# ****Vehicle detection****

Longxing Tan

**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.

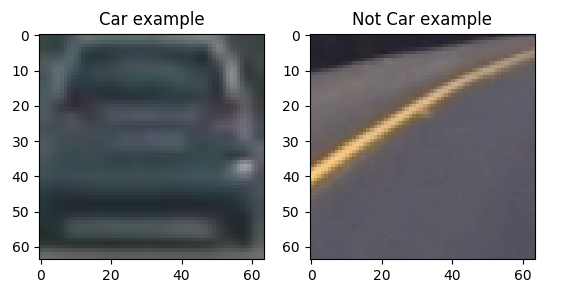
**Rubric Points**

**1. Histogram of Oriented Gradients (HOG)**

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

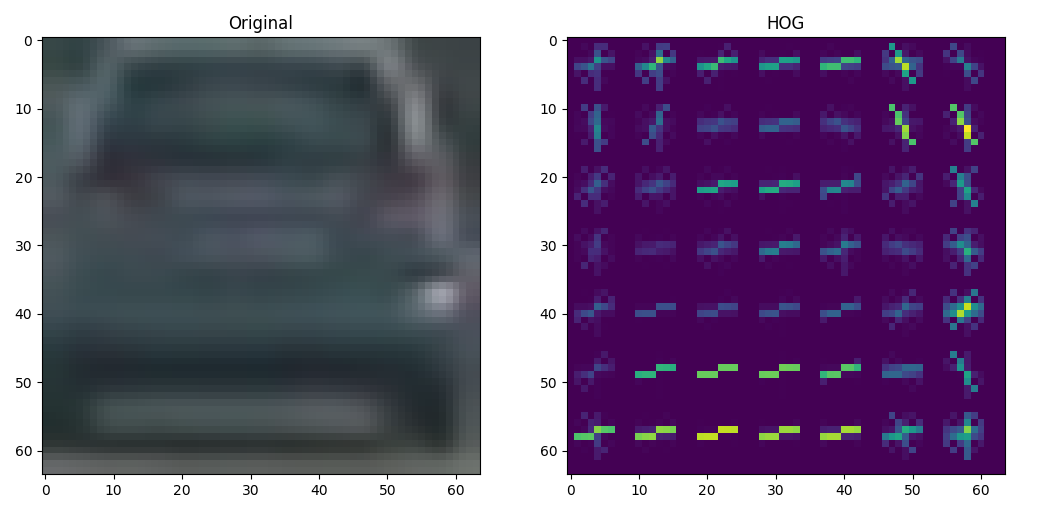
The code for this step is contained lines 17 through 39 of the file called vehicle\_detection.py).

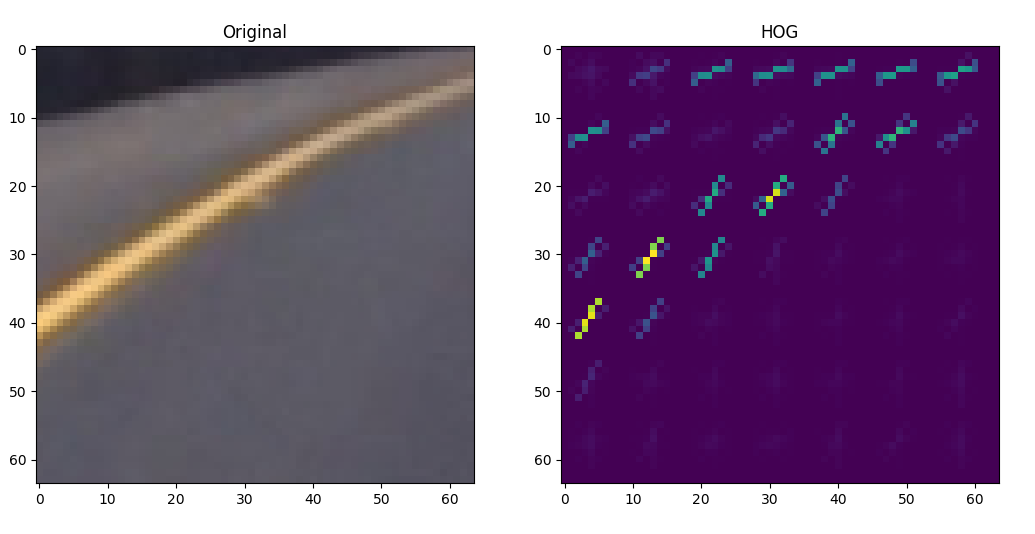
I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:



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.

Here is an example of HOG parameters of orientations=8, pixels\_per\_cell=(8, 8) and cells\_per\_block=(2, 2):



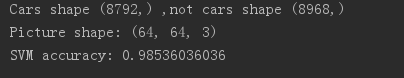


**2. Explain how you settled on your final choice of HOG parameters.**

I tried various combinations of parameters and based on a SVM base model accuracy to choose the best features (lines 151 through 157 of the file called vehicle\_detection.py). it turns out that the example of the courses can meet the classifier accuracy request, which means the orientations=8, pixels\_per\_cell=8, cells\_per\_block=2.

**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).**

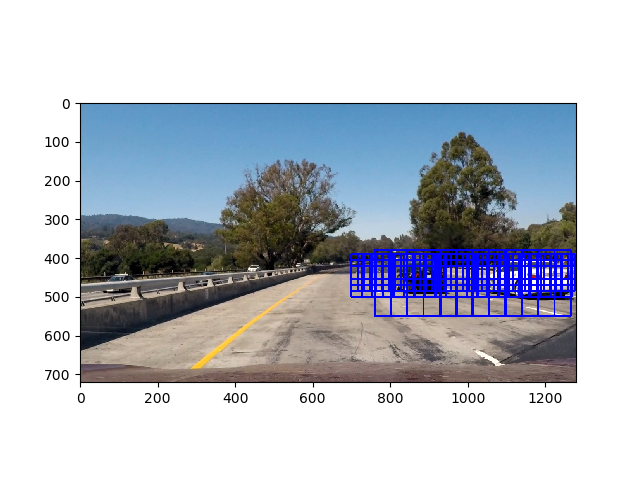
I trained a linear SVM using LinearSVC based on the features of the stack of spatial features, color histogram features, hog features. The accuracy of test images set is 98.5%. And the default C of 1 is all right.



**Sliding Window Search**

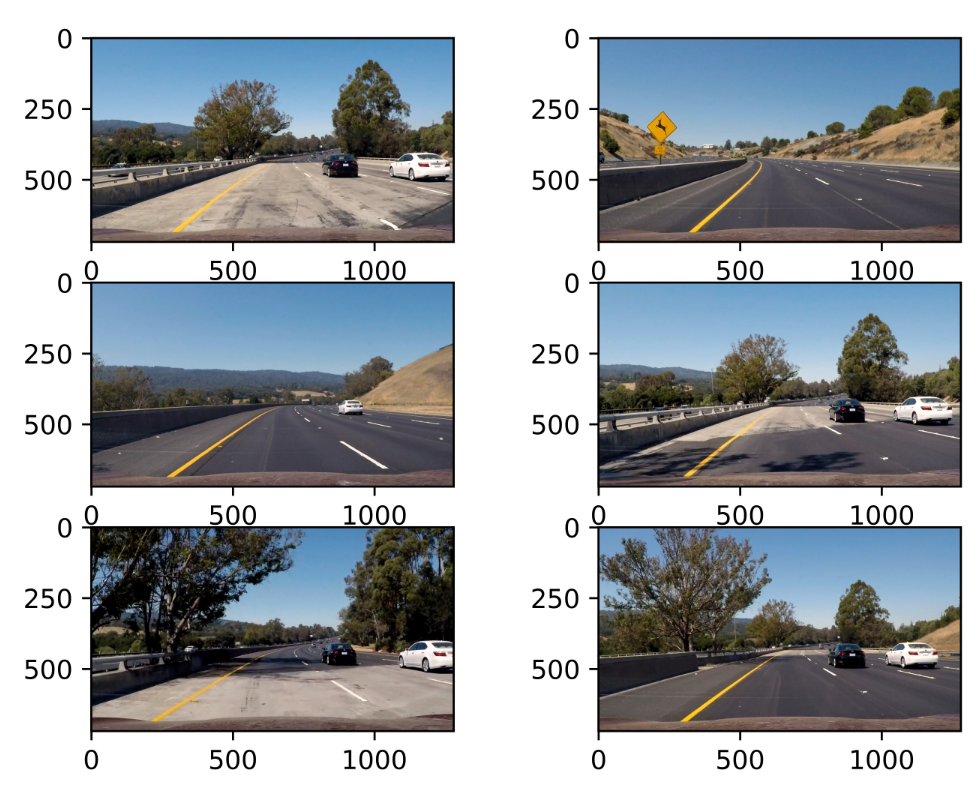
**1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?**

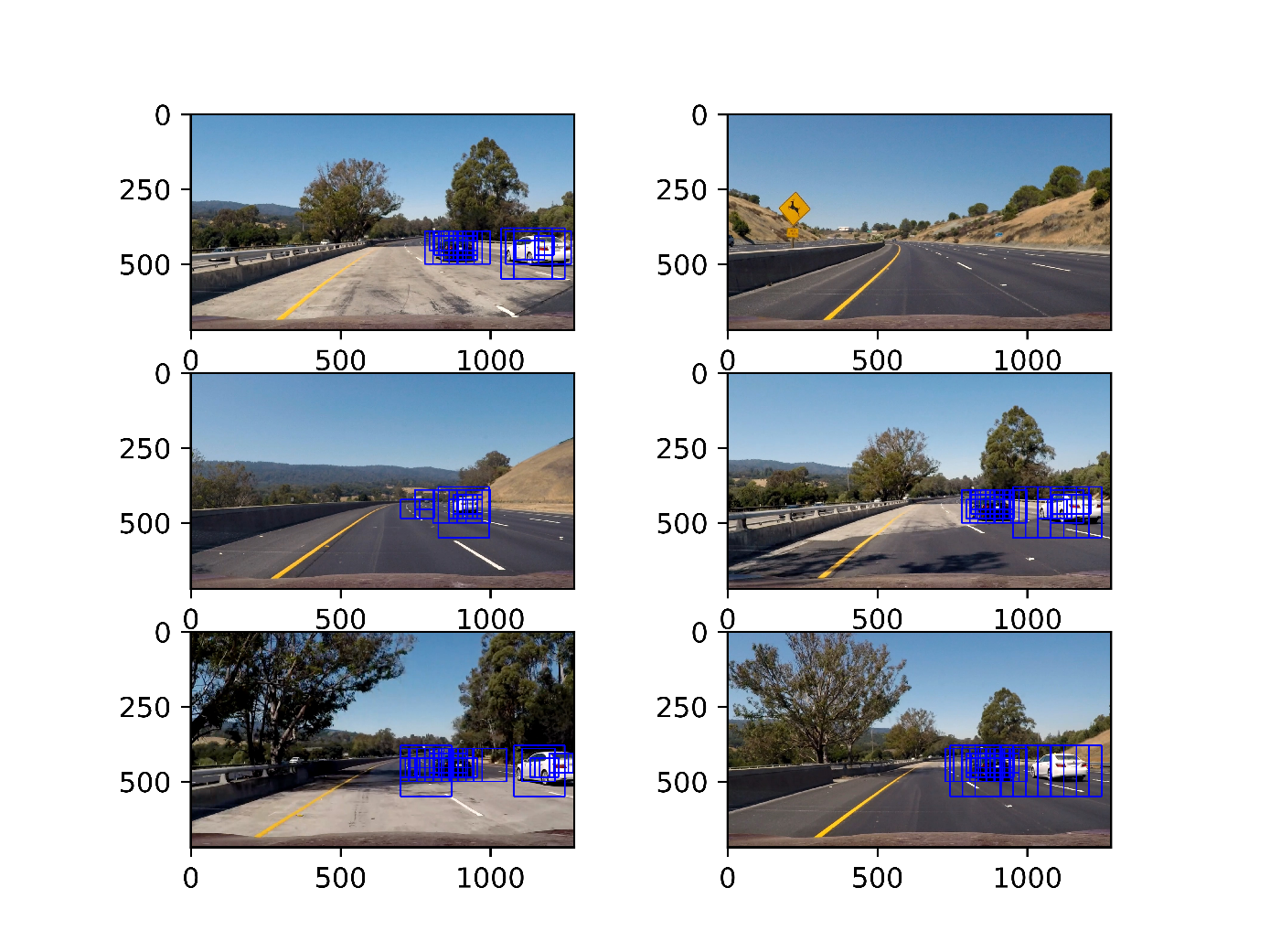
I decided to search window positions in three levels, (64,64),(110,110),(170,170), and the overlap is 75%. Through this, the windows could cover and differentiate the vehicles and non-vehicles. (line 245 through 254 in vehicle\_detection.py)



**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 three scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. 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.)**

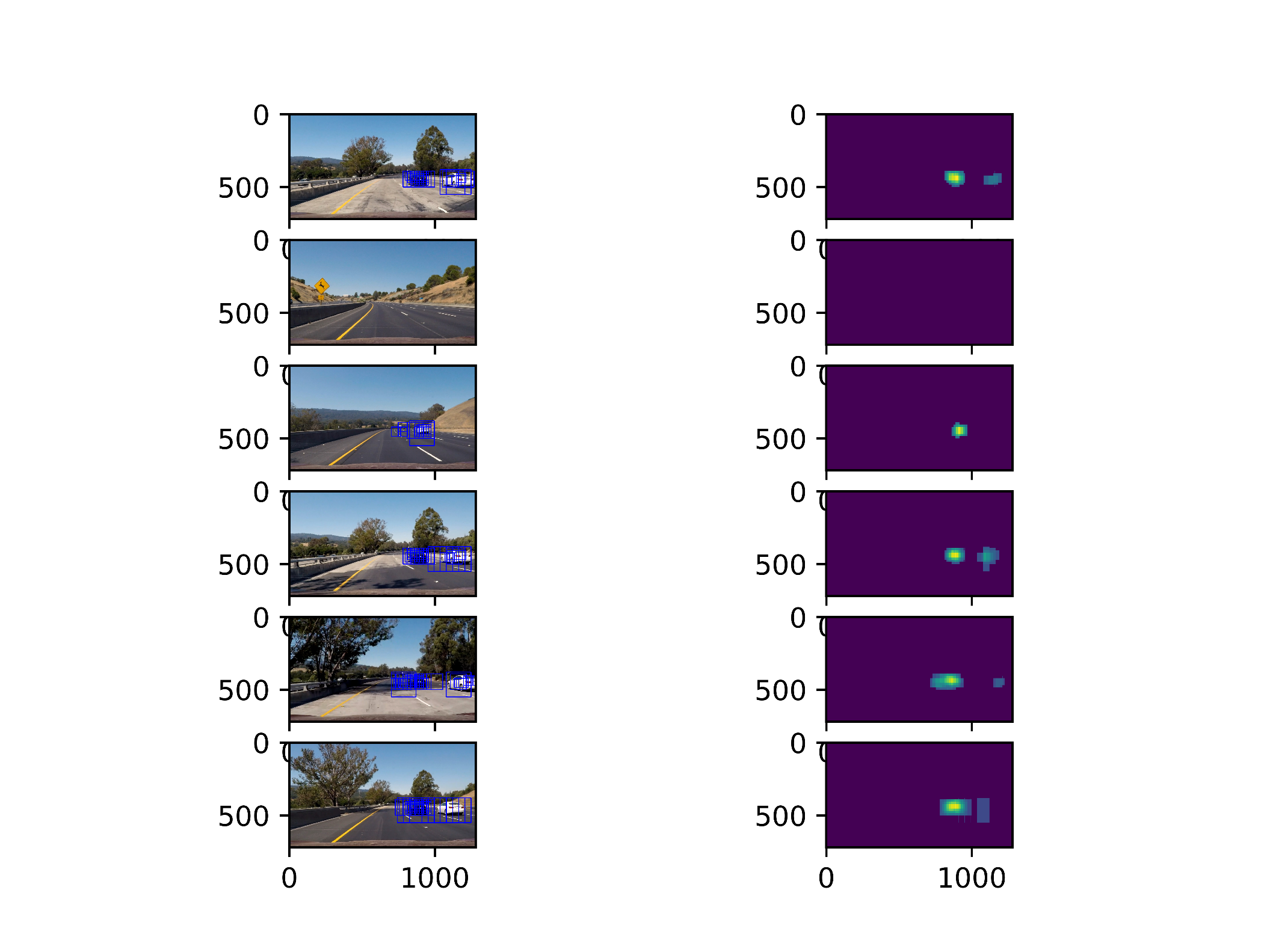
Here's a [link to my video result](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/project_video.mp4)

**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.**

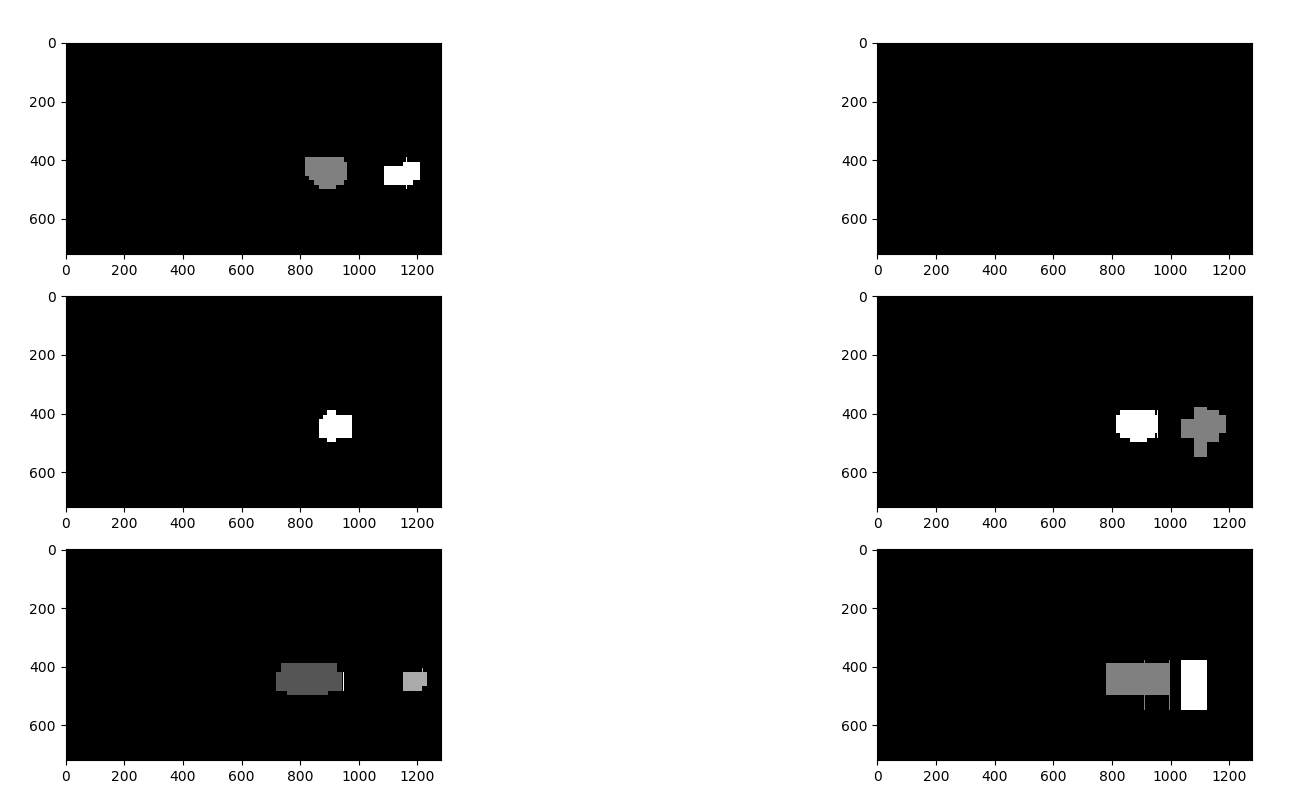
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Here's an example result showing the heatmap from a series of frames of video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of video:

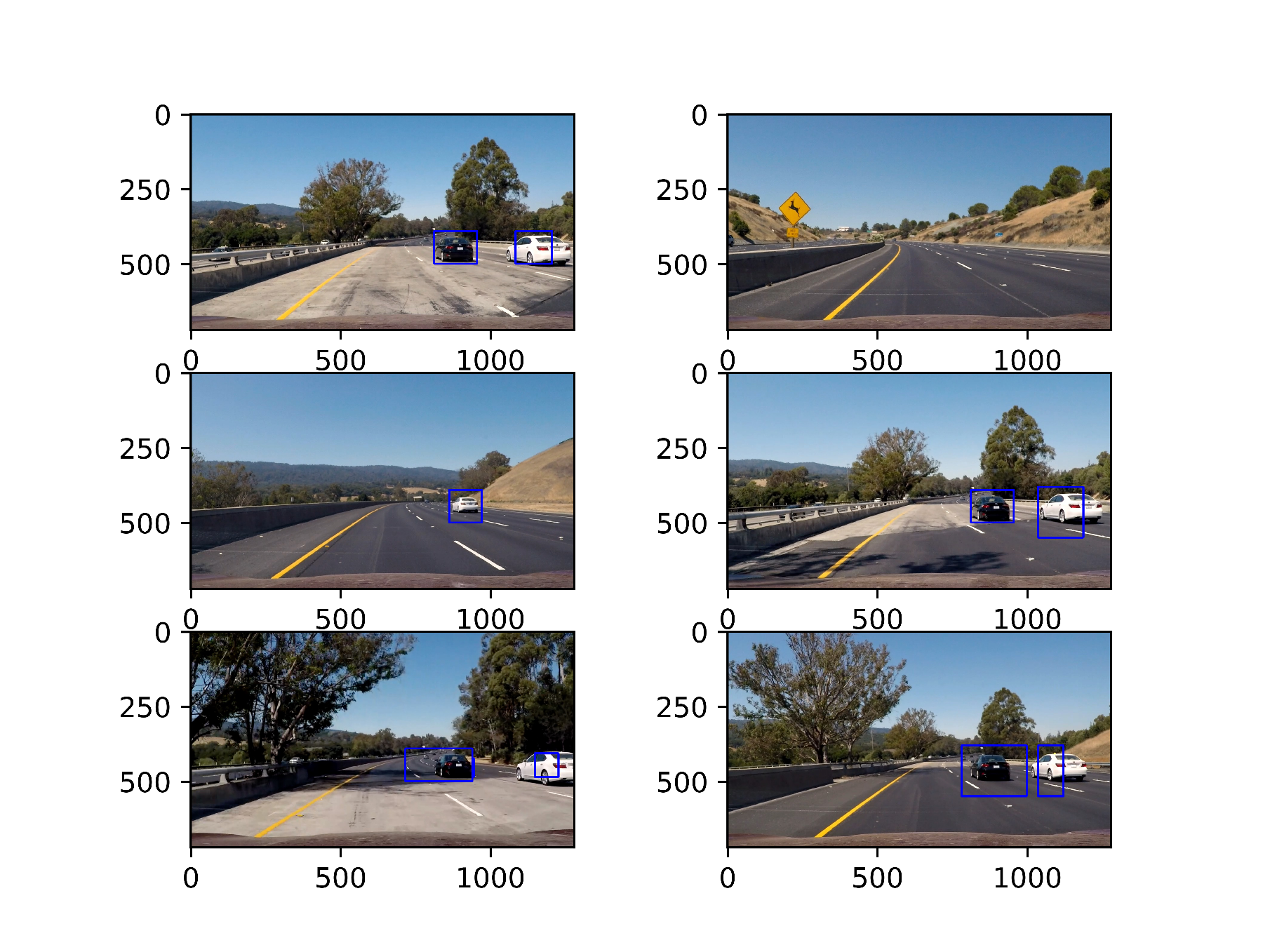
**Here are six frames and their corresponding heatmaps:**



**Here is the output of scipy.ndimage.measurements.label() on the integrated heatmap from all six frames:**



**Here the resulting bounding boxes are drawn onto the last frame in the series:**



**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?**

So, fist I train a SVM classification model to differentiate the vehicle or not vehicles based on the features combination of YCrCb 3-channel HOG, spatially binned color and histograms of color. The test accuracy could reach more than 98% accuracy. Then I search three scale level windows of 64, 110, 170 of the area where the vehicle could appear x>=700, 550>=y>=380. For each window, it will be resized to (64, 64) and input into the classifier. If it’s vehicle, the window position will be hashed. Then to remove the duplicate box and the false negative cases, the heatmap and threshold is implemented. Finally the label function from scipy is used to make it clear how many vehicles are detected and where they are detected.

The most surprising thing is that I tried the method of the course which calculate the hog only once, and then choose the window area hog as the feature. But I can’t adjust it successful. So I still come back to calculate each window’s hog every time. Also a little difficult thing is how to choose the window and threshold, where I spent most of tuning time. At last the result is just all right.

Maybe a CNN model like yolo is more robust.