Vehicle Detection

Files Submitted & Code Quality

1. My project includes the following files:

- **Vehicle_detect.ipynb** iPython notebook containing the scripts to get various image features, train a classifier – linear SVM, generate search windows, detect vehicles in an image and in a video frame, etc.
- **output_images** folder containing images obtained at various stages in an algorithm
- **writeup_report.pdf** summarizing the results
- **project_video_out.mp4** video showing my final result

Histogram of Oriented Gradients (HOG)

2. 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 **first code cell** of the IPython notebook – the '**get_hog_features**' method which is the same as the one in class lessons. It extracts the HOG features based on the inputs – orient, pixels per cell, cells per block, transform square root.
- The **first code cell** of the IPython notebook – the '**extract_features**' method which is the same as the one in class lessons. It extracts different image features based on user choice: spatial features, histogram features, or HOG features, or all of these.
- In the **fifth code cell** of the Ipython notebook I am extracting the HOG features from the training data sets.
  And example HOG image is shown below for a random car image and a non-car image using following settings:
  Color space: Gray image
  Orient: 11
  Pixels per cell: 16
  Cell per block: 2

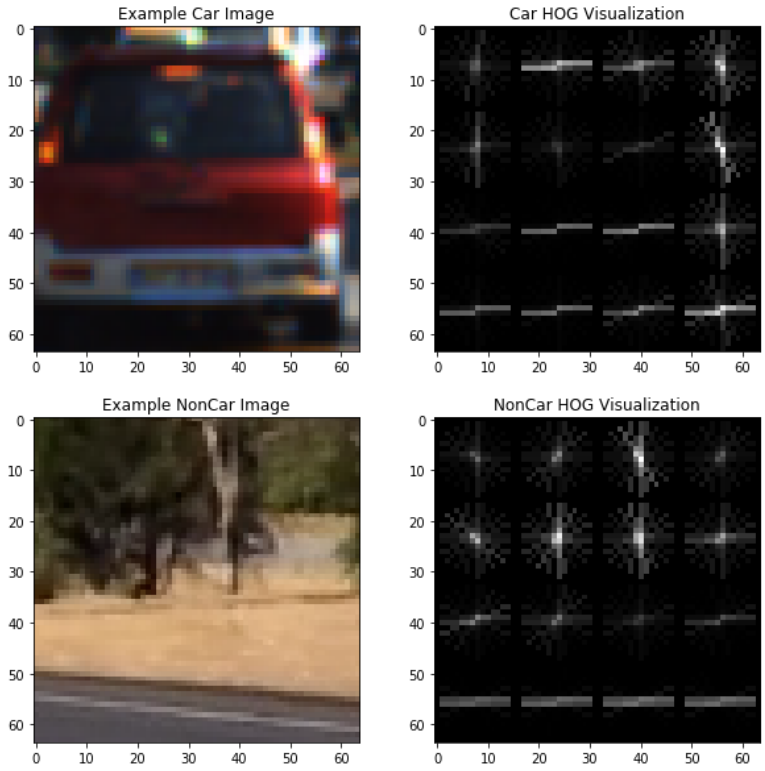

Figure 1: Example HOG image

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

I tried the following combinations for each of the input variables for the HOG function:

Color spaces: RGB, HSV, LUV, HLS, YUV, YCrCb

Orient: 9, 10, 11, 12

Pixels per cell: 8, 12, 16

Cells per block: 2

HOG channels: 0, 1, 2, ALL

Then I checked which of the above combinations gives maximum test accuracy while maintaining moderate feature extraction time and training time. The setting - YUV color space, orient = 11, "ALL" HOG channels, was always giving the best test accuracy (more than 98%).

The feature extraction time and training time was reasonable while still maintaining the test accuracy > 98% for the setting - Pixels per cell: 16, Cells per block = 2.

Therefore my final choice is as follows:

Color spaces: **YUV**

Orient: **11**

Pixels per cell: **16**

Cells per block: **2**

HOG channels: "**ALL**"

Final test accuracy: **98.56%**

Final feature extraction time for the training and test data: **184.79 s**

Final training time is **3.3 s** for the training data and **0.01101 s** to predict 20 test data.

4. **Note:** I have also tried testing the linear SVM classifier by using other two features such as spatial color (**bin_spatial** method) and color histogram (**color_hist** method) along with the HOG features. When I used them, I tried with different spatial dimentions and histogram bin sizes. I used feature scaling (feature normalization) when different types of features were combined in the feature vector. However, despite adding additional features types (spatial color and histogram features), I did not get any significant improvements in my test accuracy. It still remained at around 98% while taking more time for feature extraction and prediction. There was no improvement in test accuracy but execution time was huge. So I decided to use **only** the **HOG** features (with settings discussed earlier) so the video frames can be processed at faster rates. Since I use only HOG features for all my future analyses, I have not used feature normalization. However this option is kept in the code, if one wants to append other feature types feature normalization will be enabled (see cell 5 and cell 7).

5. 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 have trained using only the HOG features (reasons stated above) with the help of linear SVM classifier (cell 6)

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 did the sliding window approach as recommended in the lesson. But the approach is slightly different. It is explained below and is conducted for every time frame.

A. The **find_cars** (cell 7, is almost same as the one in class lesson) method actually does most of the job here. It takes the following parameters as inputs and looks for cars/parts of cars in a moving window of different scales scanned across the incoming image.
    a. Img: input image
    b. Ystart: the starting pixel in y direction, this is where the box search will begin in Y axis
    c. Ystop: the stopping pixel in y direction, this is where the box search will stop in Y axis
    d. Xstart: the starting pixel in x direction, this is where the box search will begin in X axis
    e. Ystop: the stopping pixel in x direction, this is where the box search will stop in X axis
    f. Scale: the size of the box
    g. SVC: trained classifier that detects in a box if any car/car parts are present
    h. X_scaler: the feature normalizer
    i. Hog features: orient, pixels per cell, cells per block, hog_channels
    j. Spatial_size: in one wants to use color spatial features
    k. Hist_bins: Histogram bin sizes
    l. Color_space: color transformation (chosen: RGB to YUV)
    m. spatial_feat: Boolearn – to generate spatial features or not
    n. hist_feat: Boolearn – to generate histogram features or not
    o. CellsPerStep: Decides the resolution at which search boxes will be moved across the image

p. all_search_Boxes: Boolean, when 'True' reports all the search boxes, else reports only those boxes where car/car parts are found

B. To begin with, I tested with ystart = 400, ystop = 660. This avoids unnecessary search in y-direction above the horizon and near the hood of user's car. Also, in order to save search time in x-direction, I made an assumption that there will be some sensor in the system that detects the lane in which the car is currently going. If the car is in left-most lane, then xstart can be 300 and xstop can be until the end of the frame in x-diretion. Likewise, if the car is in the right-most lane, then xstart can be 0 and xstop can be image.shape[1]-300. If the car is in middle lanes, then I can search in entire x-direction (xstart = 0, xstop = image.shape[1]).

Anyway, since I know that for the given video, car is in the left-most lane, I set xstart as 300 and xstop as image.shape[1] (until the end of the frame in x-direction). This largely saves the execution time since I don't have to look for on-going traffic in my left side.

I kept the scale as 1.5, and I got search box results as follows for the input image test1.jpg:

Figure 2: Without Window Subsampling Test1, ystart = 400, ystop = 660, xstart = 300, xstop = image.shape[1], scale = 1.5, Cellsperstep = 1

C. Next I did the moving window search with different ystart, ystop and scale values as indicated in Cell 9 of the iPython notebook. I tried several combinations (other combinations are shown in Cell 9), finally settled with the following:

(cell 9 of the code)
# My final choice...
ystart =[400, 405, 400, 405, 400, 405, 400, 405, 400, 405, 400, 405, 400, 405, 400, 405]
ystop = [500, 505, 520, 525, 540, 545, 580, 585, 580, 585, 600, 605, 640, 645, 660, 665]
scale = np.array([1.0, 1.0, 1.2, 1.2, 1.4, 1.4, 1.6, 1.6, 1.8, 1.8, 2.0, 2.0, 2.2, 2.2, 2.5, 2.5])

As you can see, there are 16 different combinations of ystart, ystop and scale values.

This search involved a total of 1172 box searches, and they look like this:

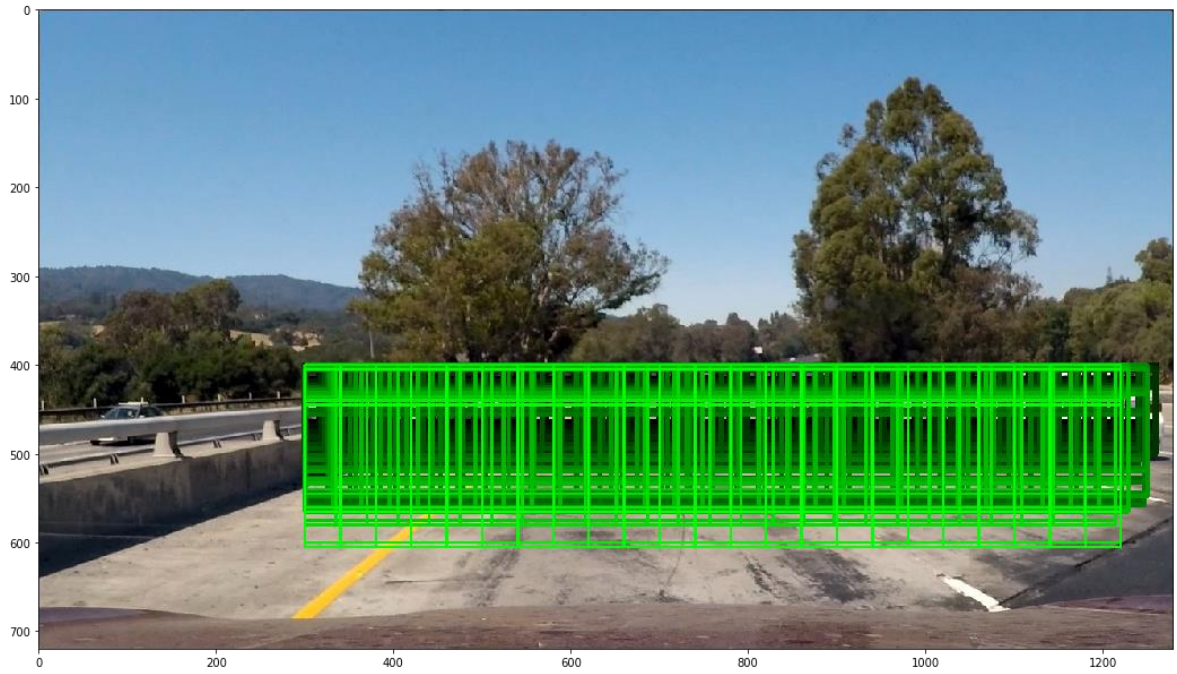

Figure 3: Subsampling Search Windows

D. With moving window search, I got the result as below for test1.jpg. It gave 114 boxes with possible locations of the car

Figure 4: With Window Subsampling Test1

2. Show some examples of test images to demonstrate how your pipeline is working.

The pipeline:

A. Obtain sub-sampled window search as discussed above (output looks like the one in Figure 4)
B. Next get the heatmap image (Cell 11)

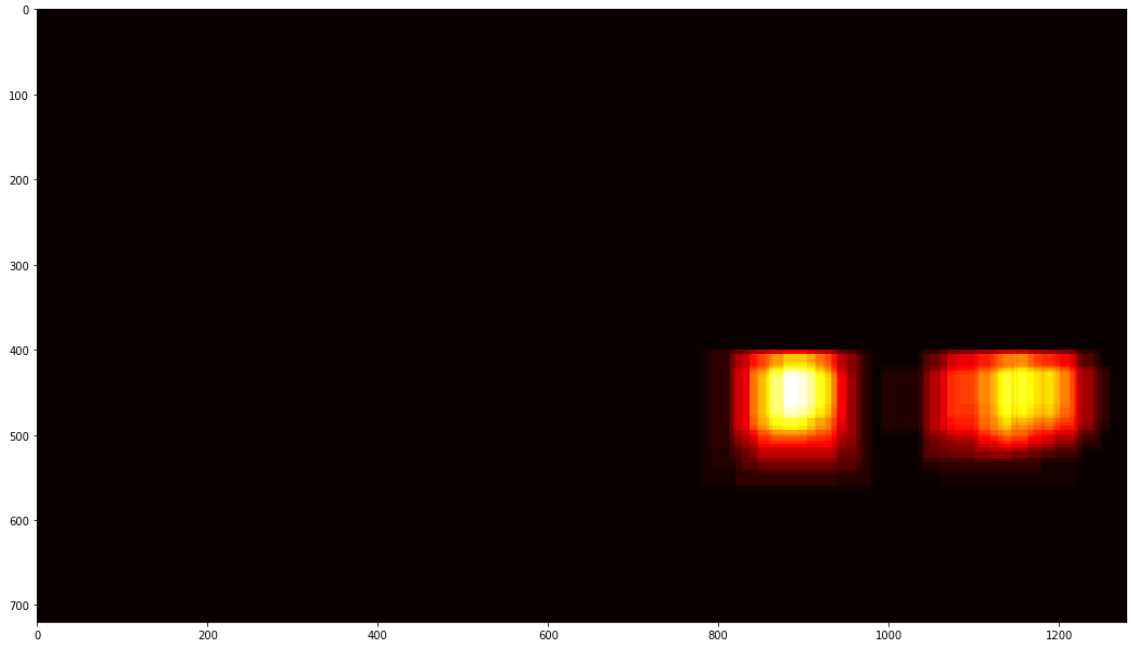

Figure 5: With Window Subsampling_HeatMap Test1

C. Apply threshold for the heatmap (Cell 13). Actually this stage needed some tuning to adjust the threshold value. I tried several different values on many video frame images and finally settled with Threshold value 3.

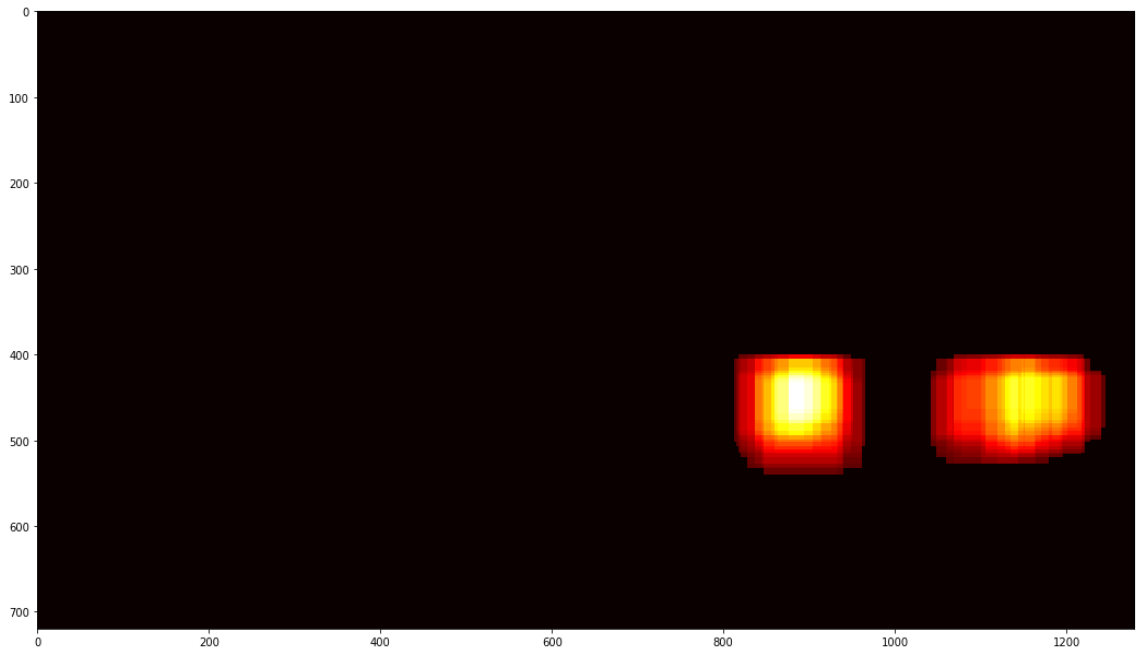

Figure 6: With Window Subsampling_HeatMapThresholding Test1

D. Then get the labeling for the above image (Cell 15)

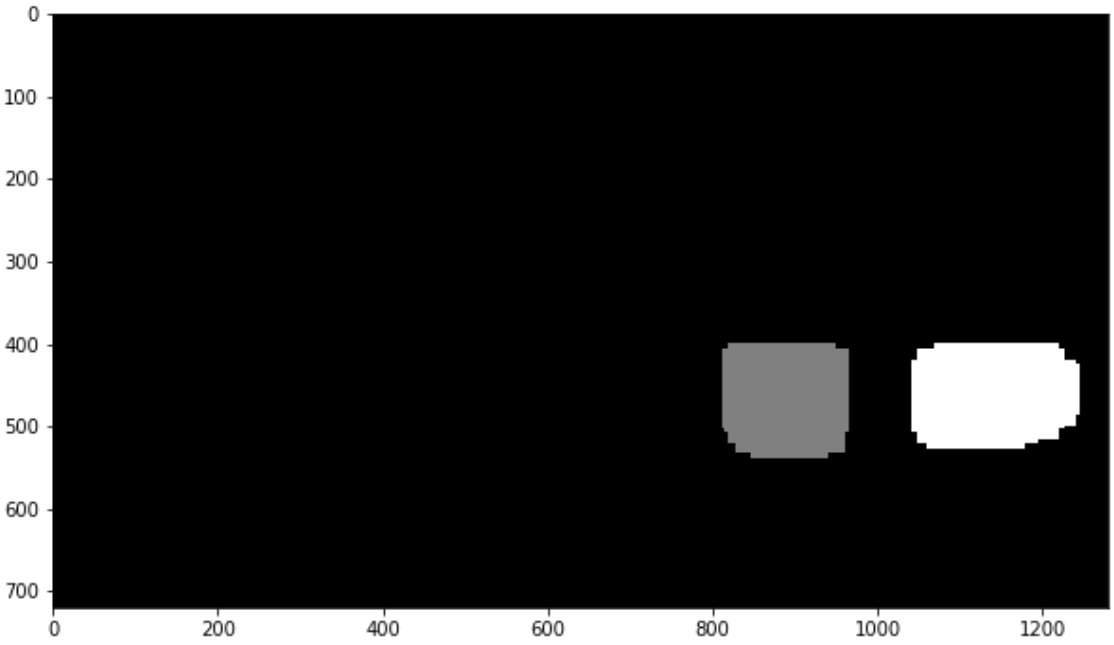

Figure 7: With Window Subsampling_HeatMapThresholding_Labeling Test1

E. Finally labeled boxes are applied on the labeled image (cell 18). There is some modification done for the method 'draw_labeled_bboxes' in Cell 18 for process the video frames. This will be explained later.

My final car detection for the test1.jpg image looks like this:

Figure 8: Car Detection With Window Subsampling_HeatMapThresholding_Labeling Test 1

Ultimately I searched on 16 different scales using YUV 3-channel HOG features only, which provided a nice result. Here are some example images:

Figure 9: Car Detection With Window Subsampling_HeatMapThresholding_Labeling Test 1 to Test 6

Video Implementation

1. A video is provided along with this submission. In the video, cars are detected successfully in 99% of the time frames. Wobbling of the detected boxes is reduced by 'box smoothing' which will be explained below. False positives are too low since I have adjusted the parameters such as heatmap threshold, box search area, features etc. In the video you can see false positives in only couple of time frames out of the total 1261 time frames.

2. The pipeline for the video processing is in Cell 23, which calls the method '**process_one_frame**' for every time frame. The '**process_one_frame**' is in cell 19 where the following steps are performed for all the consecutive frames:

    a. Do the sub-sampled box search on the entire image and get all boxes where cars/ car parts may be present
    b. Get the heat map
    c. Apply threshold for the heatmap to remove false positives. Threshold is adjusted and set as 3.
    d. Get labeling for the filtered heatmap
    e. Then pass the labels and corresponding locations to the method '**draw_labeled_bboxes**' in cell 18. Here is where I made some changes to smoothen the wobbling of boxes.

    Box smoothing: Here I will save the boxes (where cars are detected) for each vehicle in 5 consecutive frames. I have defined a class '**Detected_Boxes**' to store the detected boxes (cell 16).
    For every new detection I delete the saved box 5 frames ago, so that at any time I will have 5 boxes from current and past 4 frames, per vehicle. I will then average out (x,y) coordinate values of the 2 diagonal corners of these 5 boxes, per vehicle. The averaged corner will be then be plotted on the current frame. I then saved the 'smoothed box' for each vehicle, so that it can be used for averaging in the net time frame and so on. This way, the wobbling of frames is reduced and you can see a smooth movement of the boxes of every vehicle.