Please note that my code for the project has been split between two Jupyter notebooks – "Process Data and Train Classifier.ipynb" and "Pipeline.ipynb". Additionally, I made use of the functions provided by the class by placing them in a file ("lesson_functions.py") and importing that file into the notebooks.

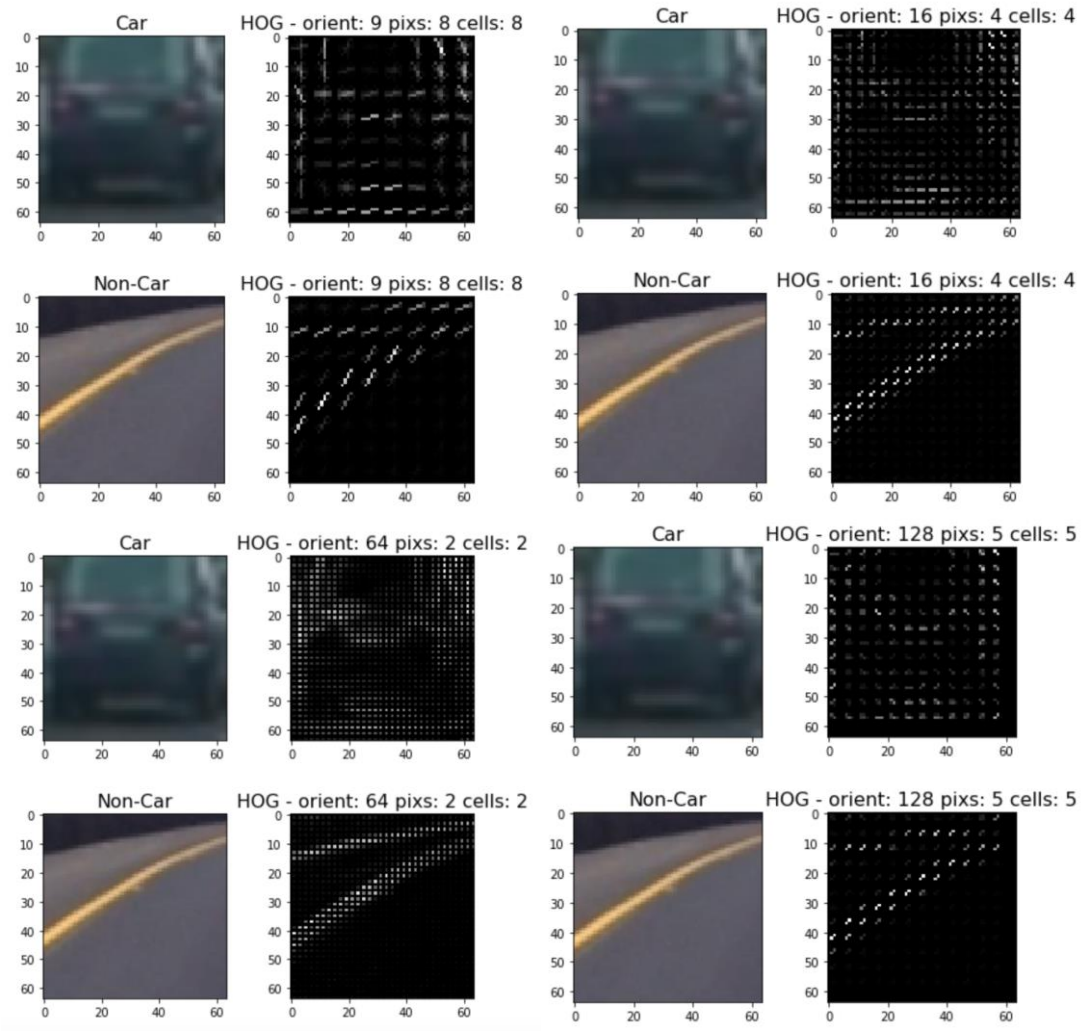
Histogram of Oriented Gradients (HOG)

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

In the Process and Train notebook I set the parameters for the feature extraction, then load the car and non-car images, and call the "extract_features" function provided by the class in the "Extract Features for Cars and Non-Cars Using Class-Provided Methods and Passing in Appropriate Parameters" section of the notebook.

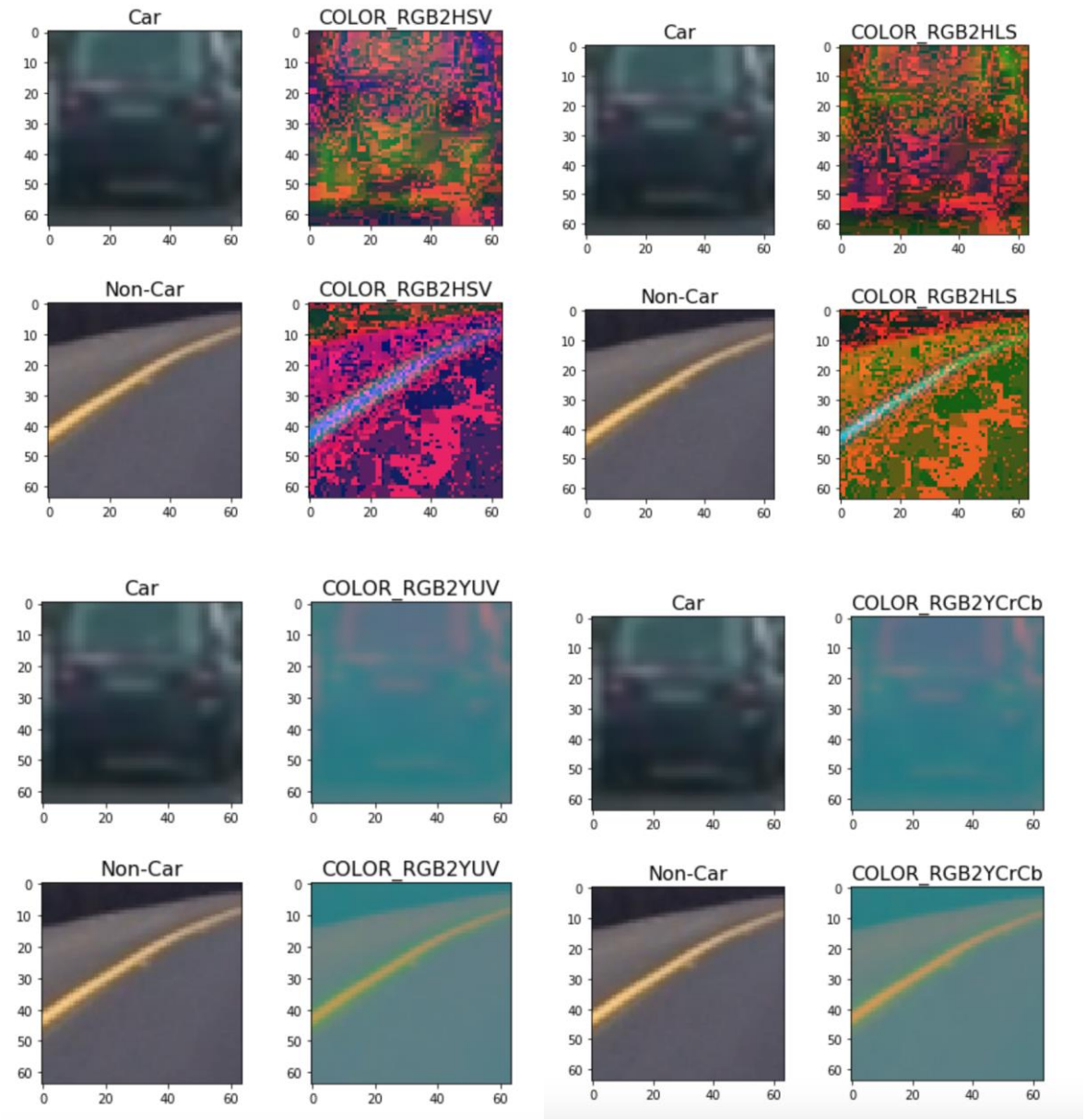
2. Explain how you settled on your final choice of HOG parameters.

I ran through a number of combinations of orient/pixels per cell/cells per block to get an understanding of their impact on the images that we are working with. I selected 9 orients, 16 pixels per cell, and 2 cells per box based on discussion in the forums and my own experimentation. This combination provided a good "middle ground" between "too much" information and "too little" as the goal of using HOG is to make the images a bit more "abstract" so that the classifier isn't trying to match to an exact image which would result in a large number of vehicles being missed (see comparisons below).



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).

In the "Extract Features for Cars and Non-Cars Using Class-Provided Methods and Passing in Appropriate Parameters" section of the Process Data and Train Classifier notebook I am performing the features extraction and it is on line 3 that I am setting the parameter for the call to the execute_features function and passing in "HLS" as the color space to use. HLS was selected based on what was presented for previous projects colors seem to be less relevant for what we are doing here and HLS seemed to provide better distinction between objects in our images (see comparisons below).



In the "Training the Classifier" section of the "Process Data and Train Classifier" notebook I am, after splitting and standardizing the data, training the classifier. As can be seen in the notebook I compared a number of classifiers to determine the right balance between performance and accuracy. Based on this I selected the Support Vector Classifier; while it did take considerable longer than the others to perform a prediction, it provided the highest 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?

In the section "Combining Sliding Window with Trained Model" of the "Pipeline" notebook is where I perform the sliding window search on images. Through online discussions and my own experimentation I decided to use a number of window sizes and percentage of overlap and x/y stop/start coordinates to address the possibility of vehicles in different parts of the image appearing at a different size. Those parameters are (the first value is the window size):

[

(64, {"xy_overlap": (0.75, 0.75), "x_start_stop": (120, 1280-120), "y_start_stop": (375, 500)}),

(70, {"xy_overlap": (0.75, 0.75), "x_start_stop": (60, 1280-60), "y_start_stop": (375, 500)}),

(90, {"xy_overlap": (0.75, 0.75), "x_start_stop": (0, 1280), "y_start_stop": (375, 560)}),

(115, {"xy_overlap": (0.5, 0.5), "x_start_stop": (0, 1280), "y_start_stop": (375, 600)}),

(154, {"xy_overlap": (0.5, 0.5), "x_start_stop": (0, 1280), "y_start_stop": (400, 680)}),

(185, {"xy_overlap": (0.5, 0.5), "x_start_stop": (0, 1280), "y_start_stop": (450, 680)}),

(218, {"xy_overlap": (0.5, 0.5), "x_start_stop": (0, 1280), "y_start_stop": (450, 680)}),

]

This produces a large number of "hot" boxes – or boxes that have a high probability of containing vehicle; unfortunately, it also produces a large number of "false positives" – or boxes that don't contain vehicles.

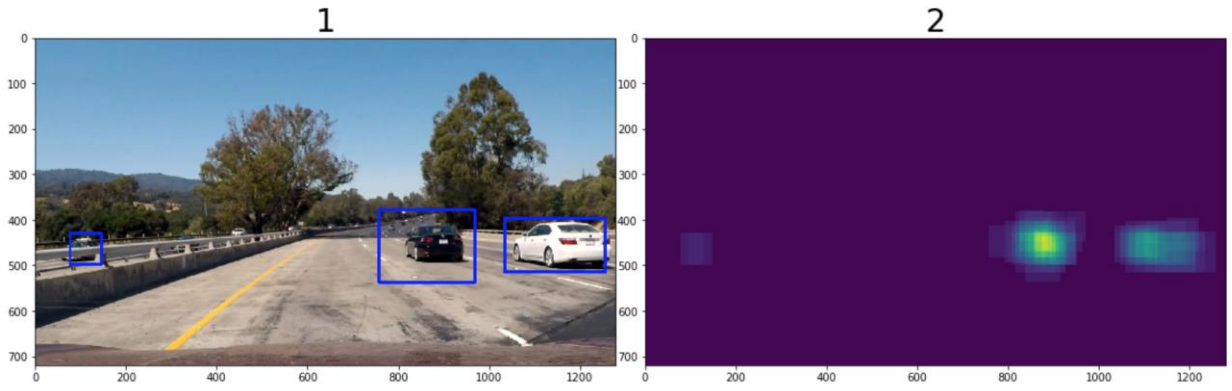
Video Implementation

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.)

Here's a link to my video result

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.

To address this I used heat maps (found in section "Code for Heatmaps" in the "Pipeline" notebook) to be able to identify those boxes that have a significant amount of overlap indicating a higher probability that these boxes have correctly identified a vehicle.



By using an appropriate threshold those "false positives" effectively get removed.

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 faced a huge number of challenges with this project. One of the biggest was with the sliding window and how to deal with the huge number of images from the video. If you perform the operations on every image you will likely get a higher number of vehicles being correctly identified; however, performance will be degraded. If you "skip" images you run the likelihood of missing a vehicle if it moves a significant amount between the frames that you classify on.

This pipeline will likely fail under anything but optimal lighting conditions and will only work for the relatively small number of vehicles on the dataset. First, a much larger dataset will undoubtedly help; additionally, I think that using a deep learning convolutional neural network potentially work much better than that standard ML classifier as doing so should eliminate the need to use sliding windows of various sizes. You could potentially use a few windows on an image to determine the location of vehicles – reducing the overhead of the high number of classifications that need to be performed on an image.