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. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

Done!

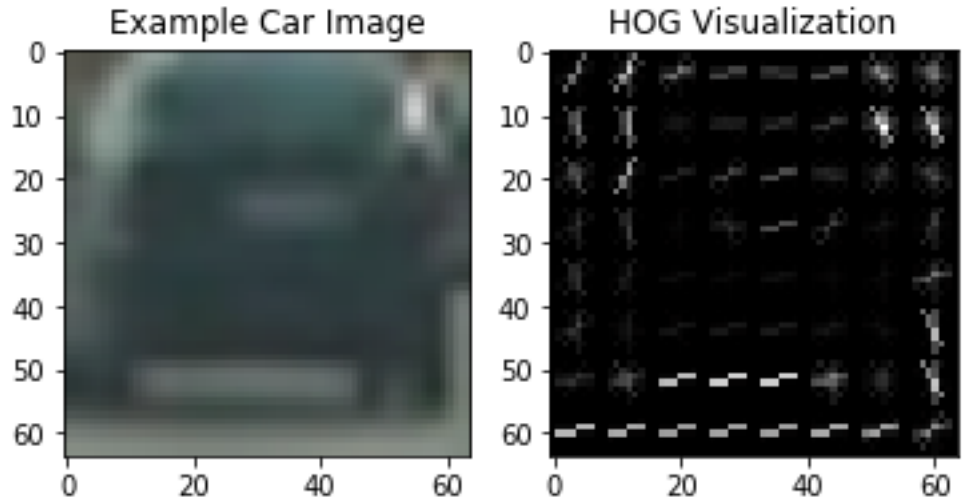
Histogram of Oriented Gradients (HOG)
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the second code cell of the IPython notebook.

The HOG code is copied from course. As taught in the course, we can directly load hog function from scikit. Moreover, the hog function provides feature_vect parameter which can be used to

normalize feature vectors. If feature_vect is set to be True, then output is directly normalized. Otherwise, the output is original extracted feature values.

I tried several color spaces, including: HSV, YUV, HLS, YcrCb. The performance of them are not showing significant discrepancy to me. But it seems the YCrCb gives me better detection on white cars.
Below is one example:



Below is the list of the parameters that were passed to hog:
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"

I played with them many times. Then I fond that pix_per_cell and orient play the most strong impact on feature extraction and later detection. If I make pxi_per_cell to large, the hog feature is very low resolution and loses too many information. If orient is too large, first takes more operation time and second, extract feature contains redundant information. The I uses all channels for hog to avoid of channel bias on features.

2. 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 HOG features, color histogram features and downsize spatial features.

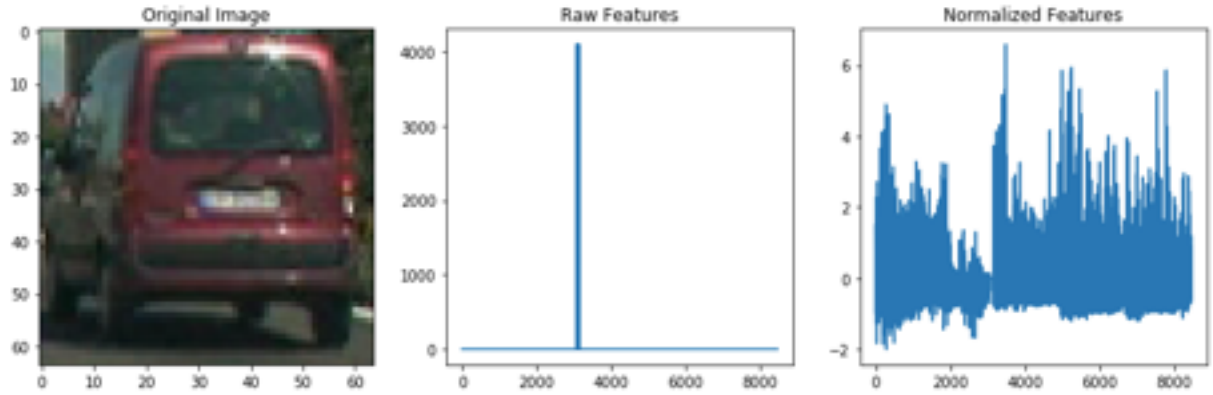
Above, I have explained how I chose parameters for HOG features. In oder to be consistent, for color histogram, I chose the same color space YCrCb as in HOG. I compared histogram bins as 16 and 32. 32 bins mean more features than 16 bins and give me better performance with the test images.

For spatial_size, original image is 64 * 64 pixels, and I still want to preserve important features, then I chose spatial_size to be 32 rather than 16.

Below is the full list of the parameters that were used to train SVM:
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
hist_bins = 32
hist_range = (0, 256)
spatial_size = (32, 32)

As we learnt from the class, the different features are not in the scale. In order to avoid one feature dominates the training results, we have to normalize the features and then concatenate them as one vector. I used StandardScaler from spleen to help me resolve this issue.

Below is one example comparing features before and after normalization. As we can see, after normalization, all the features are in the same scale. Before, features were completely biased.



Please check second, third, forth and eighth blocks for details of feature extraction and SVM models.

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?

In the video, the cars are moving from near to far another size on the image is from large to small as well. Multi-scale window sliding technique is a perfect solution to our problem. Based on the car moving trace, I cut the image into several sections based on y axis, shown as below. Meantime, I changed the image scales versus yend-ystart to capture the varying car size in the image. The total effective image y axial range is from 400 to 656. For overlapping, I followed the cause and chose 2 cell steps which gives me 75% coverage.

rectangles = []

ystart = 400
ystop = 464
scale = 1.0

```
rectangles.append(find_cars(image, ystart, ystop, scale, svc,
X_scaler, color_space, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))

ystart = 416
ystop = 480
scale = 1.0
rectangles.append(find_cars(image, ystart, ystop, scale, svc,
X_scaler, color_space, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))

ystart = 432
ystop = 496
scale = 1.0
rectangles.append(find_cars(image, ystart, ystop, scale, svc,
X_scaler, color_space, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))

ystart = 400
ystop = 496
scale = 1.0
rectangles.append(find_cars(image, ystart, ystop, scale, svc,
X_scaler, color_space, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))

ystart = 416
ystop = 528
scale = 1.5
rectangles.append(find_cars(image, ystart, ystop, scale, svc,
X_scaler, color_space, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))

ystart = 432
ystop = 512
scale = 1.5
```

rectangles.append(find_cars(image, ystart, ystop, scale, svc, X_scaler, color_space, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins))

ystart = 432
ystop = 528
scale = 1.5
rectangles.append(find_cars(image, ystart, ystop, scale, svc, X_scaler, color_space, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins))

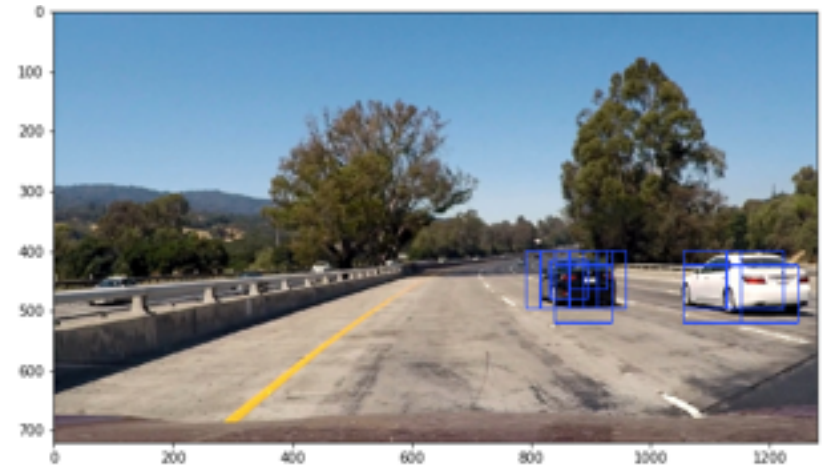
ystart = 432
ystop = 596
scale = 1.5
rectangles.append(find_cars(image, ystart, ystop, scale, svc, X_scaler, color_space, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins))

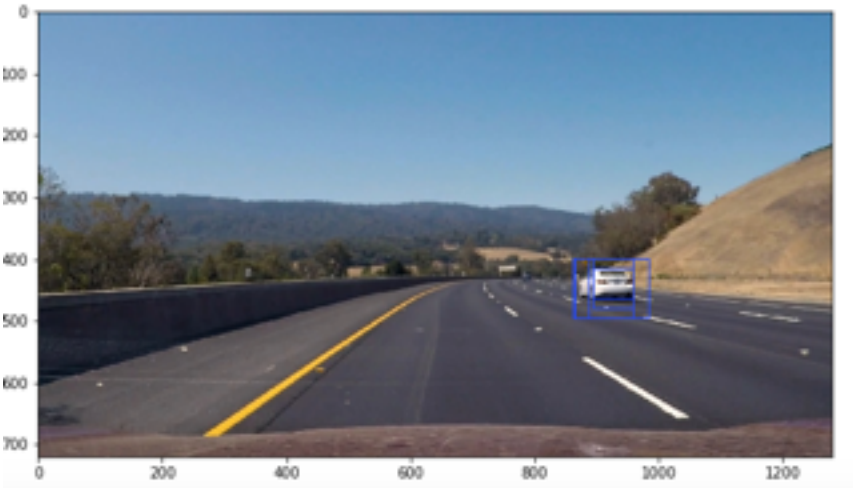
ystart = 512
ystop = 608
scale = 1.5
rectangles.append(find_cars(image, ystart, ystop, scale, svc, X_scaler, color_space, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins))

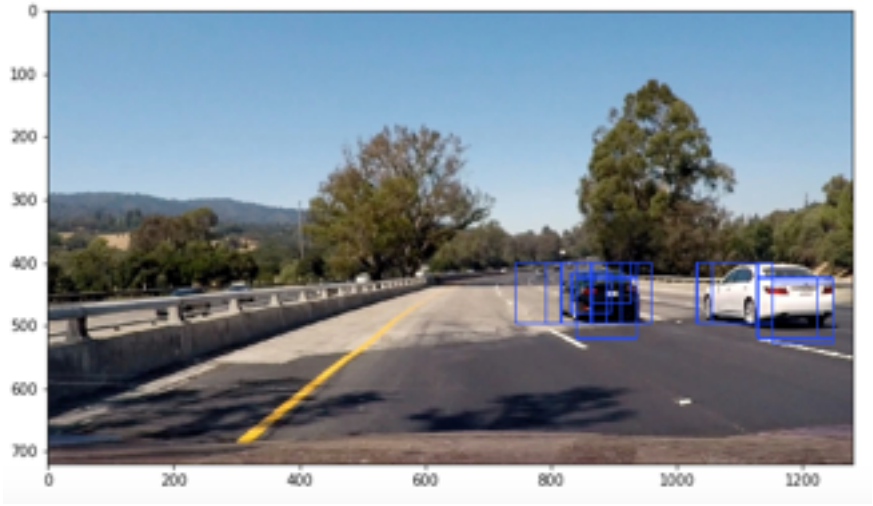
ystart = 400
ystop = 656
scale = 1.5
rectangles.append(find_cars(image, ystart, ystop, scale, svc, X_scaler, color_space, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins))

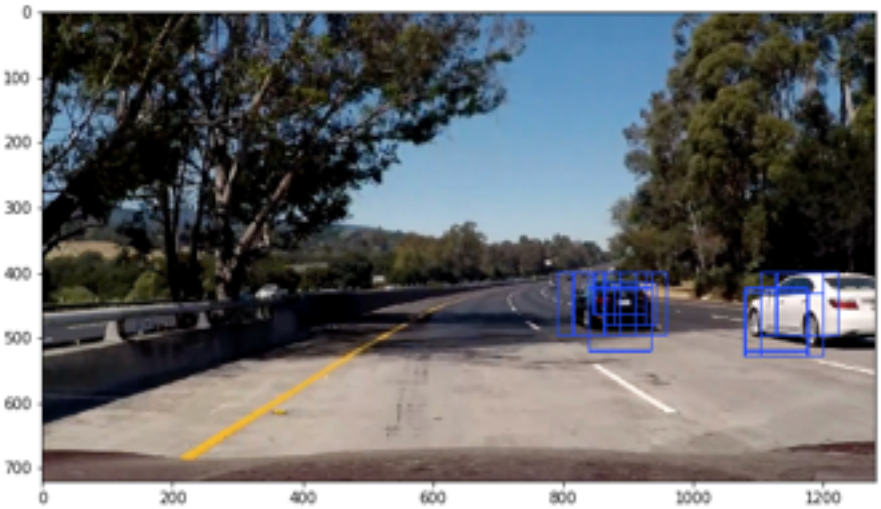
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on two scales versus nine sub windows 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.)

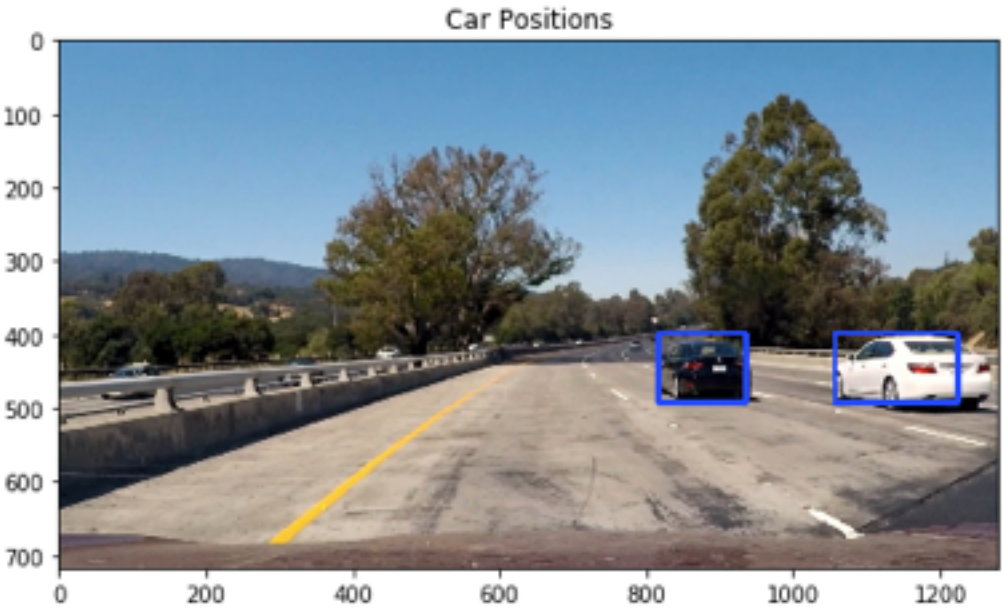
Project_video_out is attached.

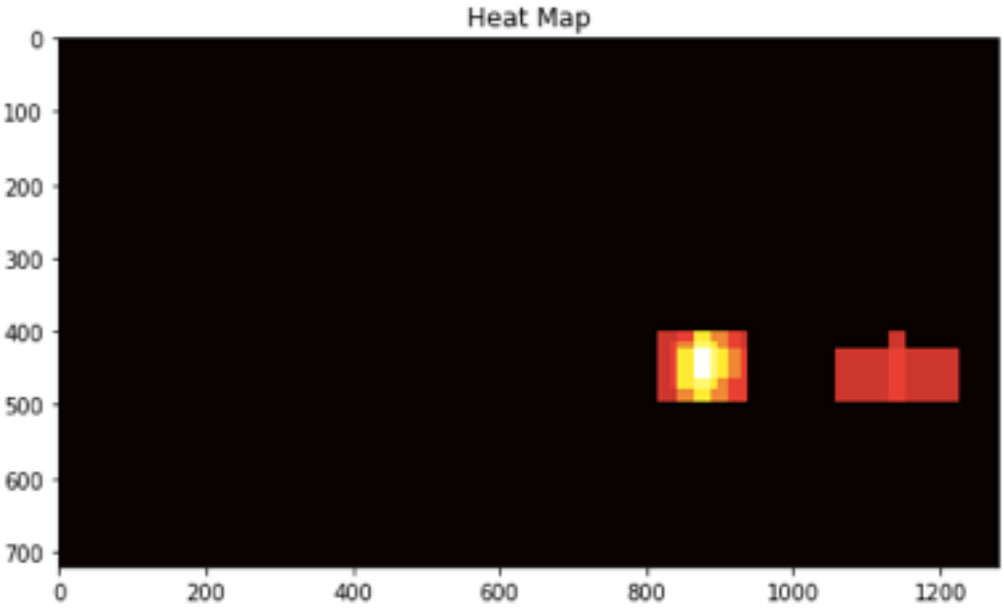
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. The threshold is 1 after testing with test images.

I then used scipy.ndimage.measurements to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. Then blobs are labeled. I constructed bounding boxes to cover the area of each blob detected.

Here's an example showing result after hot map filtering:





In the pipeline, I was inspired by Jeremy Shannon. He created a vehicle class which saves most recent 15 car rectangular sets after hot map filtering, which was similar to project 4. This approach connects images in a short time period and further filters transient false positive together with hot map approach.

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 major problem I feel in this project is how to effectively filter false positives. Though hot map is applied here, if the video contains multiple cars driving in the wrong direction one image, once those cars are grouped, the hot map would not be able to remove them. In this case, I am thinking of improving the vehicle class by comparing the center of the current rectangular and previous rectangular from earlier time. If the center is moving forward, that's right positive and keep it. If the center is moving backward, that means it is from the opposite direction. I would remove it. But in the current video, the overall image quality is good. Will include it in the future work.