was used as I was running into memory related issues during the training. The parameter C was set to 1.0. 35% of the total test set was used as the validation set, and an accuracy of 99.646%. The output was as follows:

```
EXTRACTING CAR FEATURES...
DONE, TOOK 50.91111373901367 SECONDS
EXTRACTING NON CAR FEATURES...
DONE, TOOK 51.661510705947876 SECONDS
SCALING ALL FEATURE VECTORS...
FEATURES: (17760, 8460)
DONE, TOOK 6.141252040863037 SECONDS
NOW TRAINING SUPPORT VECTOR CLASSIFIER...
..*.*
optimization finished, #iter = 3591
obj = -421.314172, rho = -0.372389
nSV = 2153, nBSV = 190
Total nSV = 2153
[LibSVM]DONE, TOOK 269.88582372665405 SECONDS
CHECKING CLASSIFIER ACCURACY...
Test Accuracy of SVC =  0.996460746461
DONE, TOOK 142.3891327381134 SECONDS
```

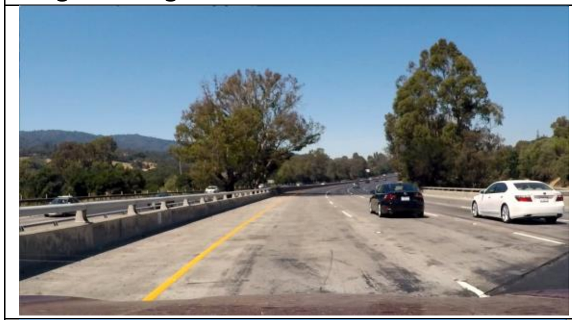
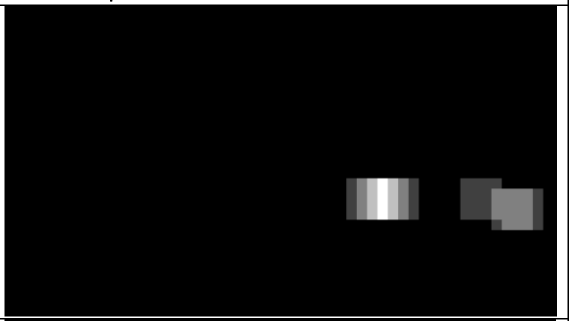
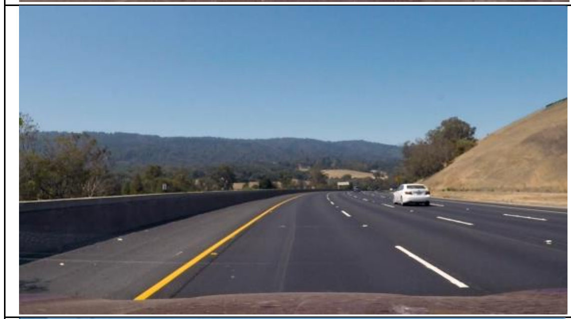
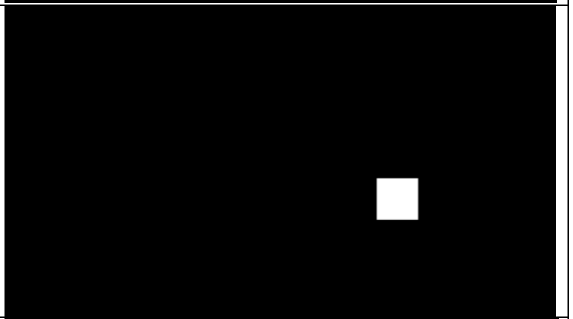
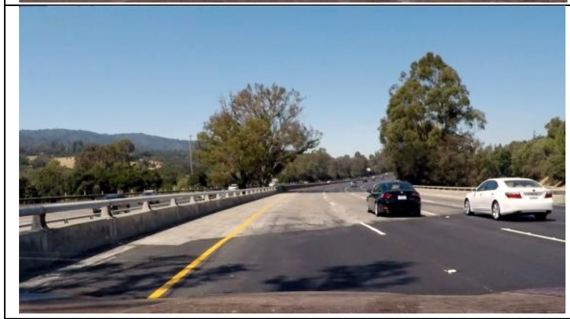
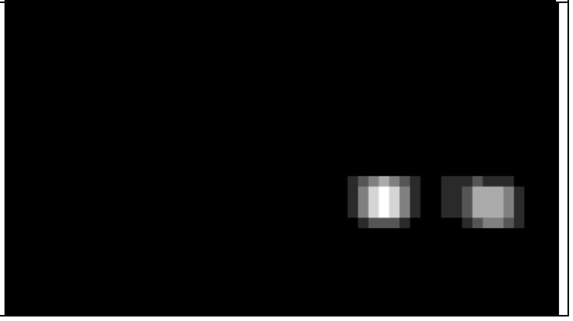
4. **Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?**
   The sliding window function is called find_cars, implemented in lesson_functions.py and utilized in main.py. It has been adapted from the function provided by Udacity. The function analyzes only the lower portion of the image as that is the only portion which contains relevant information (with the possibility of vehicles appearing). It utilizes a scale of 1.5 and a cell step size of 2 cells while sliding the window. Multiple scales have not been utilized. However, this function computes the HOG features only once per frame.

5. **Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?**
   In order to optimize the performance of the classifier, I tried experimenting with different color spaces, and scale factors. The parameters chosen seem to work pretty well at this point.

Original Image	Result of sliding window search
	
	
	
	

6. **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 is located here. Every 15th frame is run through the classifier, as it did not seem necessary to run every video frame through it, given that the video itself was at 25 frames per second. The results look quite reliable.

7. **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 order to deal with overlapping bounding boxes, the 'heat map' technique described in the Udacity course was chosen. It involves the identication of all pixels inside a bounding box, with more weight given to pixels which reside in multiple bounding boxes. Some examples are given below. The code is in lines 257 to 263 of main.py. A filter for false positives has also been implemented, but its use was not necessary in this project so far. This filter is function apply_heatmap_threshold, which would allow the algorithm to ignore the below a certain threshold value

Original Image	Heat Map
	
	
	

8. **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?**
   This project was relatively straightforward to implement. The greatest challenge was the understanding of the HOG subsampling window search. One thing that could be done to make it faster in its analysis is to better utilize the history of detected boxes to allow searching smaller neighbourhoods for a particular vehicle, once it is identified. Currently the algorithm runs slower than desirable (around 13 seconds per frame). This problem was mitigated by downsampling, but that may not be necessary if the history of detected vehicles could be better utilized.