Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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 : https://review.udacity.com/#!/rubrics/513/view

1. Histogram of Oriented Gradients (HOG)

1.1 Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

I use scikit-image library. it hads a built in fuction to extract Histogram of Oriented Gradient(HOG) features. We can load the hog fuction from scikit-image library.

```
from skimage.feature import hog
```

This site explain about the hog function. It extracts features like bellow. HOG feature descriptor is

popular for object detection.

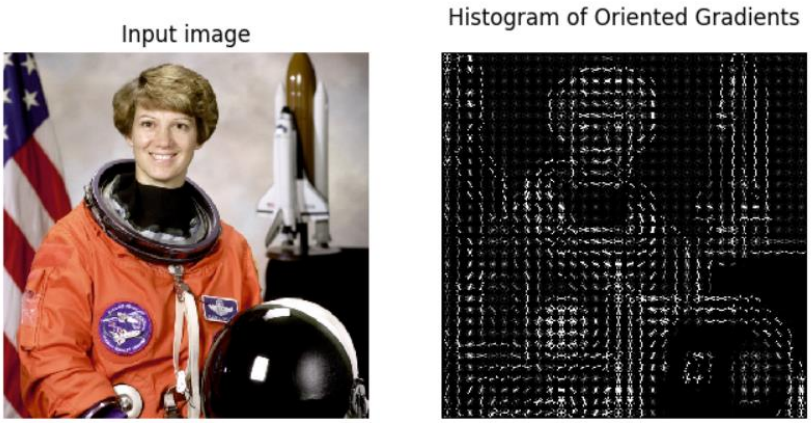

Figure 1 http://scikit-image.org/docs/dev/_images/sphx_glr_plot_hog_001.png

Hog function use parameters as orientations, pixels_per_cell and cells_per_block. And I use all channels on final training. The result is better than a channel.

Parameters	Value	explains
orientations	9	The number of orientations is specified as an integer, and represents the number of orientation bins that the gradient information will be split up into in the histogram. Typical values are between 6 and 12 bins.
pixels_per_cell	8	The pixels_per_cell parameter specifies the cell size over which each gradient histogram is computed. This paramater is passed as a 2-tuple so you could have different cell sizes in x and y, but cells are commonly chosen to be square.
cells_per_block	2	The cells_per_block parameter is also passed as a 2-tuple, and specifies the local area over which the histogram counts in a given cell will be normalized. Block normalization is not necessarily required, but generally leads to a more robust feature set.

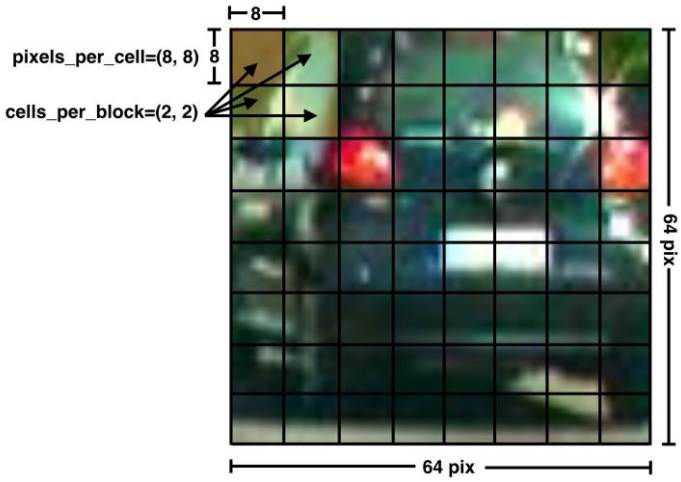

Figure 2 Reference by [20. scikit-image Hog Udacity lecture]

1.2 Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

At first I tested C parameter used in the svc algorithm. The result is below.

	0	1	2	3
mean_fit_time	11.0952	4.82659	5.42269	6.07546
mean_score_time	0.0865533	0.0630008	0.0847253	0.0952532
mean_test_score	0.989443	0.989443	0.989443	0.989443
mean_train_score	1	1	1	1
param_C	1	10	100	1000
params	{'C': 1}	{'C': 10}	{'C': 100}	{'C': 1000}
rank_test_score	1	1	1	1
split0_test_score	0.990921	0.990921	0.990921	0.990921
split0_train_score	1	1	1	1
split1_test_score	0.988809	0.988809	0.988809	0.988809
split1_train_score	1	1	1	1
split2_test_score	0.988598	0.988598	0.988598	0.988598
split2_train_score	1	1	1	1
std_fit_time	8.18497	0.273263	0.480257	0.821062
std_score_time	0.0408805	0.00272625	0.0147532	0.0226916
std_test_score	0.00104868	0.00104868	0.00104868	0.00104868
std_train_score	0	0	0	0

Figure 3 Linear SVC C parameter test result

The results are all the same. Only results for time are different. So I use 10 to C. It has the best

time.

And I tested all channels and color spaces. All Channels is the best in most cases. The best of color space is HLS and HSV, but they detect too much of the shadow as positive. YCrCb has next best score. I choose LUV color space.

	0	1	2	ALL
HLS	0.9538	0.9820	0.9544	0.9918
HSV	0.9592	0.9640	0.9792	0.9918
LUV	0.9789	NaN	NaN	NaN
RGB	0.9707	0.9747	0.9744	0.9806
YCrCb	0.9772	0.9581	0.9566	0.9893
YUV	0.9747	0.9592	NaN	NaN

Figure 4 test color channels and color spaces test result

The images convert to YCrCb color spaces and extract the HOG features.

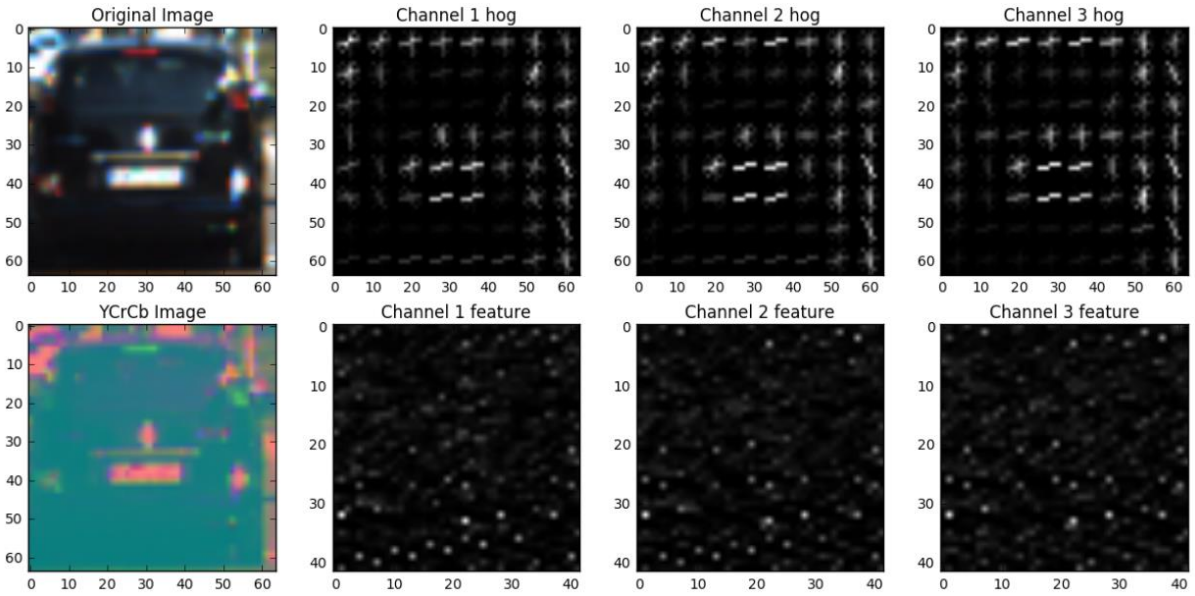

Figure 5 YCrCb color spaces and extract the HOG features

2. Sliding Window Search

2.1 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 defined the find_cars function to return boxes and choose some different image sizes. While I using one box size, it miss low size. If I only use low size, it has too much false positive. Finally I decided 3 size for detecting the images 64x64(8*8), 48x48(8*6), 80x80(8*10) with heatmap.

```
boxes = []
boxes += find_cars(img, xstart, xstop, 400, 656, scale, svc, X_scaler, orient, 8, cell_per_block, spatial_size, hist_bins)
boxes += find_cars(img, xstart, xstop, 400, 500, scale, svc, X_scaler, orient, 6, cell_per_block, spatial_size, hist_bins)
boxes += find_cars(img, xstart, xstop, 400, 656, scale, svc, X_scaler, orient, 10, cell_per_block, spatial_size, hist_bins)
```

Figure 6 find_cars function detect and return boxes

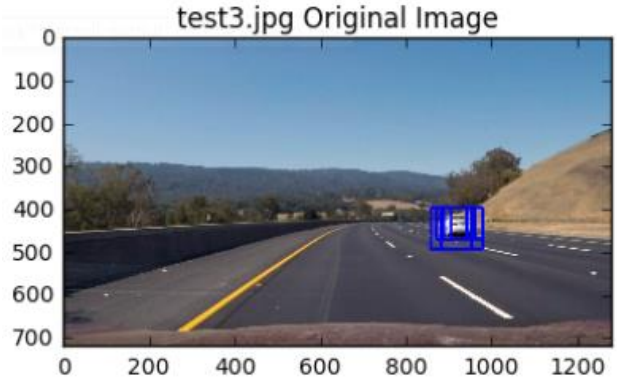

Figure 7 small size windows is needed

Figure 8 Big size windows is needed

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

This detector sometimes falsifies the image. So we use heatmap. A lot of windows is overlapped. If one point is detected in many cases, this is very likely to be properly detected. At first, draw a heatmap and apply a threshold.

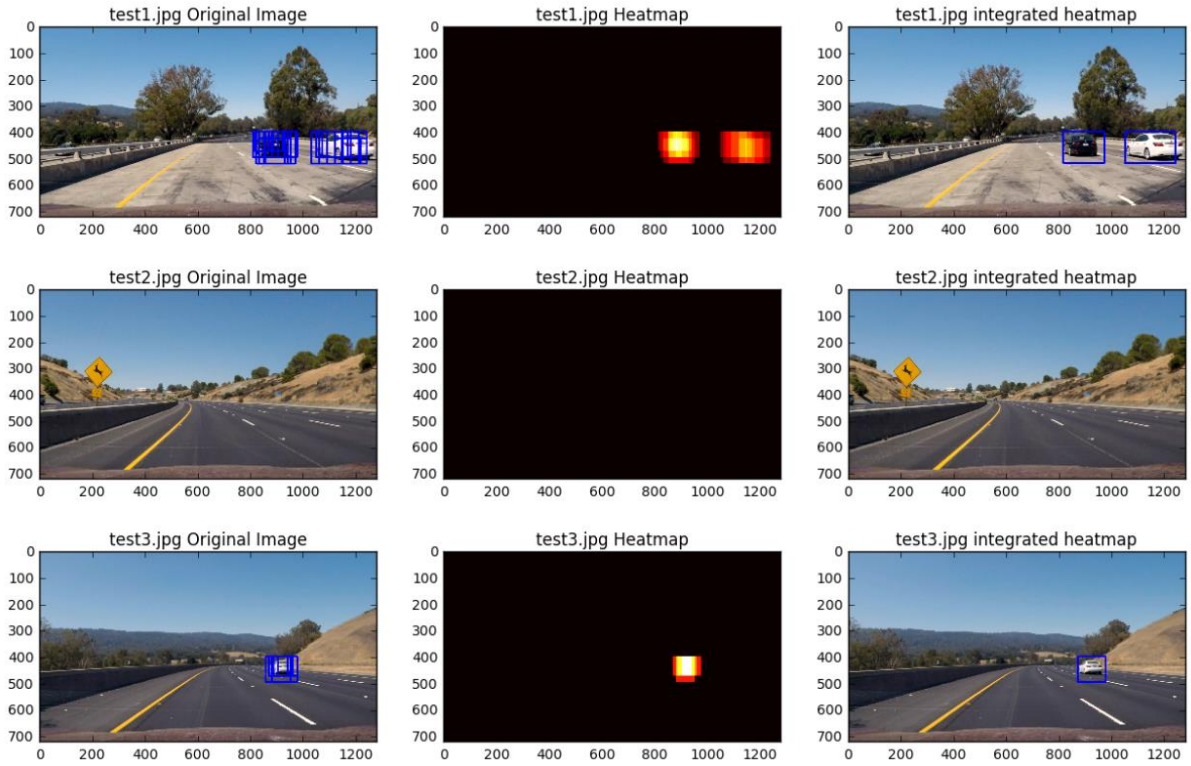

Figure 9 row boxes(left), heatmap(center), applying threshold(right)

3. Video Implementation

3.1 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.)

The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

3.2 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.

A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

The process_image function is the last function for videos. This function save history of images

before 8 frames using que. After take average from the history, It draw the boxes. Because use history, it can reduce false positives.

```
# Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML

history = deque(maxlen = 8)

def process_image(img, threshold=1):
    boxes = []
    boxes += find_cars(img, xstart, xstop, 400, 656, scale, svc, X_scaler, orient,
                               8, cell_per_block, spatial_size, hist_bins)
    boxes += find_cars(img, xstart, xstop, 400, 500, scale, svc, X_scaler, orient,
                               6, cell_per_block, spatial_size, hist_bins)
    boxes += find_cars(img, xstart, xstop, 400, 656, scale, svc, X_scaler, orient,
                               10, cell_per_block, spatial_size, hist_bins)

    draw_img = draw_boxes(img, boxes, color=(0, 0, 255), thick=6)

    heatmap = draw_heat(img, boxes)

    history.append(heatmap)
    history_average = np.average(history, axis=0)
    history_average[history_average <= threshold] = 0

    labels = label(history_average)
    result = draw_labeled_bboxes(np.copy(img), labels)

    return result

white_output = 'convert_test_video.mp4'
clip1 = VideoFileClip("test_video.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects color images!!
%time white_clip.write_videofile(white_output, audio=False)
```

Figure 10 The process_image function

4. Discussion

4.1 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?

4.2 Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

Like P4, it can be mistakenly recognized in case of shadows, trees, people, objects with a lot of noise. Remove noise as much as possible and add more notcar data. In addition, it is extremely rare for a vehicle to appear suddenly, so it is possible to record past values and apply appropriate algorithm values to minimize the sudden change in value. If it is difficult to handle false positives, it may be a good idea to display a window of another color that offers a possibility.