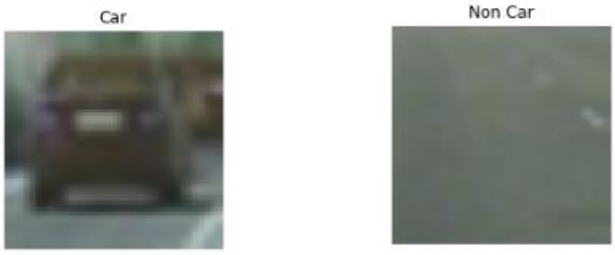
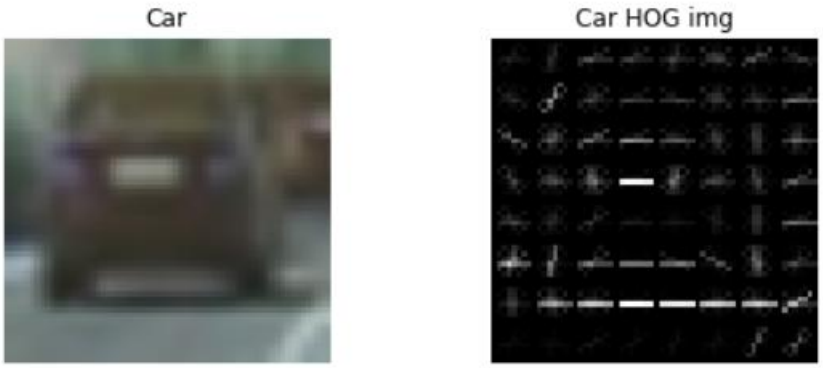
Vehicle Detection project writeup

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images:
The code this step is contained in the 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:



I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block).
The code for extracting HOG features from an image is defined by the method get_hog_features in cell #2. The figure below shows a comparison of a car image and its associated histogram of oriented gradients, as well as the same for a non-car image.



The method extract_features in cell #4 takes a list of images and produces the HOG parameters (bin spatial and color hist are also options) and flatten HOG features of each image. This is the key step for training SVM classifier

2. Explain how you settled on your final choice of HOG parameters:
I chose the HOG parameters from the lectures first and then tune them for the best performance of the classifier prediction accuracy. There is a balance between accuracy and speed of classifier. In this project, I choose accuracy as my priority and I

increase the number of orientation, pixel per cell and hog channels.
The parameters I chose are listed as followings:

colorspace = 'YCrCb'
orient = 8
pix_per_cell = 8
cell_per_block = 2
hog_channel = 0

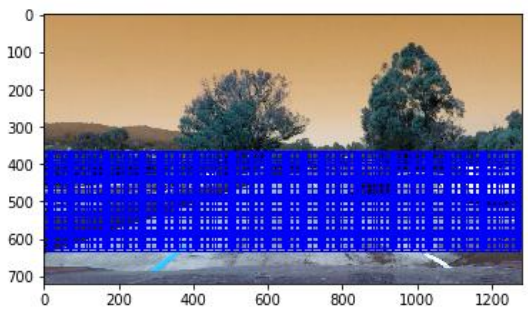
In addition to HOG features, I also apply Color classifier (Spatial Binning of Color & Gradient Features) to train the SVM. Sptial size of (16, 16) and Histogram bin size of 32 are chosen.

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 I trained my classifier is in cell #6. I extracted all of the examples images (8792 for car images and 8968 for non car images) and chose the HOG parameters as it were discussed. I created an array storing features vectors for car and non-car images as my training and testing samples:
X = np.vstack((car_features, notcar_features)).astype(np.float64)
Then I defined labels vector:
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
I also split the training and testing samples by 8:2 and chose a linear SVM classifier. The accuracy of the classifier I got is 99.42%!

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?

The code for sliding windows is in cell #8.
I chose 3 scales of windows to slide through the images: (96,96), (128, 128), (160, 160).
(64, 64) is used for detecting far vehicles. (160, 160) is used for detecting close vehicles and (128, 128) is used for detecting middle close and middle far vehicles. Overlap is chosen as 0.85.
A examples of the sliding window is shown below



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 search on YCrCb space with all 1 channel HOG features. I also use binned color and histograms of color to search the vehicles.
The pictures below shows the output of searching vehicles:

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.)

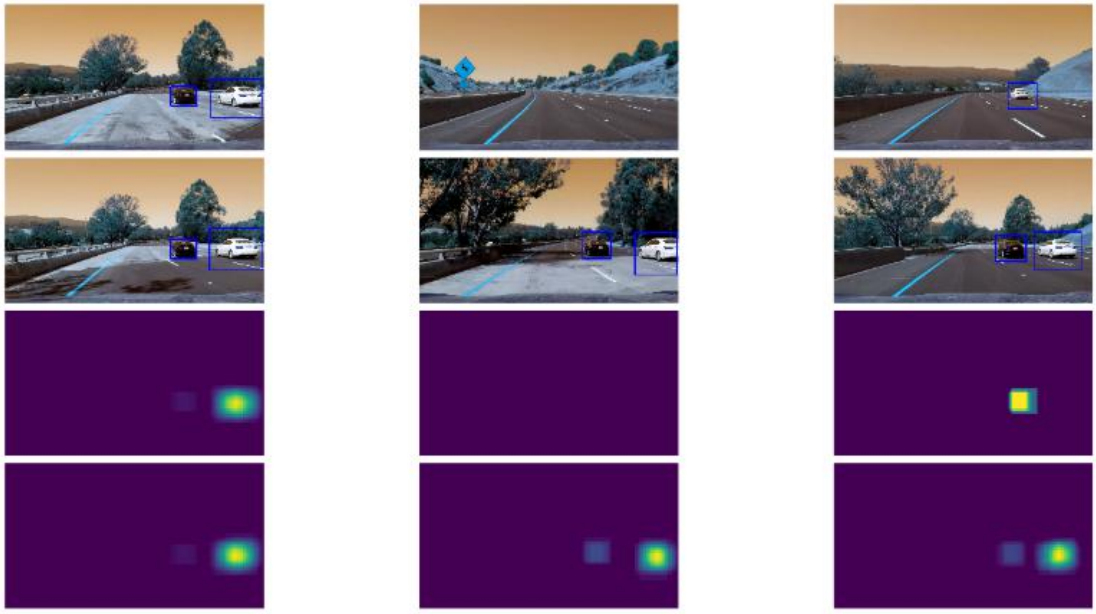
Video output would be in the attached zip file I uploaded.

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.

2.1 Filter of False positives

First, I apply a search windows (Cell 26) on the images and I restrict the search region to be on the Right side and middle portion of the images by defining the *x_start_stop* and *y_start_stop as:*
x_start_stop1 = [800, None]
x_start_stop2 = [400, 1200]
y_start_stop1 = [350, 650]
y_start_stop2 = [350, 500]
Restricting the search region could not only filter out the false positives but also reduce the processing time.
I recorded the positions from the windows detecting the vehicles in search_window function(Code cell #10). Then I add “heat” to the position where the vehicles are detected by simply adding 1 to the array element.
After the heat is added to the image (Cell 27-28), I apply a threshold as 1 to filter out the area

which is not hot. Then I convert those filtered hot areas with labels draw the labels as boxes on the image.
Samples are shown below:



2.2 Smoothing boxes in Frames
In order to smooth the boxes in the video. I average the heat map on every three frames by using a heat_list to store the heatmap detected on each frame and deleted the oldest in the list once the length of the list is larger than 4. Detailed codes can be found in cell #31.

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 problem I faced in my project is balancing the accuracy of classifier and the execution speed. I believed that the accuracy should be higher if I used binned color and color histogram also as my features to train the classifier but the execution time and memory would be the obstacles.
The pipeline would most likely to fail in cases where vehicles don't resemble those in the training dataset, but lighting and environmental conditions might also play a role. Distant car or shadow would be a issue.
I believe that a more accurate classifier should be used. We can determine the vehicle location and speed to predict its location in subsequent frames or begin with expected vehicles in the nearest areas and preclude overlap and redundant detection from smalled scale search areas to speed up execution