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

* Apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.

* Normalization of features are done using sklearn module's Standard Scaler.(0 mean and 1 variance)

* 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.

Output video with bounding boxes

./out_project_video.mp4

Project5.ipynb has the main code for this project. All the cell references made in the file are from the ipython notebook unless explicitly mentioned. For few experimental code, ipython notebook **Experimental-project5.ipynb** is used.

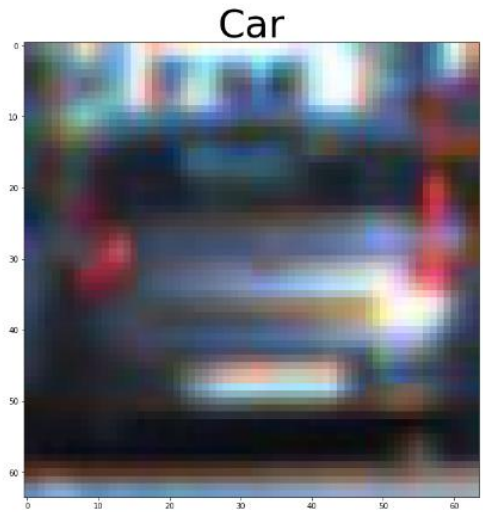
Histogram of Oriented Gradients (HOG)

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

Cell #2 contains the functions *get_hog_features()* and *extract_features()* which helps in extracting HOG features. The actual extraction of HOG features for training is done in cell #14

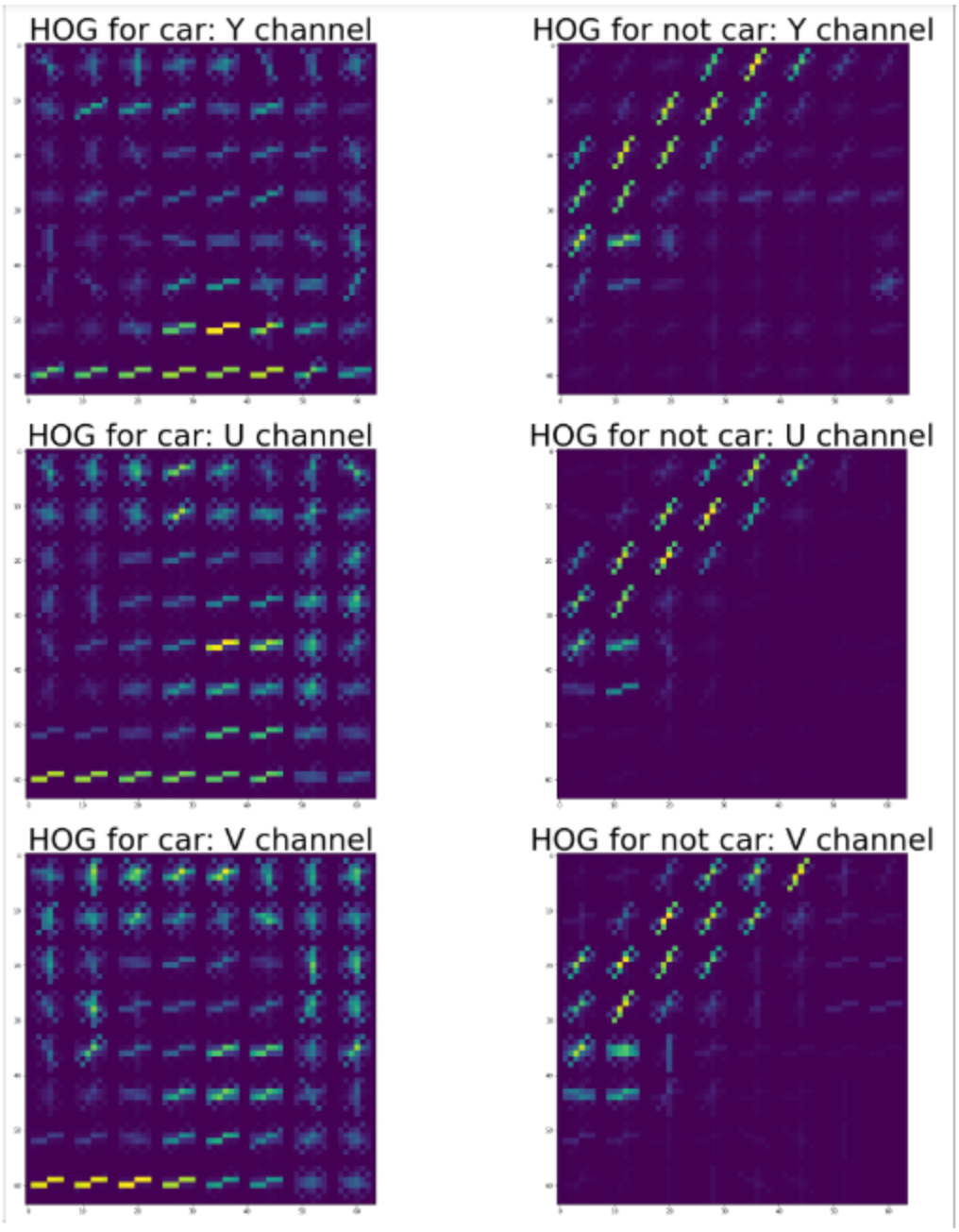
I started by reading in all the `vehicle` and `non-vehicle` images in cell #13 There were 8792 car images and 8968 non-car images.

Example car and not-car images:



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 using the `YUV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)' for the above 2 images:



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

I tried the parameters given in the Udacity lessons and the Test score was around 99% with SVC. Hence I didn't try changing the parameter a lot. I tried with orientations as 8, but didn't improve the score much. So settled with all parameters as given in the Udacity lessons.

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 SVC in cell #15. I used HOG on all 3 channels of YUV along with spatial binning and color histogram as features. My feature vector is of length 6108. I used C value of 1000 to reduce false positives.

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 do sliding window search of various scales in different regions of the image. At the end of the visible road in front I used lower scales and gradually increased the scale towards the image closer to the vehicle. The sliding windows of varying scale also overlaps.

find_car() function does the sliding window search of given scale and in given part of the image(ystart, ystop).

In the below message, window of the bounding boxes were searched and found that it contains car. It can be seen that, towards the car the bounding box size increases.

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

I optimized the feature vector computation by computing HOG for the given image fully and then sub-sampled it according to the window size. Here are some example images:

More example images can be found in cell #14. I extracted test_images[7-10] from the project_video.mp4 to test the classifier on passing by cars.

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.)

./out_project_video.mp4

The function *process_vid()* in cell #9 contains the pipeline that is to be executed for each frame of the video.

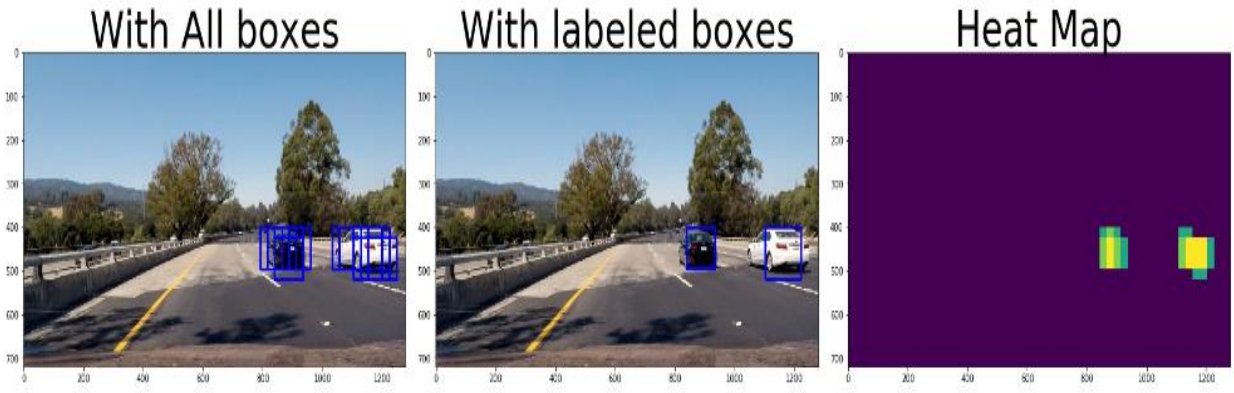
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

The "C" parameter of LinearSVC is increased to 1000 to reduce false positives. Also LinearSVC.decision_function() is used to determine whether car is detected by comparing its value to be >= 0.6. I found 0.6 to be an optimal value to avoid false negatives; for higher value than 0.6 I noticed that sometimes car was not recognized.

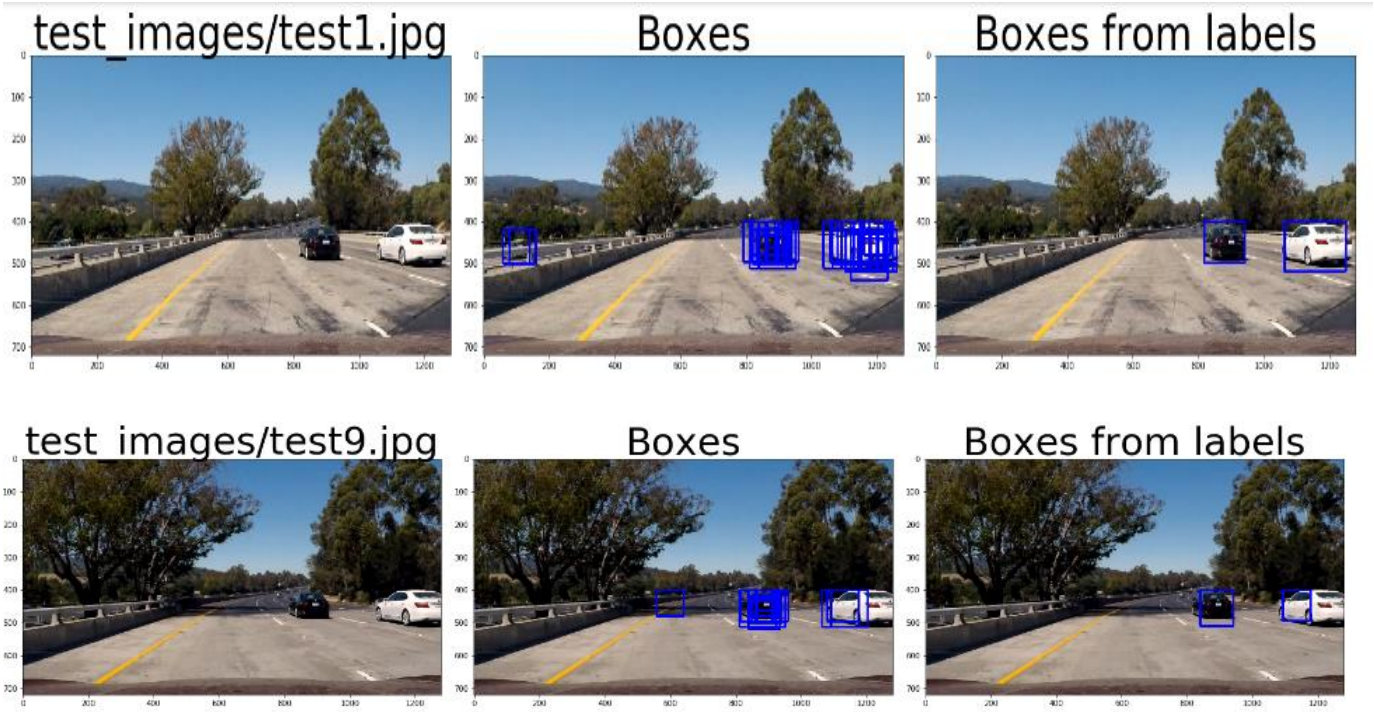
Secondly heatmap is created for the last 8 frames and a combined threshold value of 8 is used to determine the bounding boxes (using labels).In cell #9 *process_vid()* function has the pipeline to detect cars in an image.

i) Heatmap is computed by summing the heatmap for the last 8 frames on varying sizes of window on different region of the image.
ii) A threshold of 8 votes is used on the sum-heatmap.
iii) Computes labels and identifies blob after applying the threshold.(using `scipy.ndimage.measurements.label()`) on the sum-heatmap
iv) Draw bounding boxes over the detected blobs

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:



For 13 test_images, bounding boxes without filtering and with filtering can be seen in cell #8. Few examples are given below:



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?

The classifier is running very slow for me since my feature vector is of size 6108 and also I was searching with 6 different scales. Although I am searching small part of the image with each scale, it would still be lot of computation for a single frame.

I could reduce the feature vector size without losing the important information by using some machine learning technique.

The number of scales of window to search can be reduced with acceptable false positives.

Also instead of searching each frame, every alternating or even less frequent frames could be searched for and for skipped frames the previous bounding box can be just maintained or moved in the direction of car.