Vehicle Detection Project

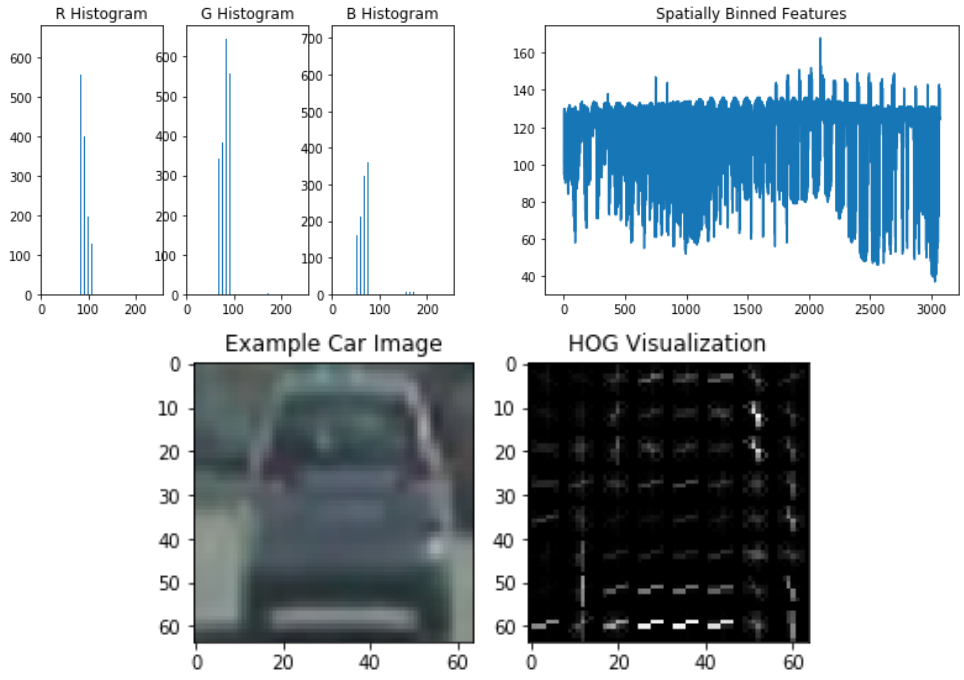
The goals / steps of this project are the following:

- 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.

###Histogram of Oriented Gradients (HOG)

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

- The code for this step is contained in the 2nd and 3rd code cell of the IPython notebook.
- In the 2nd code cell, I created a function get_hog_features() with image, orientation, pix_per_cell, cell_per_block as inputs which I call from my extract_features() function in the same code cell.



- 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.
- I noticed that when I used a larger pixels_per_cell, the resulting HOG feature had less detail and would not train as well.

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

- I tried various combinations of parameters and compared the HOG visualization and the results from the search windows until I was able to get the least number of false positives.

####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).

- I trained a linear SVM in the 4th code cell. Here I loaded all of the images of cars and non cars, then extracted the features into an array.

- I then apply the StandardScaler to remove the mean and scaling to unit variance.

- Next I split the data into 80% train and 20% test set.

- I then train the model and print the accuracy which is **99.13%**

###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 decided to use the HOG sub-sampling window search suggested in lesson #35 of the project. I liked the idea of it being more efficient. This can be seen in the 6th code cell in the function find_cars().

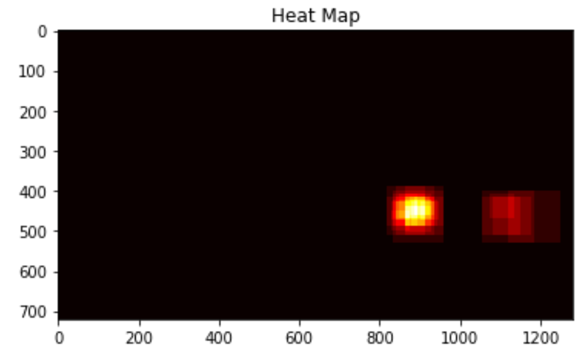
- I use find_cars() inside of my process_image() function in the 7th code cell for each window and scale.

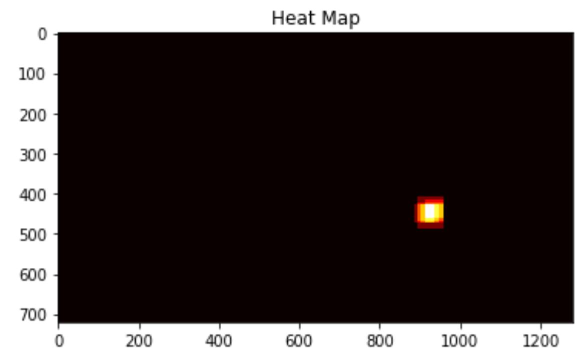
- I determined these windows and scale through trial and error. Starting with a small scale of .5 and moving up. Once I got a better understanding of how each scale performed, I chose to use the ones that would find the cars in various images.

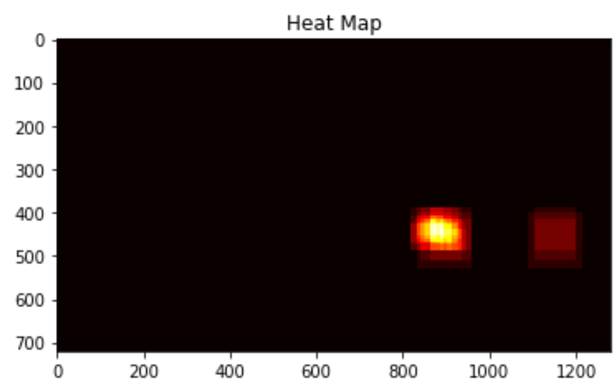
####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

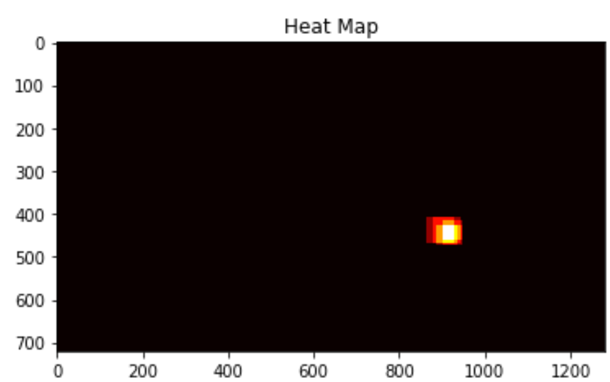
- Ultimately I searched on four scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

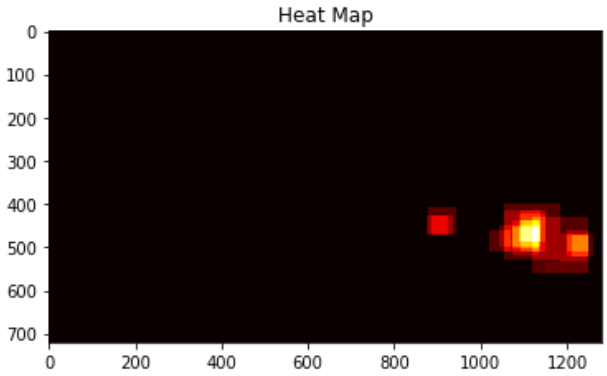
- I used heat maps to help tune the windows and scaling size of my search function. Here are some example images:











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.

I recorded the positions of positive detections in each frame of the video. I would then add up the detections for each window and scale and created a heat map. In the add_heat() function in the 7th code cell, I went through all of the detected boxes and if the heat map in this window had any values above the threshold, I would then increase every pixel contained in the window to above the threshold. I then used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap and 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?

This project came together really quickly since wrote a lot of the code during the project lectures. So when combining everything and creating the pipeline, I was focused more on getting a working pipeline at first, then focusing on tuning parameters. This helped me move along much more quickly and add enhancements later. One thing that slowed me down was not paying attention to the number of training examples being used. I wasn't reading in the whole set, so when tuning the parameters I would get a lot of false

positives. This lead me down a path of tuning for the wrong model. After going back and verifying that I indeed trained on the right data set, I was able to tune everything much quicker.