Vehicle Detection Project

The goals / steps of this project are the following:

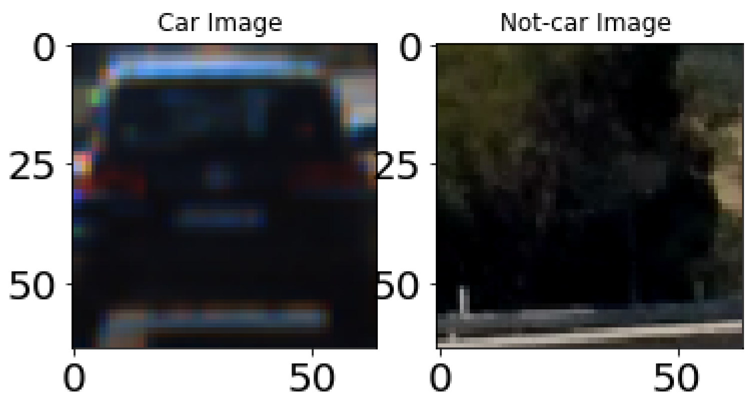
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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 Points

Histogram of Oriented Gradients (HOG)

1. **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.**

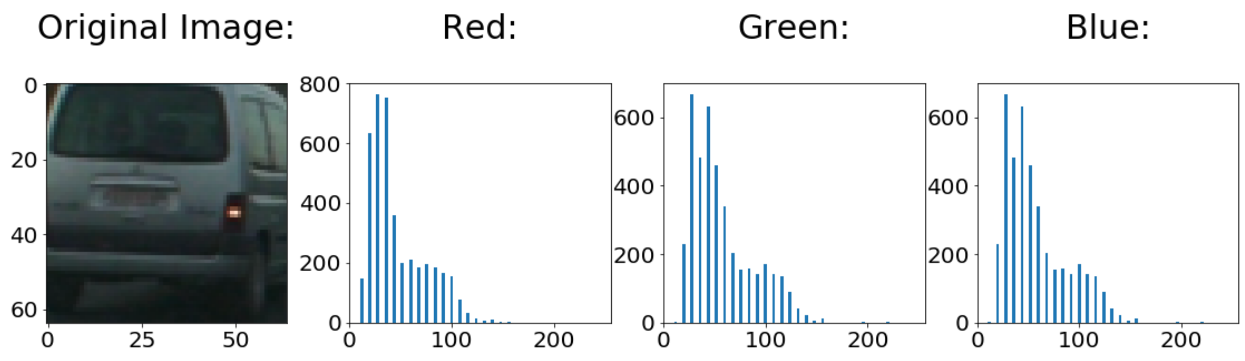
The code for this step is contained in the code cell 11 of the IPython notebook vehicle_detection.py. I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:



Feature Extraction

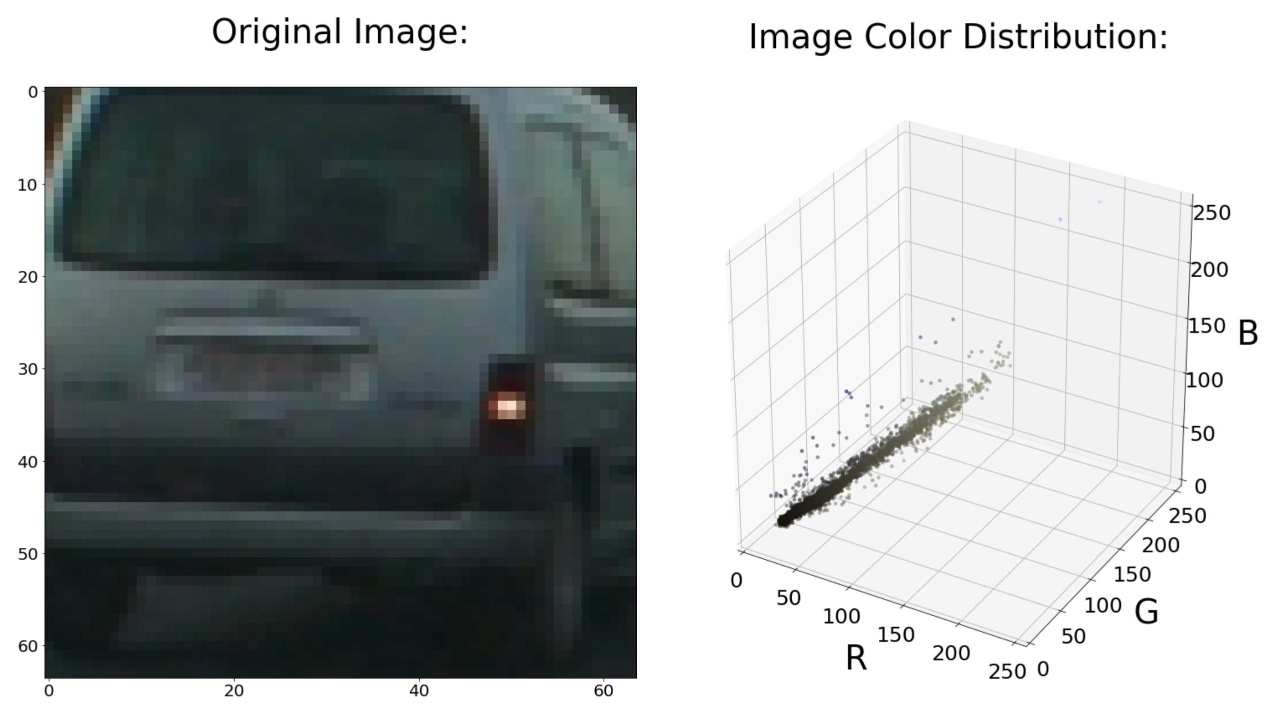
I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Histograms of Color:



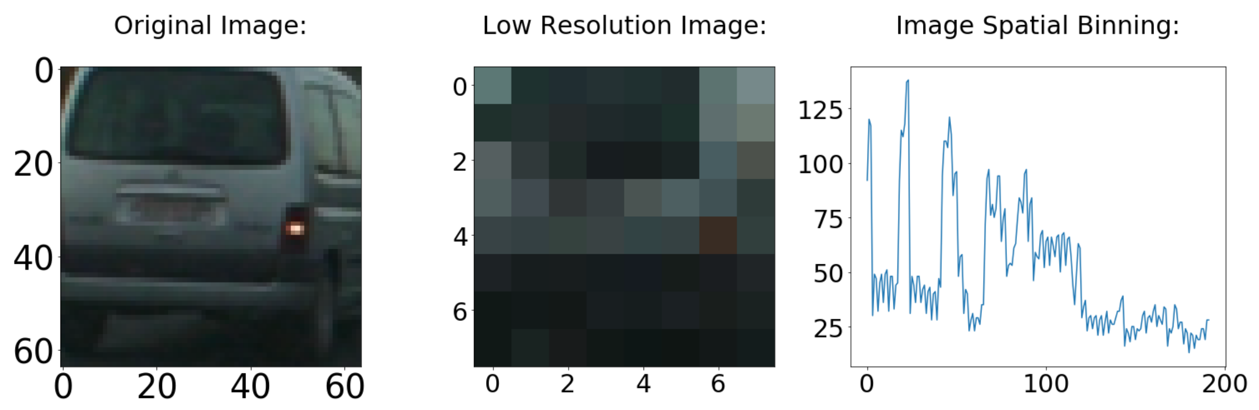
Several different techniques for feature extraction were used in Section 2 of this project, including histograms of color, color distribution, spatial binning, gradient magnitude, and Histogram of Oriented Gradients (HOG). Each has its own effect on the feature vector that is produced, and when combined the techniques tend to improve the chosen classifier's performance.

Image color distribution:



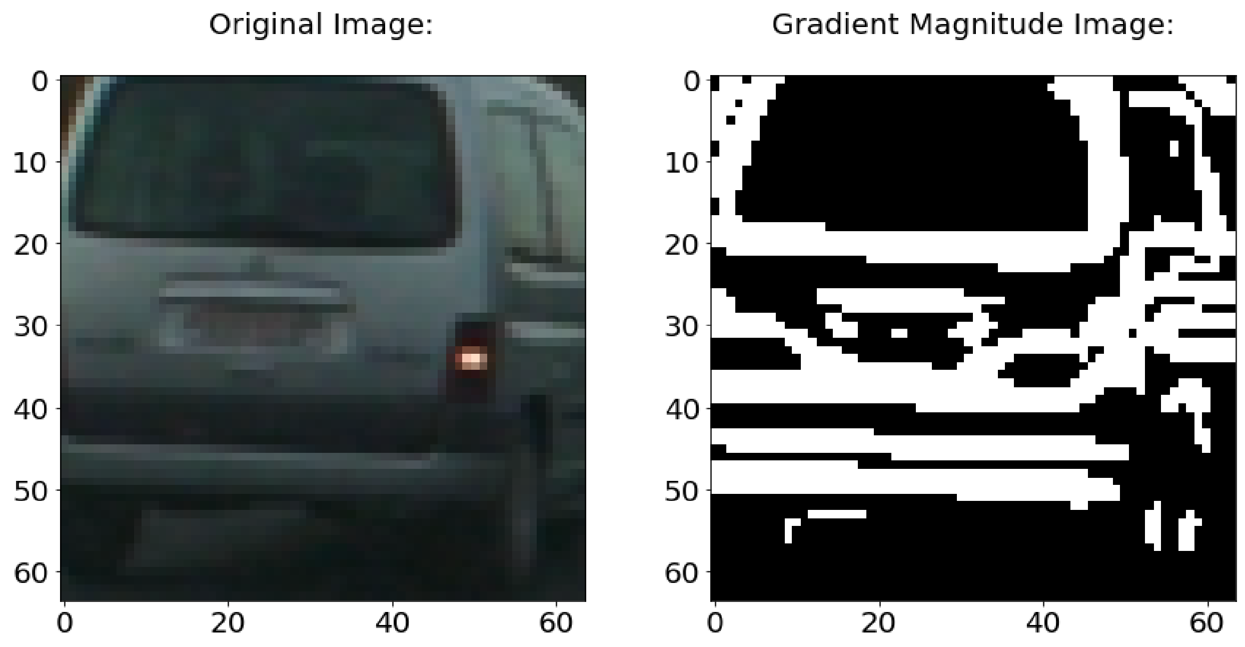
Spatial Binning

Raw pixel values are useful to include in the feature vector when the desired objects in an image remain identifiable at low image resolutions, such as a car object.



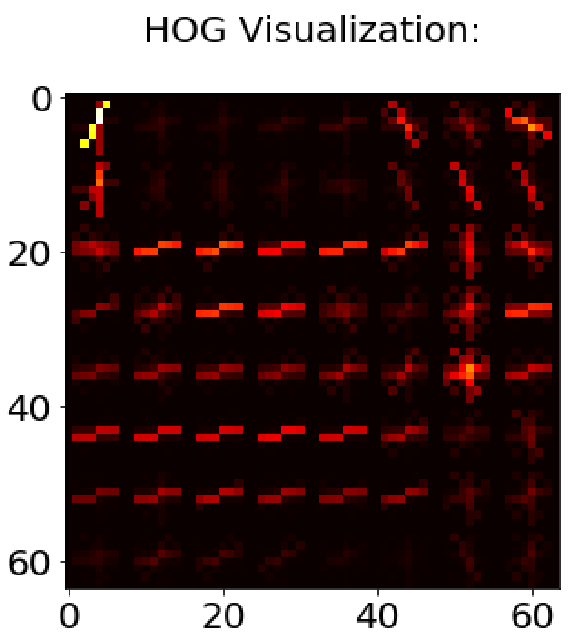
Gradient Magnitude

Gradient magnitude is a technique used in previous computer vision projects (Projects 1 & 4) that applies a filter that represents the magnitude of the sobel-x and sobel-y gradients of odd-numbered pixel squares, such as 3x3 or 5x5.



Histogram of Oriented Gradients (HOG)

HOG feature extraction is the most important technique utilized in this project. The scikit-image package has a built in function to handle HOG extraction, which is tuned by parameters including orientations, pixels_per_cell, and cells_per_block.



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

I tried various combinations of parameters and the final feature extraction method that was implemented includes color histograms, spatial binning, and HOG, as shown in Sections 2 & 3. For HOG, the parameters were chosen as follows:

Parameter	Value
orientations	9
pixels_per_cell	(16,16)
cells_per_block	(4,4)
visualise	True

feature_vector	False

I chose these parameters for HOG after trial and error on test4.jpg. As shown in the visualization above, the parameters optimize the gradients, and work well for the heat map step later in the pipeline that limits the false positive classifications.

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).

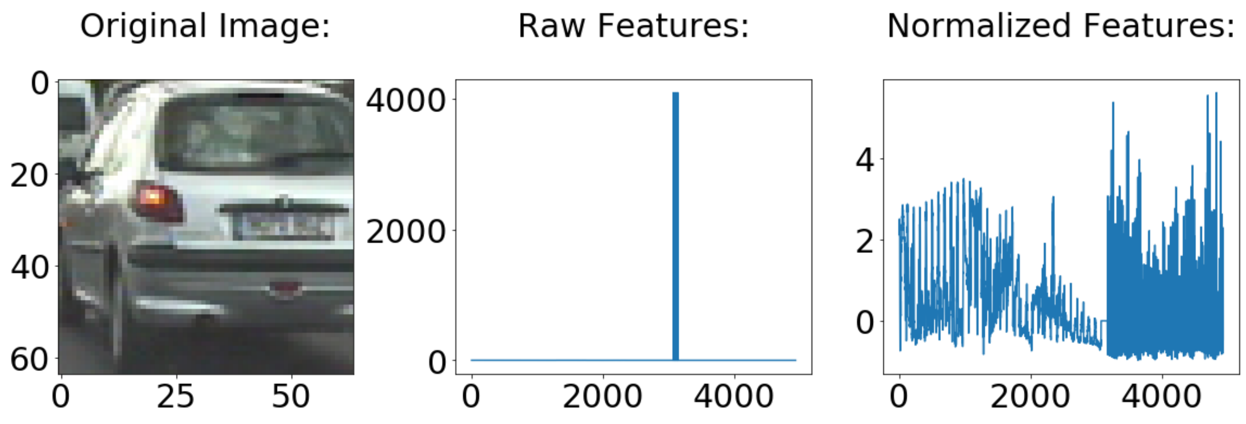
Preprocessing Data

The training data was normalized, randomized, and split into training and testing sets, with 80:20, 20% of data for testing.

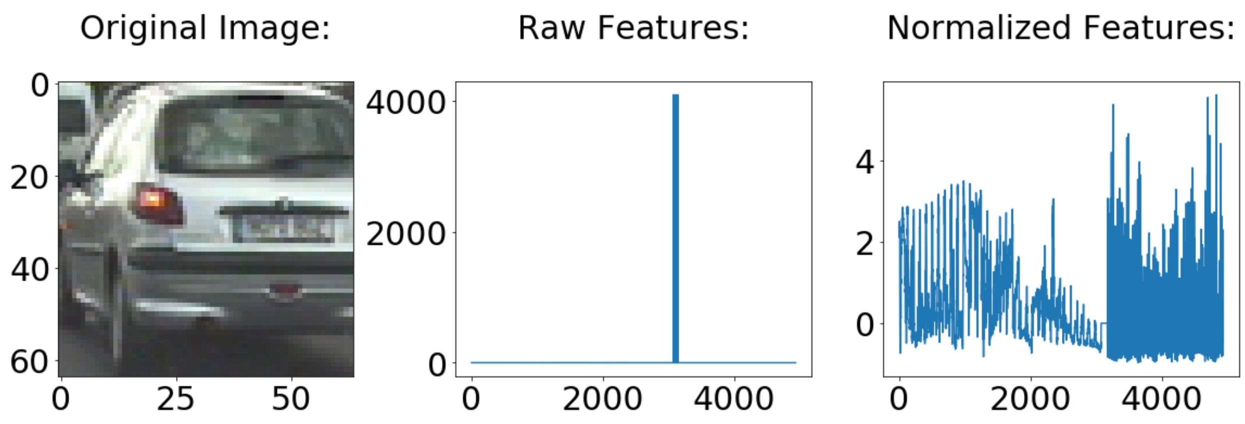
Training the SVC Classifier

I trained a SVC classifier to classify the dataset as "car" or "not-car" , accuracy for this classifier was above 94%.

Classification by Color and Spatial Binning



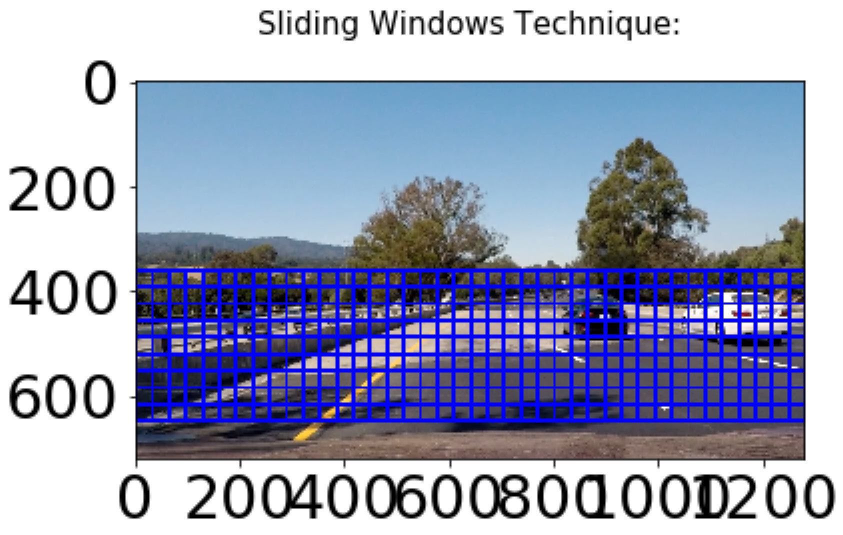
Classification by HOG



After comparing these two results, I decided to incorporate both of them as one feature vector that I retrained prior to running sliding windows on the test image, as explained next.

Sliding Window Search

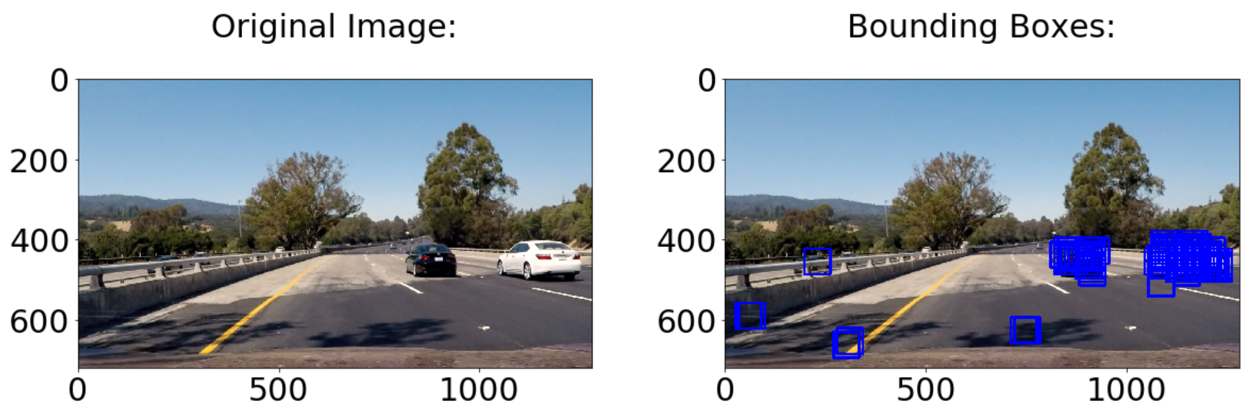
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?



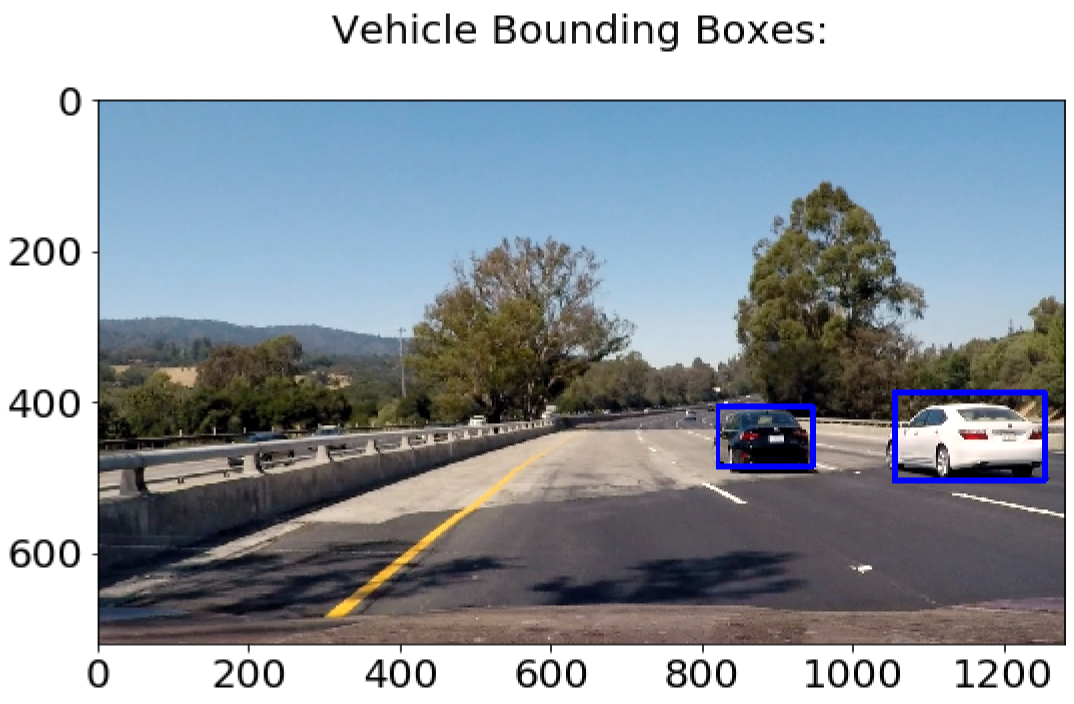
I limited the search area to the bottom half of the image, minus the bottom 10%, for a total of 40% of the image to lower the total number of windows I needed to make. The parameters I chose for the sliding windows themselves are as follows:

Parameter	**Value**
y_start	image width x 0.5
y_stop	image width x 0.9
xy_window	(64,64)
xy_overlap	(0.85, 0.85)

I constructed bounding boxes to cover the area of each blob detected. After detecting the correct amount of cars, I was able to combine the positive bounding boxes into one bounding box for each car:



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?



Video Implementation

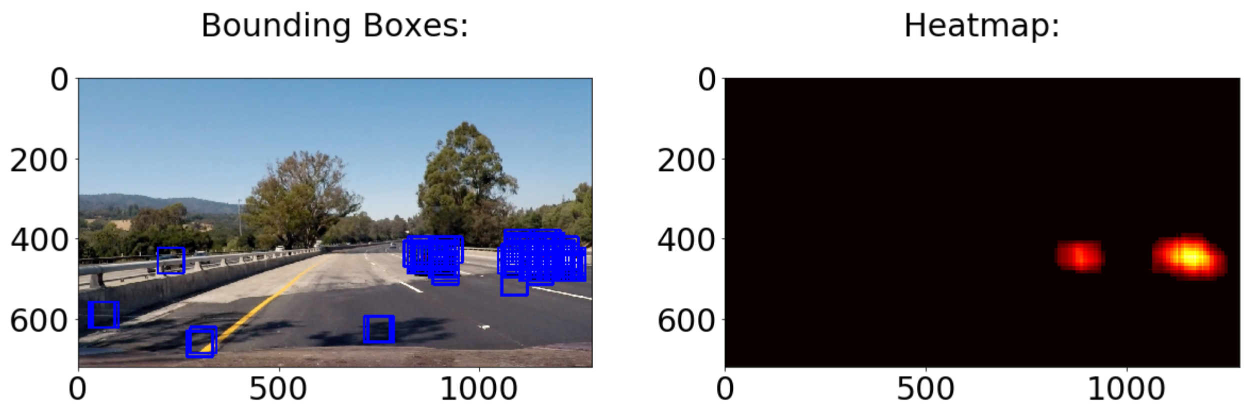
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

A link to the final video output for this project is provided below. The code pipeline performs reasonably well on the entire video.

https://youtu.be/MKBiDWsfA9A

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.



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?

Firstly, The biggest challenge with this project was getting everything to work on a video stream. Due to the limitations of my laptop, a CPU, I was not able to troubleshoot the parameters as much as I would have liked, as each iteration of testing took a frustratingly long period of time. That said, I was able to find a few combinations of feature extraction parameters that performed better than others, and ultimately I realized the importance of spatial binning and HOG as two critical factors in creating a successful pipeline.

Secondly, I would also have figured out a way to smooth the vehicle detections from frame to frame. Perhaps some of my code would have run faster, and the overall output would be more desirable.

Lastly, the sliding window technique that I implemented was carried out for each frame, and HOG features were generated within each window, increasing processing time. Although I was able to produce a decent output, the code would have run faster had I run HOG features once as an array over the whole test image, then applied indexes of that array to each individual sliding window. Perhaps I will implement this concept in the future.

Future Plans

- Include Additional Datasets- I would like to implement Udacity's recently released dataset in the future.
- Test Additional Classifiers- Linear SVC was sufficient for this project, as it provided me with above 94% accuracy on a regular basis. However, if in the future I require a real-world system that must be able to classify cars on a human performance-level, I would need to experiment with other classifiers to improve past 94%.

--End Of Document-------------------------------------