P5 -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.

1. Loading and Visualizing the data

The code for this step is contained in the second and 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:

In this project, I used the image data provided by Udacity, including the pictures of the car and 8968, which are not the pictures of the car. They are 64*64, as shown in the figure:

2.Define a function to return HOG features and visualization

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).

The scikit-image `hog()` function takes in a single color channel or grayscaled image as input, as well as various parameters.

Parameters:

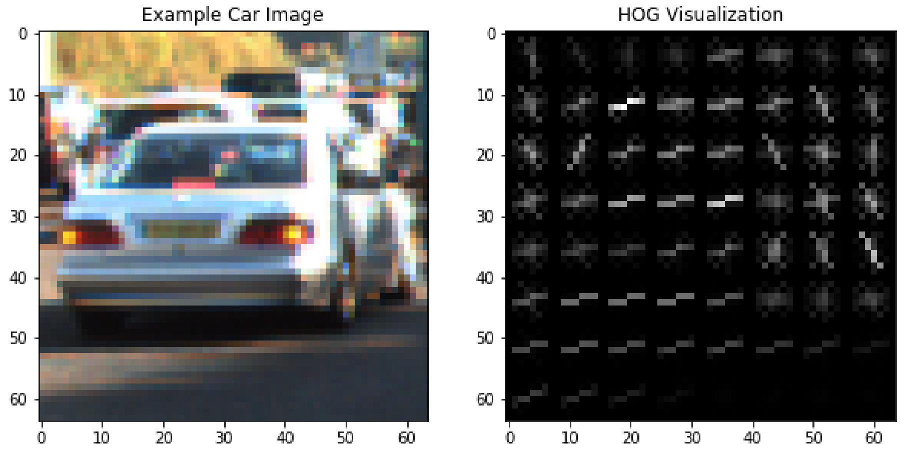
`orientations` is specified as an integer, and represents the number of orientation bins that the gradient information will be split up into in the histogram. Typical values are between 6 and 12 bins.

`pixels_per_cell`is specifies the cell size over which each gradient histogram is computed.

`cells_per_block` is also passed as a 2-tuple, and specifies the local area over which the histogram counts in a given cell will be normalized.

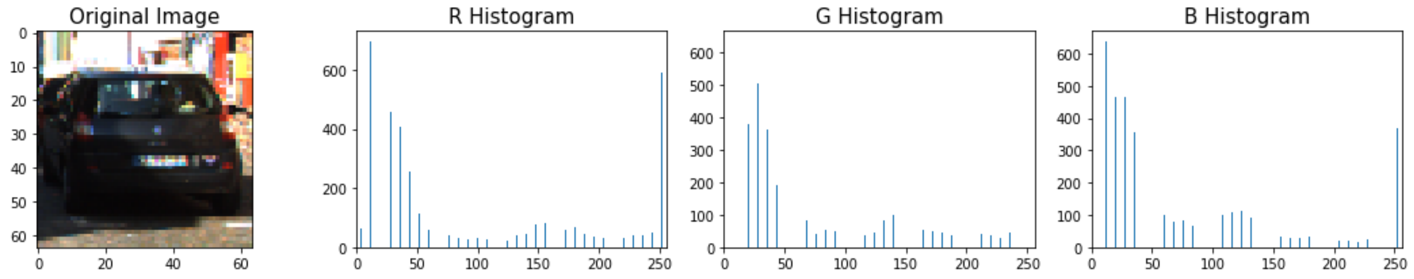
`transform_sqrt` is another optional power law or "gamma" normalization scheme set.

output looks like.



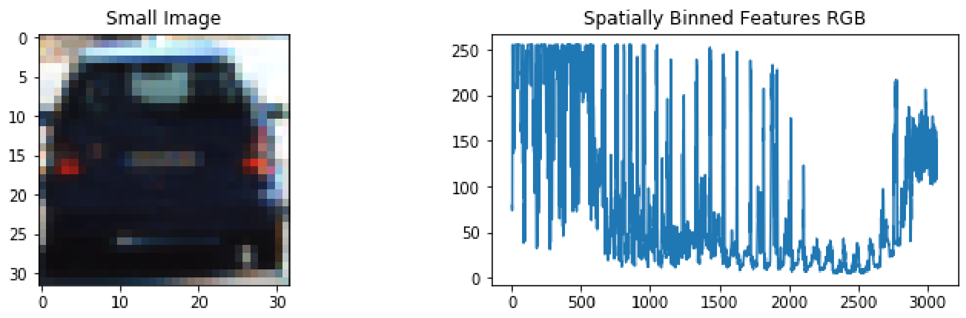
3.Define a function to compute color histogram features

`color_hist`is takes an image and computes the RGB color histogram of features given a particular number of bins and pixels intensity range, and returns the concatenated RGB feature vector, result like this:



4.Define a function to compute color histogram features

`bin_spatial()`is convert this to a one dimensional feature vector.`cv2.resize()` is a convenient function for scaling down the resolution.`*.ravel()` is convert this to a one dimensional feature vector, result like this:



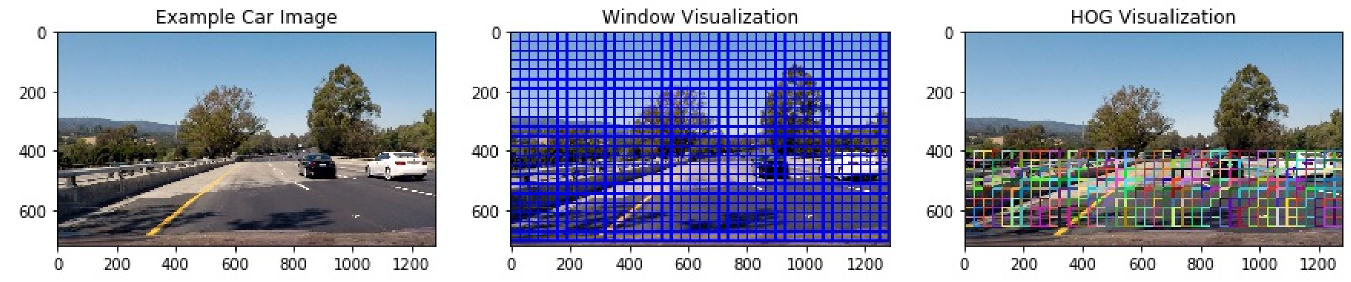
5. Define a function to extract features from a list of image locations

This function could used to call `bin_spatial()` , `color_hist()` and `get_hog_features()` to extract flattened spatial color features and color histogram features and HOG features and combine them all (making use of StandardScaler) to be used together for classification.

6. Sliding Window

Define a function that takes an image,start and stop positions in both x and y, window size (x and y dimensions), and overlap fraction (for both x and y).

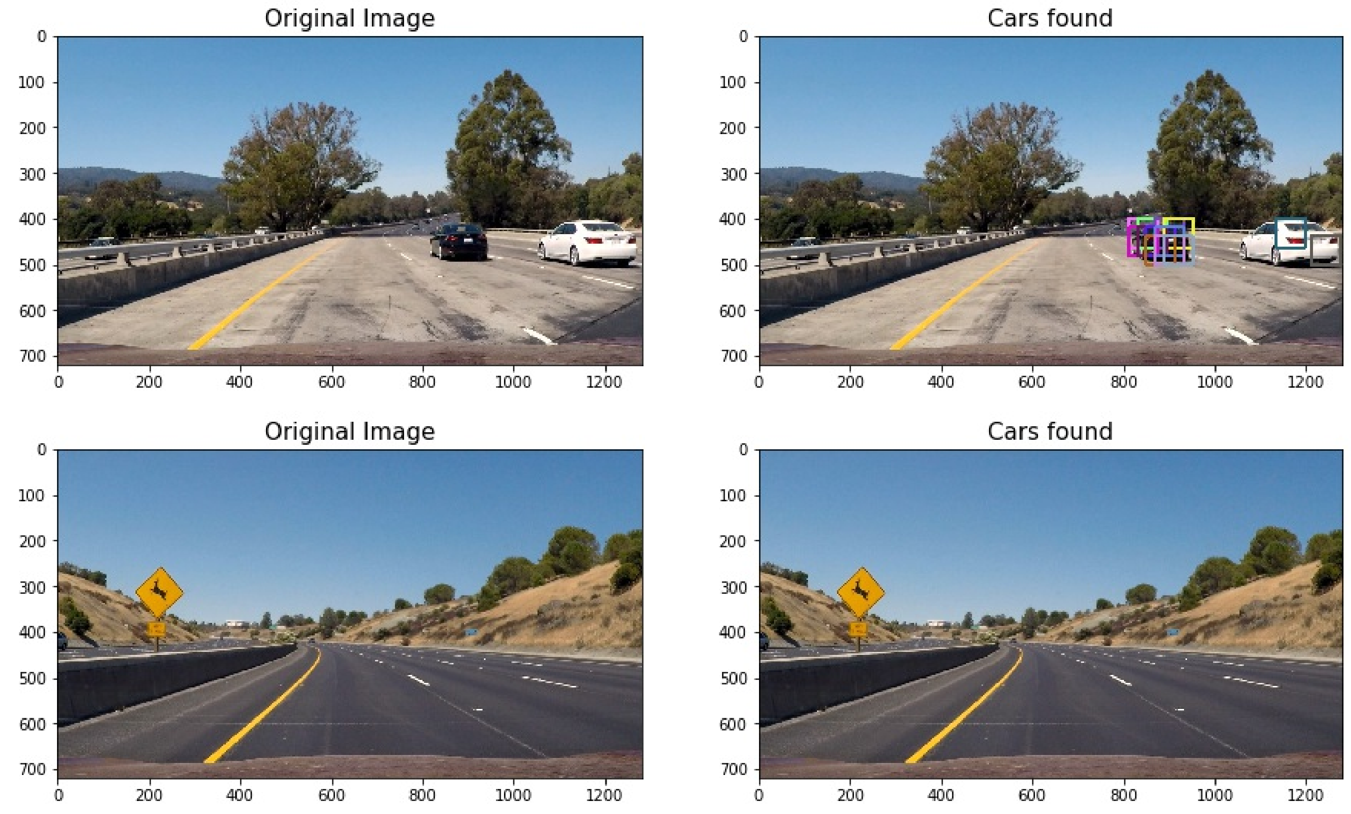
The following is visualized by adjusting the y*start*stop value [400,656] and y*start*stop=[None, None], and the window size is (64,64).

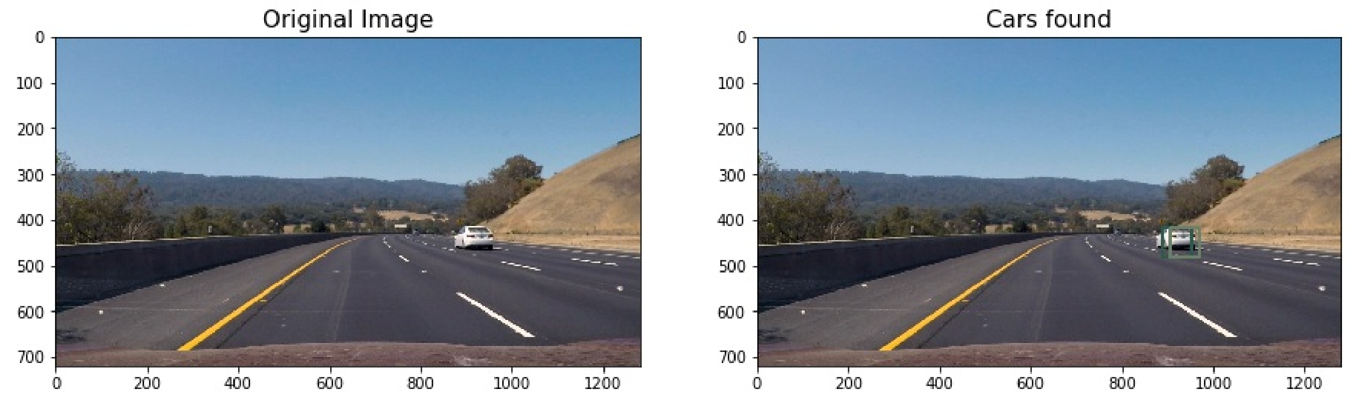


7. Defining a function to extract features from a single image window and The list of windows to be searched

Two new functions are defined: `single_img_features()` and `search_windows()`. these to search over all the windows defined by your slide_windows(), extract features at each window position, and predict with your classifier on each set of features. This function is very similar to `extract_features()` just for a single image rather than list of images.`Search_windows()` passes the image to `single_img_features()` to extract the HOG and color features and color space features, and then to predict the image.

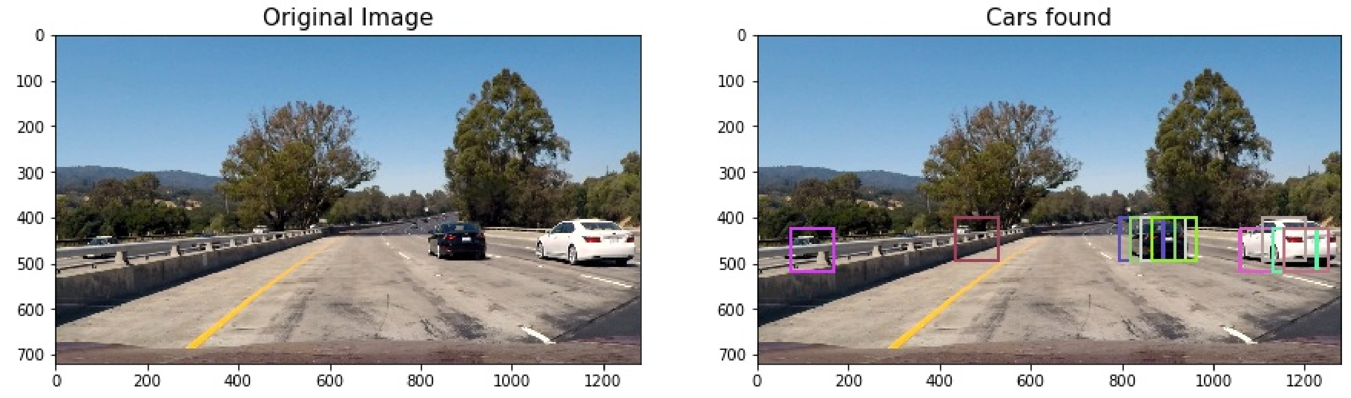
Of course, I used linear SVM as the predictive image model. After training, my model was correct with 98.62%, which is a good number.Then need to select the relevant parameter values: color space, I choose 'YCrCb, of course, you can also choose to RGB, HSV, LUV, HLS, YUV, YCrCb, etc., HOG orientations I choose 10, HOG pixels*per*cell of 8, HOG cells per block is 2, HOG channel for' ALL 'is ALL color channel, Spatial binning dimensions is 32, Number of histogram bins is 64. HOG features, Histogram featureson, Spatial features is on. the results like this:

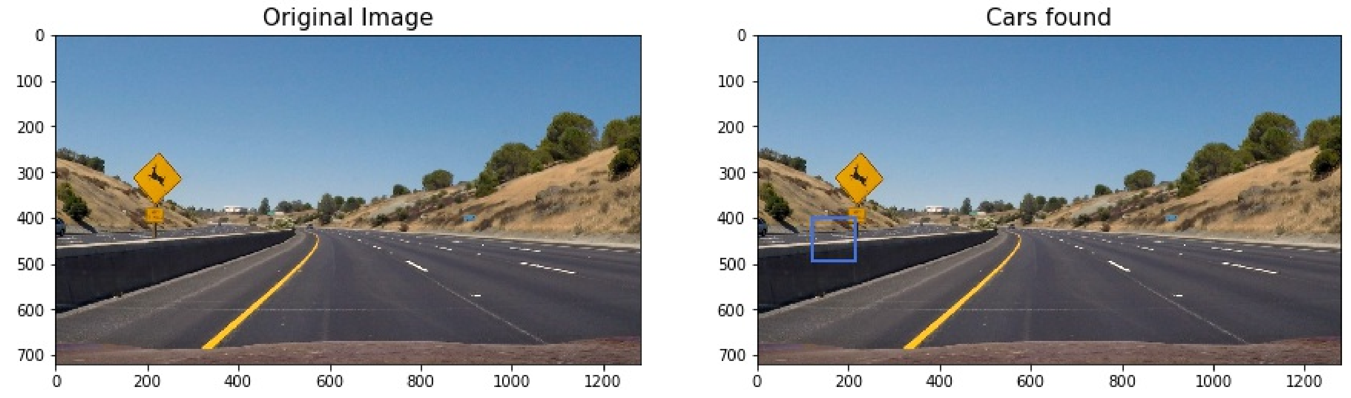


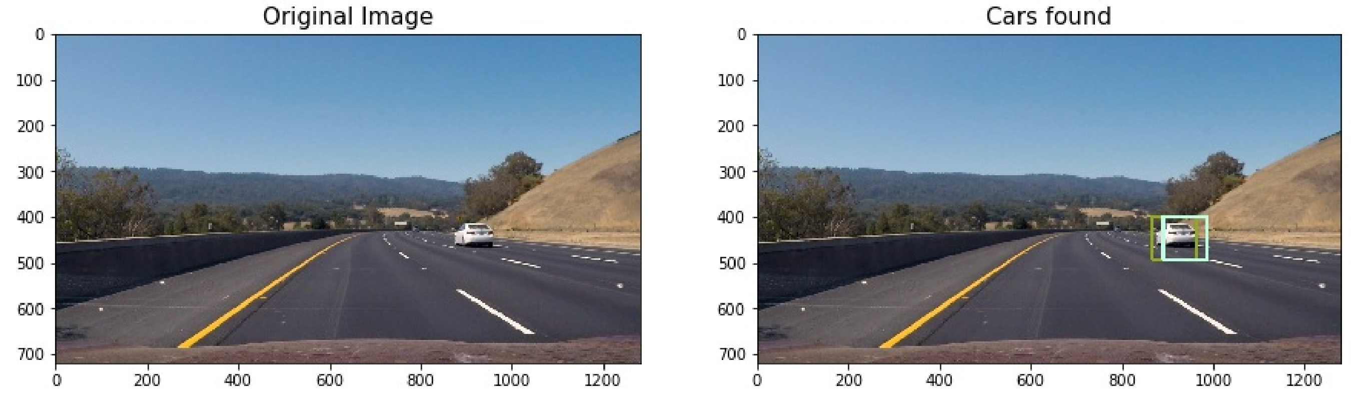


8.Define a single function that can extract features using hog sub-sampling and make predictions

`find_cars` that's able to both extract features and make predictions.Is a collection of the above functions.you can see a result like this:



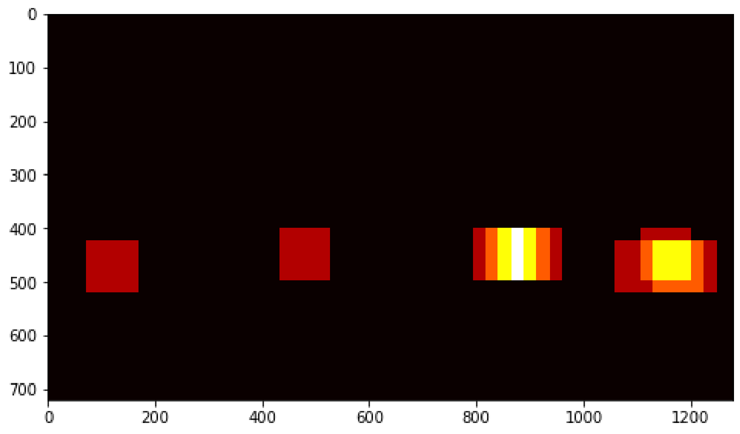




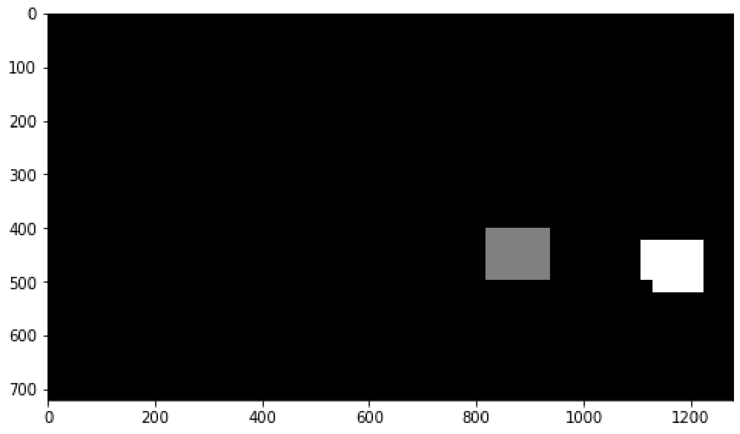
9. Adding heatmaps , bounding boxes and find cars Method

I'm showing all the bounding boxes for where my classifier reported positive detections. You can see that overlapping detections exist for each of the two vehicles, and in two of the frames, I find a false positive detection on the guardrail to the left. so we are build a heat-map from these detections in order to combine overlapping detections and remove false positives.

1.`add_heat()` make a heat-map, you're simply going to add "heat" (+=1) for all pixels within windows where a positive detection is reported by your classifier.like this:



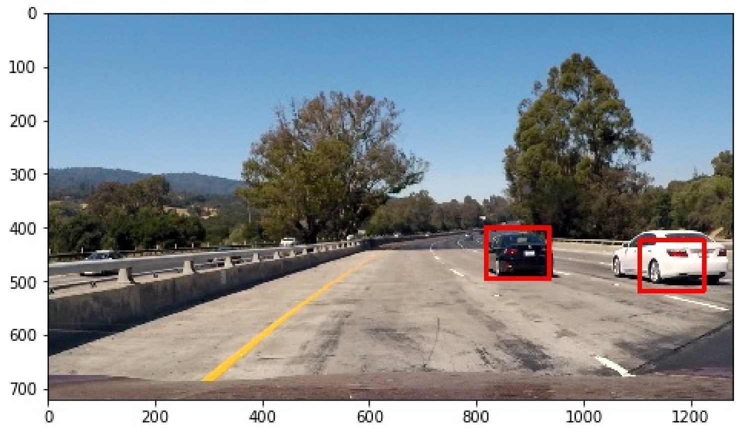
2.Then the "hot" parts of the map are where the cars are, and use `apply_threshold()` imposing a threshold, you can reject areas affected by false positives.like this:



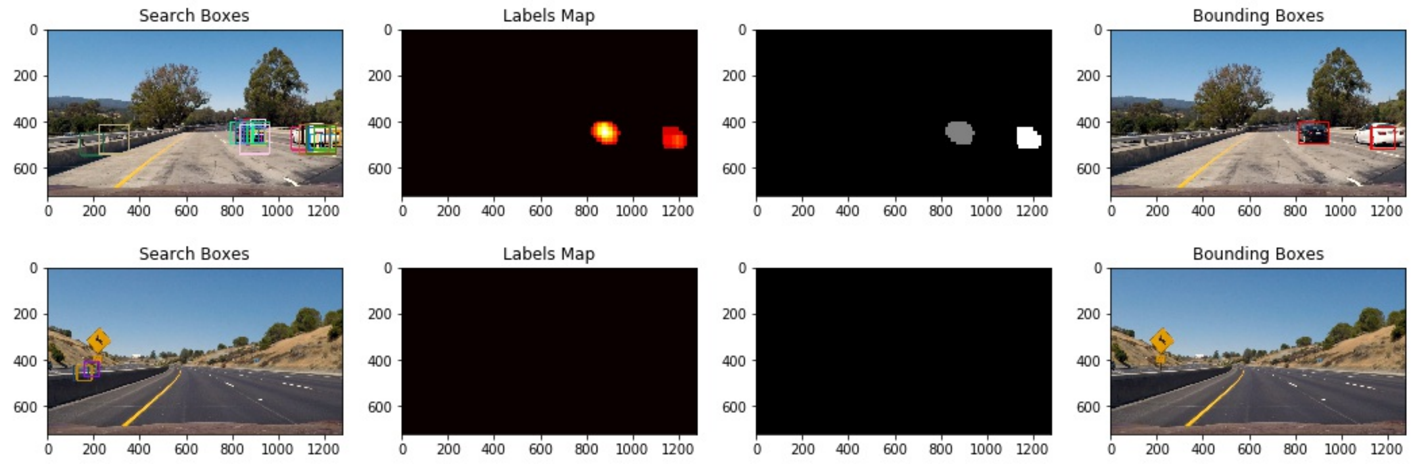
3.Once you have a thresholded heat-map, we use `label()` function from `scipy.ndimage.measurements`.Now labels is a 2-tuple, where the first item is an array the size of the heatmap input image and the second element is the number of labels (cars) found.

The labels[0] is the last picture.

The labels[1] like this :



Based on the number of boxes found, I set a threshold of 4. Now let's look at the effect of the test image:



10.Video output

Finally, I define pipeline function: `process_image ()`, set up inside the related parameters, also used the car of the three range to search image (ystart, ystop), including: (390, 500), (400,550), (410, 600), so can more accurately identify to the car, of course, you can increase the range, after continuous adjustment, can get a better position to accurately identify the vehicle.

Here is a link to the final effect:

link to my video result

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?

To improve the accuracy of the model, I chose the linear SVM. I spent a lot of time here to verify the accuracy of various color Spaces and finally chose YCrCb.HLS has the highest accuracy, but it is easy to over-fitting and easy to identify errors.HSV is also a good choice. Here, YCrCb is selected because it is relatively stable for this project, and it is better to recognize the effect of the car.If you have more data, consider other models, such as neural networks.