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

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

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!. All code for this project with comments are in here - CarND-Vehicle-Detection.ipynb

Histogram of Oriented Gradients (HOG)

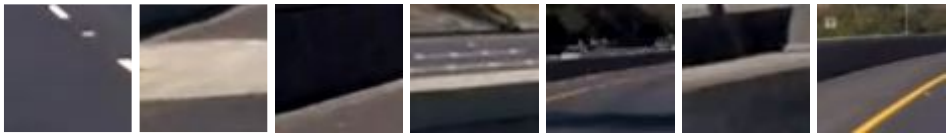
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in my jupyter notebook. You can see the function under the section *"Function to return HOG feature for image"*

This is then later used in `def extractFeatures` function while we extract the function.

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:

Vehicles - 

Non Vehicles - 

Total samples - 12,405 (non vehicles 55.4%) 9988 (Vehicles 44.6%)

The sample is a little skewed for non-vehicles but that should be fine.

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 then found that Gray scale image is the best for our purpose.

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

I tried various combinations of parameters and explored many parameters. I displayed many of the images in the jupyter notebook and examined. I also ran the SVC prediction and ran tests on same set of data for different features which gives the better test set error. I also cross checked on the test video to make sure that I didnt overfit. I found the following healthy balance

- histogram features of color space
- BIN_Spatial of RGB, HSV, HLS, YCrCb
- HOG feature of the gray image

For which I got Test Accuracy of SVC = 0.9621

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. The features are obtained through “def extractFeatures” function. This function returns an array of features using the techniques mentioned in the previous question. You can see the section “Load the training images and run LinearSVC on it” in notebook.

We use it to load the image features for all the training and testset. We create a Train/Test split of 90%/10%. The class balance for this set is

total training samples (20153) {Non-vehicles : 11153, Vehicles: 9000})
total training samples (2240, 2) Counter({Non-vehicles: 1252, Vehicles: 988})

I then ran the training and got Test Accuracy of SVC = 0.9621

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 sliding window technique was used for the following window sizes

[(48,48),(64,64),(96,96),(128,128),(160,160)]

We also eliminated a lot of nose by ignoring regions where we know in advance there wont be any cars. - x_start_stop=[800, None], y_start_stop=[360, 600]

The window overlap is 0.5

The above was arrived at after running a lot of combination of searches on test videos and final video. I iterated and tinkered around it. When I found the bounding boxes were larger than car I eliminated a few of the larger window sizes. Similarly when the car moved further away and we didnt recognize, I added smaller window sizes.

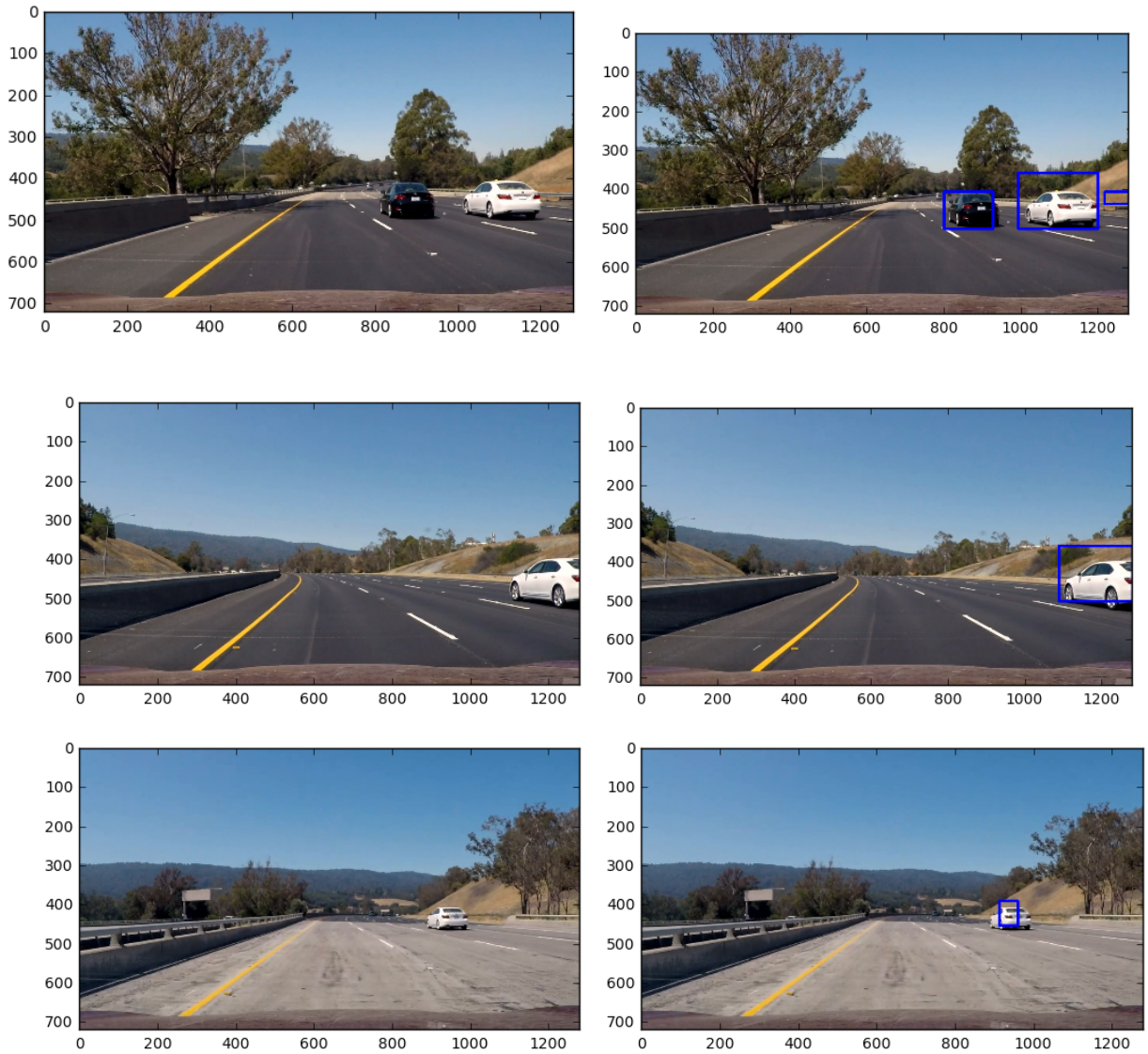
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

My pipeline uses the following feature vectors

- histogram features of color space
- BIN_Spatial of RGB, HSV, HLS, YCrCb
- HOG feature of the gray image

To optimize the performance I

- Added/collected some false positive data that It was performing badly on
- Removed features not necessary by eliminating 2 color channels
- Applied HOG on grayscale



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.)

Video outputs:

- Final project video
- Test videos - 1 , 2 , 3 , 4 , 5 , 6

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 a heatmap to track vehicles. This heat map has the following properties

```
# Subtract a standard amount from all places always to cool it over time
heatmap[heatmap > 0] -= 1.75
for box in bbox_list:
    # Add += 1 for all pixels inside each bbox
    heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 3
```

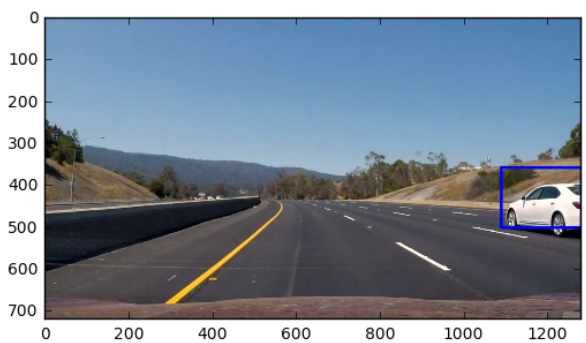
I then apply a threshold as follows

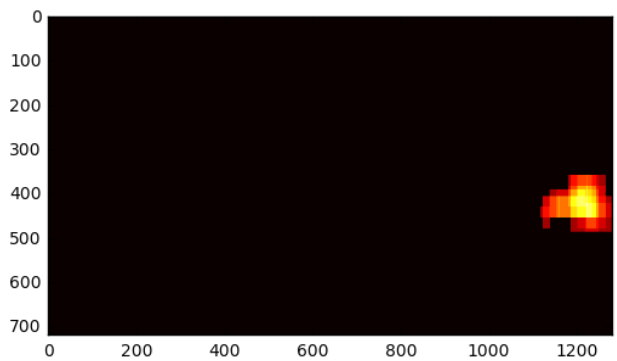
```
heatmap[heatmap <= threshold] = 0
heatmap[heatmap > threshold+200] = threshold+200
```

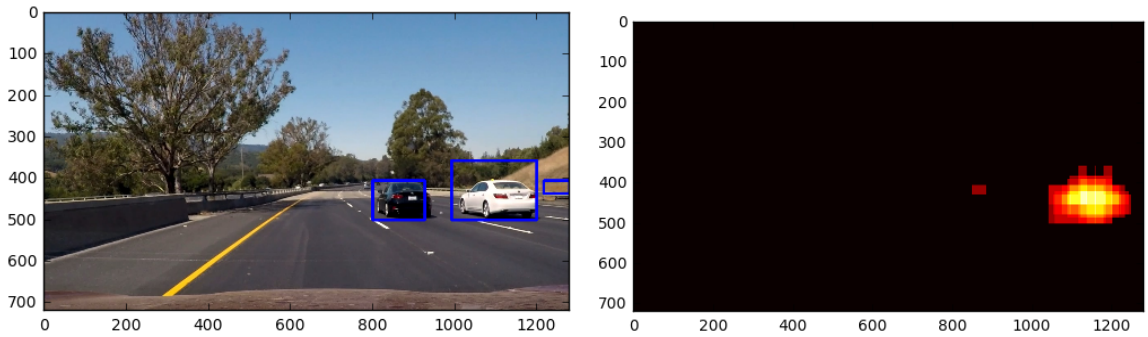
Note how there is an upper limit and we don't let it get too hot and prevent storing where the cars were historically. The threshold is set at line

```
heat_thresh = apply_threshold(heat,6)
```

Here are some heatmaps:







NOTE - The black car is small heated up but this is an example when we run a single image. When we run the video the black car gets heated up overtime and will be same as white

---

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?

1. I solved - The heat map need to be managed and cooled down. We should make sure to penalize areas where we didn't find car in a few frames.
2. I need to improve - I should calculate the centroid of a car after a few seconds of constant detection and project its next moves to make the algorithm faster.
3. I need to improve - I need to further tune how I apply penalty for heatmap. In some frames there are false positives, and also in some frame when cars are next to each other, the first few frames there is a BIG rectangle for them until the heat cools down. I need to be more intelligent about this.
4. I need to improve - I could reduce the number of features input by using a decision tree to identify which features are important