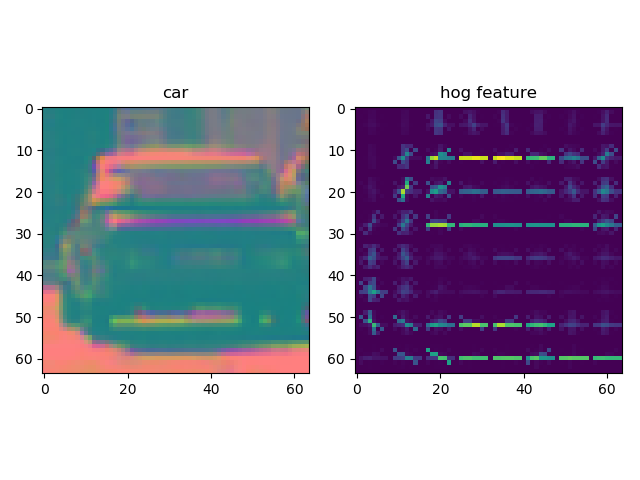
**CarNd-P5 Write Up**

# Histogram of Oriented Graidents(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 function ‘extract\_features()’ in the line 78 in ‘train\_svm.py’. First I read car and no-car images and transform them to ‘YCrCb’ color space. Then I flatten image to get spatial feature, and get it’s color histogram with bins value 32, and get hog features using skimage’s hog() function.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=9`, `pixels\_per\_cell=(8, 8)` and `cells\_per\_block=(3, 3)`:



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

I tried various combinations of parameters and their accuracies are all high.

Finally I chose the best one:

orientations=9,

pixels\_per\_cell=(8, 8)

cells\_per\_block=(3, 3)

## 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).

This step code is in line 111-136 in ‘train\_svm.py’.

First, I extract images’ spatial feature, color feature and hog feature and combine them use ‘numpy.concatenate()’ function.

Second, I make features and labels to numpy form.

Third, I split data in to train, test parts.

Fourth, I normalize my training data.

Finally, I fit data.

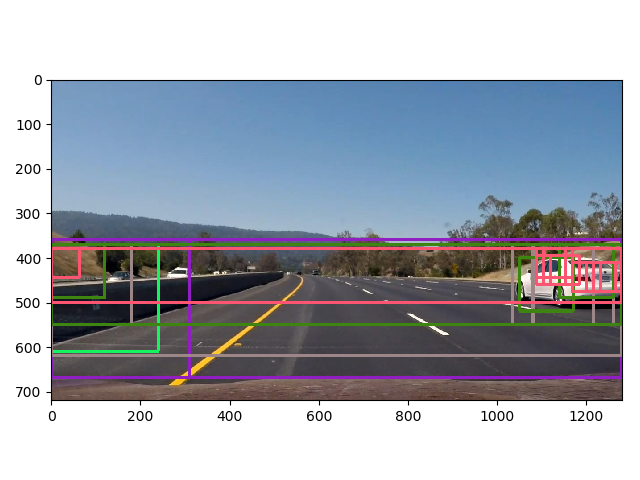
# Sliding Window Search

## 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?

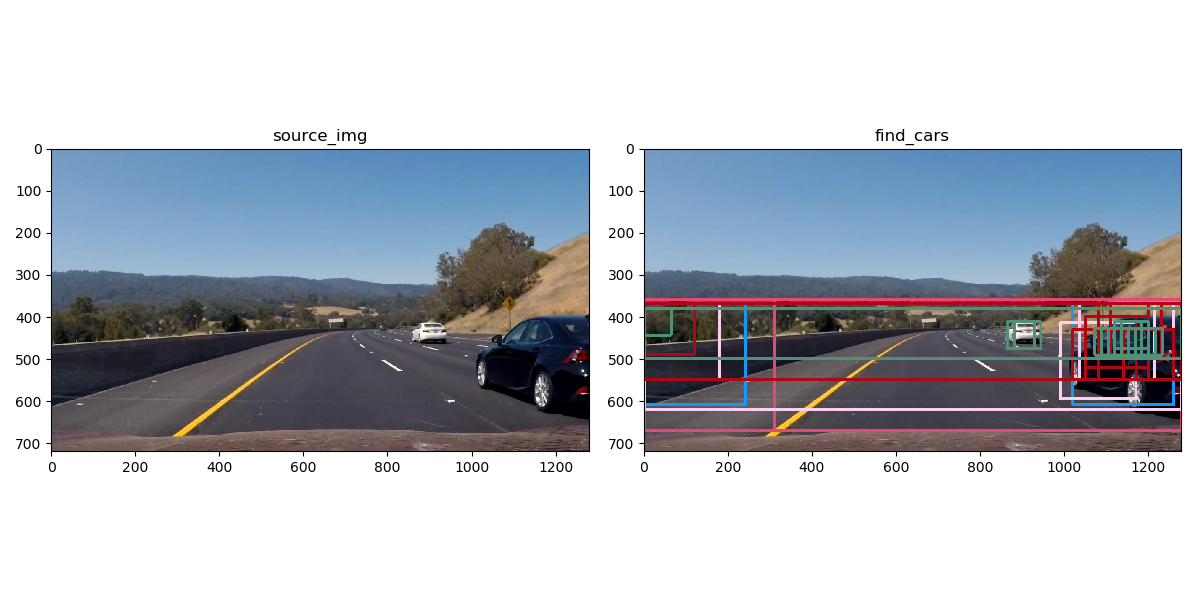
The code is in function find\_car() in line 86 in file ‘find\_cars.py’. I count how many steps will be need sliding in x and y direction, and write a two-layer loop to loop through every single step.

I draw the range and window on test image to make sure scales and the numbers of overlap just like this

Finally, I use five overlaps.



## 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?

Examples:

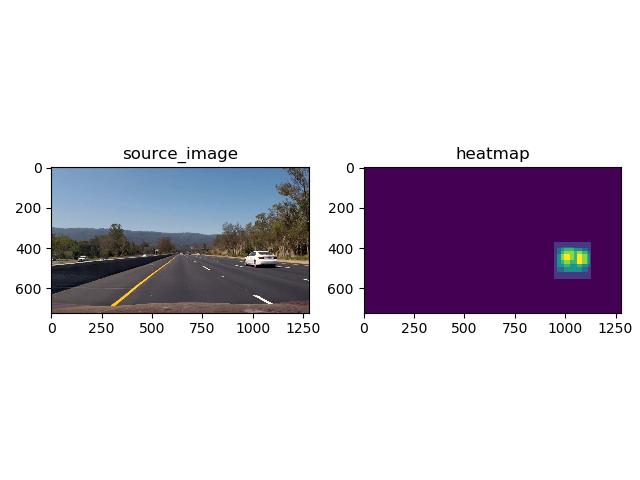
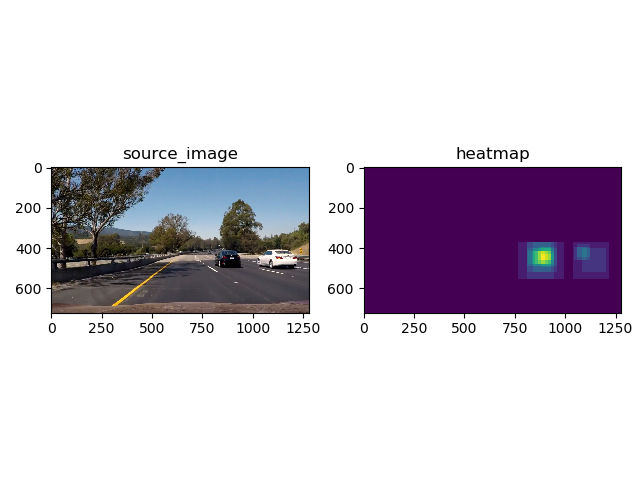
I used a linear svc at first, but it’s not good enough for sometimes it can not detect car and will have false positive results. So I use svc’s default parameters and get a good result.

# Video Implementation

## 1. ‘find\_cars.mp4’ is my final video output.

## 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 use heatmap to combine bounding boxes. My code is at line 154 in function ‘get\_realcar\_boxes()’ in file ‘find\_cars.py’

Heatmap like this:

I use `scipy.ndimage.measurements.label()` to make sure cars bounding box.

Filter: this step’s code is in function ‘draw\_cars()’ at line 178 in file ‘find\_cars.py’.

I create a deque(length=3) to storage boxes detected. Also, I use a heat map to draw boxes in deque. If a box got detected in continuous 3 frames, it’s center value will be three. This way, I delete false positive value.

# Discussion

1. The speed of my code to deal video is very low. The reason might be my svm model is too complex. I could optimize my algorithm to speed up it.

2. Although my code always detected cars, the bounding box is not stable. It’s related to the sliding window’s size. If I speed up my algorithm, I can use more overlap which will get a more stable result.