# 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 do not 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.

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

## Histogram of Oriented Gradients (HOG)

The code for this step is contained in the third code cell of the IPython notebook (function : get_hog_features)  I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle(First image) and non-vehicle(second image) classes:

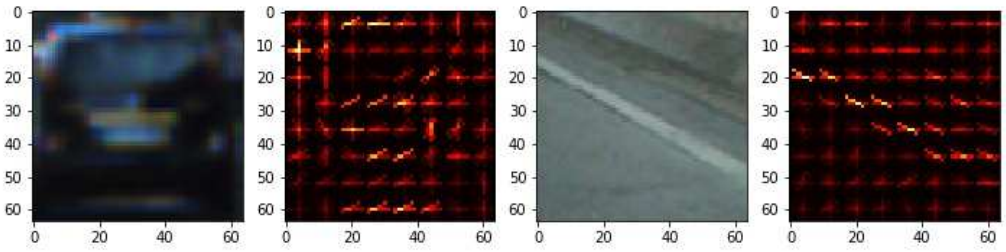


Image1 : HOG features using RGB color space

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. 2nd and 4th image in above. And this result is from using RGB color space

Here is an example using the YCrCb color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

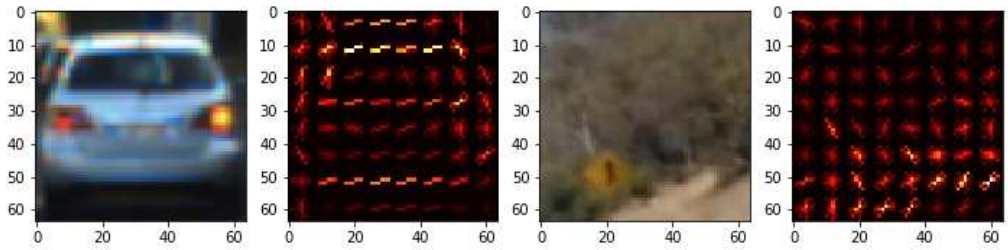


Image2: HOG features using YCrCb color space

I tried various combinations of parameters and plotted their results; Image1 is based on RGB and orientation 6 and image 2 based on YCrCb and orientation 9.

Feature extraction in Image 2 is better than Image 1 so concluded to use YCrCb color space and Hog orientation 9 and other parameter values.

## Train model with car and non-car images :

This involves multiple steps as described below:

1. Extracted Hog features from vehicle and non-vehicle images using same parameters described above(Ycrbc, color space and Hog orientation 9)
2. Used sklearn.preprocessing.StandardScaler() to normalize training
3. Split up data into randomized training and test sets
4. Then trained a linear SVC
5. Check  Accuracy by running model on Test set by using score

Following results is printed from the cell 5(which also includes above mentioned steps)

```
75.98 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 6108
20.93 Seconds to train SVC...
Test Accuracy of SVC =  0.9893
My SVC predicts:  [ 1.  0.  1.  0.  0.  1.  1.  0.  1.  0.]
For these 10 labels:  [ 1.  0.  1.  0.  0.  1.  1.  0.  1.  0.]
0.019 Seconds to predict 10 labels with SVC
```

## Sliding Window Search

First used sliding window search to find vehicle with Y start and stop position [400,656] and with overlap of 0.5. Below image is the result on the test images, this part of the code implemented in cell 6.

**Image 3: Sliding window search result**

I decided to choose 0.5 for overlap and 1.5 since these 2 values gave better result as in Image3. But observed it takes long time for detection.

Then Decided to use Hog Sub-sampling Window Search, learned from the lesson that it is more efficient method for doing the sliding window approach, this method allows us to only have to extract the Hog features once. The code corresponding code implemented in function `find_cars`, which is in cell 8 and this function is able to both extract features and make predictions.

The `find_cars` only has to extract hog features once and then can be sub-sampled to get all of its overlaying windows

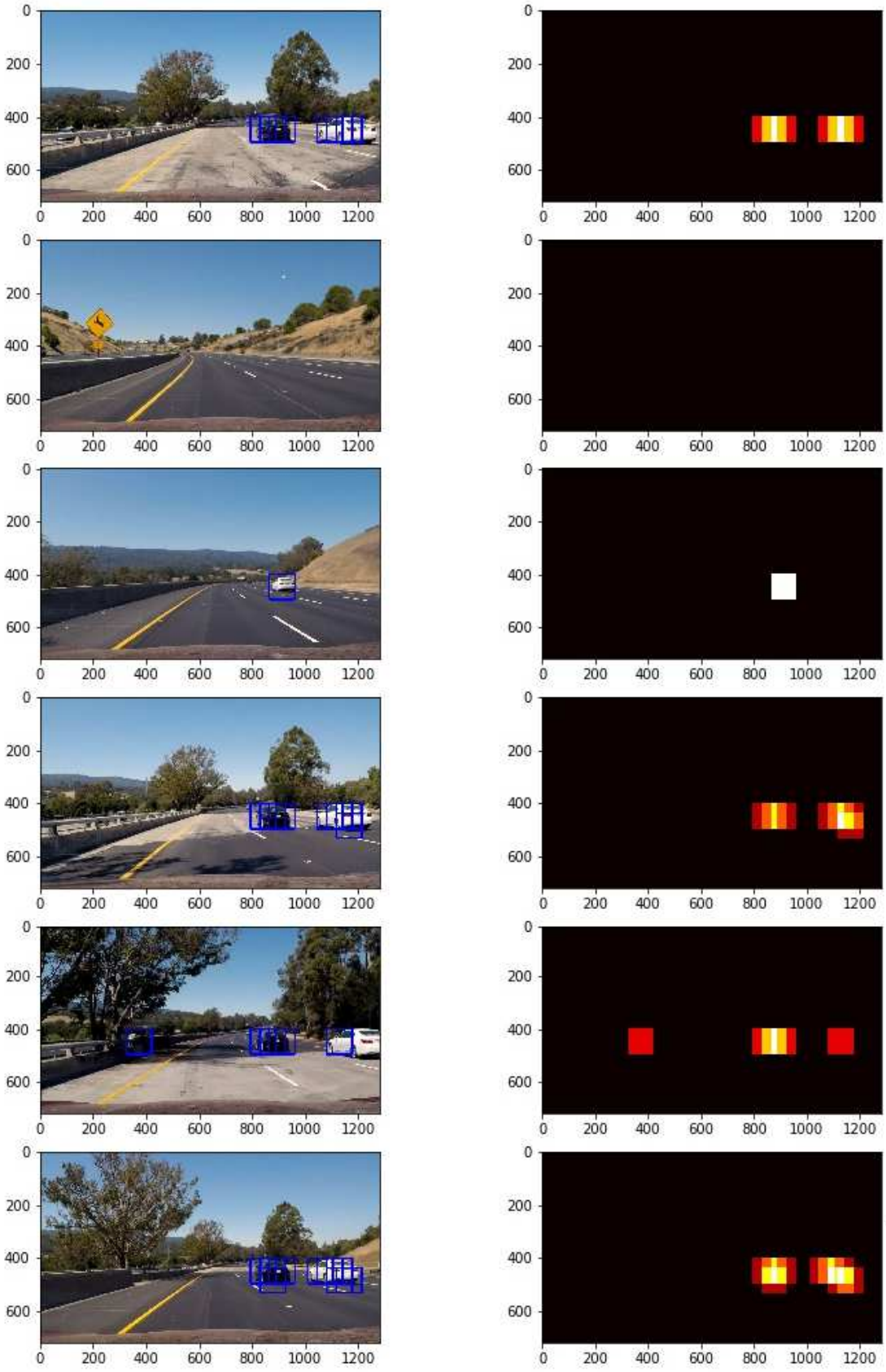
Tried with `cells_per_step = 2` would result in a search window overlap of 75% and result seems to be good enough. Vehicles are able to detect correctly on test images

One of the parameter tuned at this step is window size : Firstly tried window size 128, then 64 and finally found 96 is good value to detect all the vehicles in test images.

The SVC prediction is performed on the detected feature output from Hog Sub-sampling Window Search And created a heat-map from these detections in order to combine overlapping detections and remove false positives.

Below image is the output of above explained implementation in Cell 7, left side: Searched boxes are drawn around the detected vehicle and right side the individual heat-maps for these images.

In Next step I created above functionality as one function, `find_cars` so this functionality can be used in pipeline and additional implementation required to apply threshold to avoid false positive detections and finding boxes form the heatmap using label


0
200
400
600
800
1000
1200

Now I have all the functions to create image pipeline to run on an image or a video frame.

Image pipeline (implemented in cell 11) steps are:

1. Find Car : This function is able to both extract features using Hog sub-sampling window search, make predictions using the SVC and build heat map
2. Apply Threshhold to the heatmap to remove false positives
3. Find Boxes from heat map using scipy.ndimage.measurements label function
4. Final step, draw the labeled boxes

`find_cars` function, Parameters:

YCrb color space, which I found better color space than RGB in previous experiments on test images

Scale, tried value 1 and 1.5 and found that 1.5 scale gives better results

hog_channel , This parameter value I tried with 0 , 1 and 'ALL' and found that 'ALL' gives best results.

`find_cars` function, step:

1. Convert from RGB to YCrCb
2. Resize the image if scaling is required
3. Compute and extract the Hog features for the entire image
4. Step through the x,y position within the Hog features
5. Extract Hog patch
6. Extract image patch
7. Get color features(spatial and histogram features)
8. Scale features (spatial and histogram features and HOG features)
9. Make prediction
10. Draw box using Cv2.rectangle
11. Add heat to the heat map.

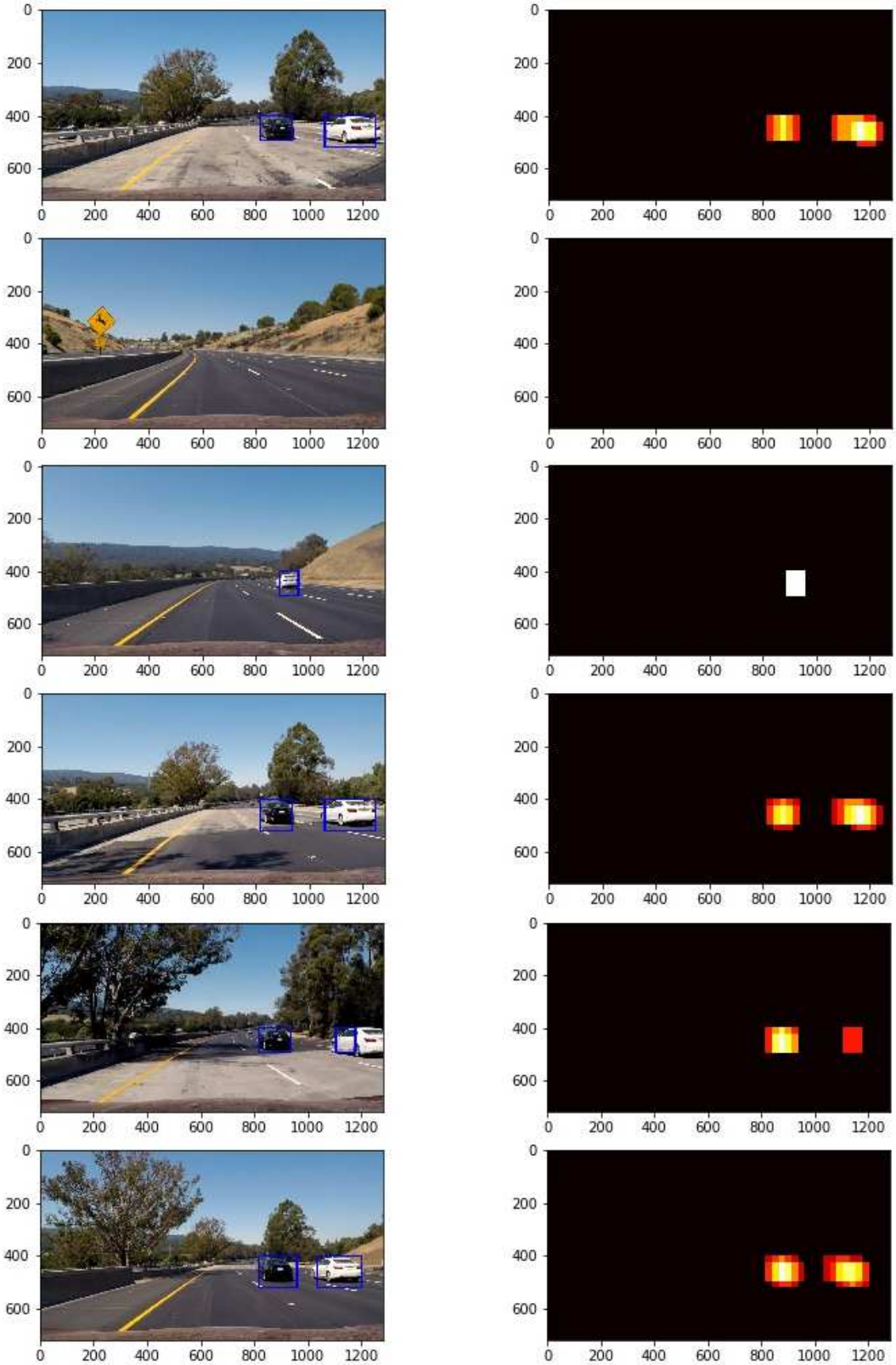


Image 5: Image pipeline result

Improvements: Created heat-map fin order to combine overlapping detections and to remove false positives, applied threshold to heatmap with value 1

Implemented two functions in cell 9, to solve problem of false positives and multiple boxes due to multiple detection

> apply_threshold : which imposes a threshhold on heatmap to remove areas affected by false positives
>
> draw_label_boxes : it takes labels image and put bounding boxes around the labeled regions, labels are generated using label function on heatmap

For the result images, refer image 5, above

## Video Implementation

Here's a link to my video result

https://github.com/anilhd2410/UdacityProjects/blob/master/CarND-Vehicle-Detection-Project/white.mp4

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.

More details of how I implemented is explained in page 6(Find car function) and above “Improvements” section

## Issues faced during implementation:

- Experimentation to find optimal color and gradient based features
- False positive detections

## Improvements and current pipe line performance:

- Recording high confidence detection and estimate subsequent frames
- Reject false detection from the estimated detections
- Taking the average of the detections  to draw the boxes to avoid flickering
- Current pipeline will fail to detect far vehicles which appears to be small and not directly on the horizon, define robust window area to be searched in the horizon