Vehicle Detection and Tracking

Done by: Wael Farag

1. Training Data Preparation

The data preparation can be summarized as follows:

1. **Udacity Supplied Data**: Only the Udacity supplied data have been used throughout the project. The data consists of almost balanced "non-vehicles" and "vehicles" images:
    a) The "non-vehicles" collections consists of the "GTI" collection and the "Extras". Both contains 8968 RGB images of size (64, 64, 3).
    b) The "vehicles" collections consists of the "GTI" collection and the "KITTI". Both contains 8792 RGB images of size (64, 64, 3).
2. These collections with an unzipped size of 149MB.
3. **Data Augmentation**: The data is augmented by flipping all the images around the "Y" axis. As a result, the training data become a total of 35,520 images. The loading and flipping the images can be found in "P5 VehicleDetect8.py" lines 55 => 88.

2. Training Data Visualization

The following steps describes the implemented data visualization steps in order of execution:

1. **Display of Vehicles Data**: 50 randomly selected images of the vehicle data have been displayed as shown in Figure 1. Each image has its order in the training data as a title. The code for this part can be found in "P5 VehicleDetect8.py" lines 87 => 124.
2. **Display of Non-Vehicles Data**: 50 randomly selected images of the non-vehicle data have been displayed as shown in Figure 2. Each image has its order in the training data as a title. The code for this part can be found in "P5 VehicleDetect8.py" lines 87 => 124.
3. **Display of HOG features of Vehicles Data**: A selected image of the vehicle data have been supplied to the "get_hog_features()" function to extracts its hog features after converting it to gray scale. The result is shown in Figure 3. The code for implementing the "get_hog_features()"can be found in "P5 VehicleDetect8.py" lines 126 => 148. The code for this part can be found in "P5 VehicleDetect8.py" lines 151 => 195.

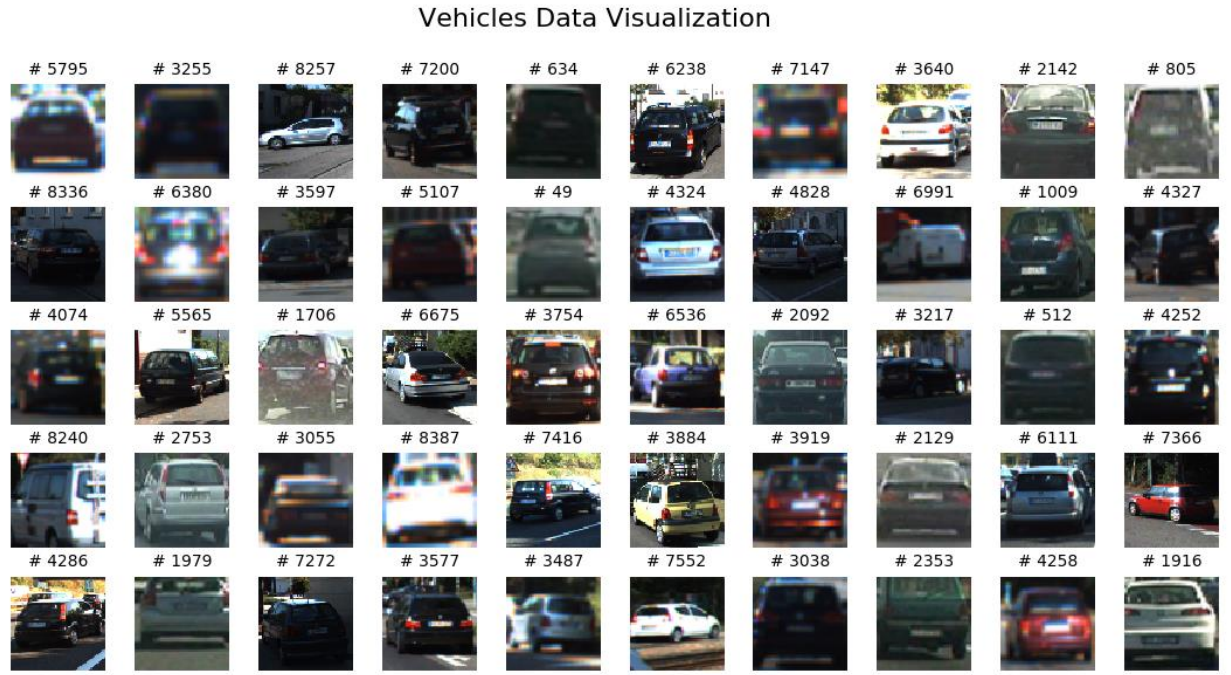


Figure 1. Visualization of 50 randomly selected vehicle images.

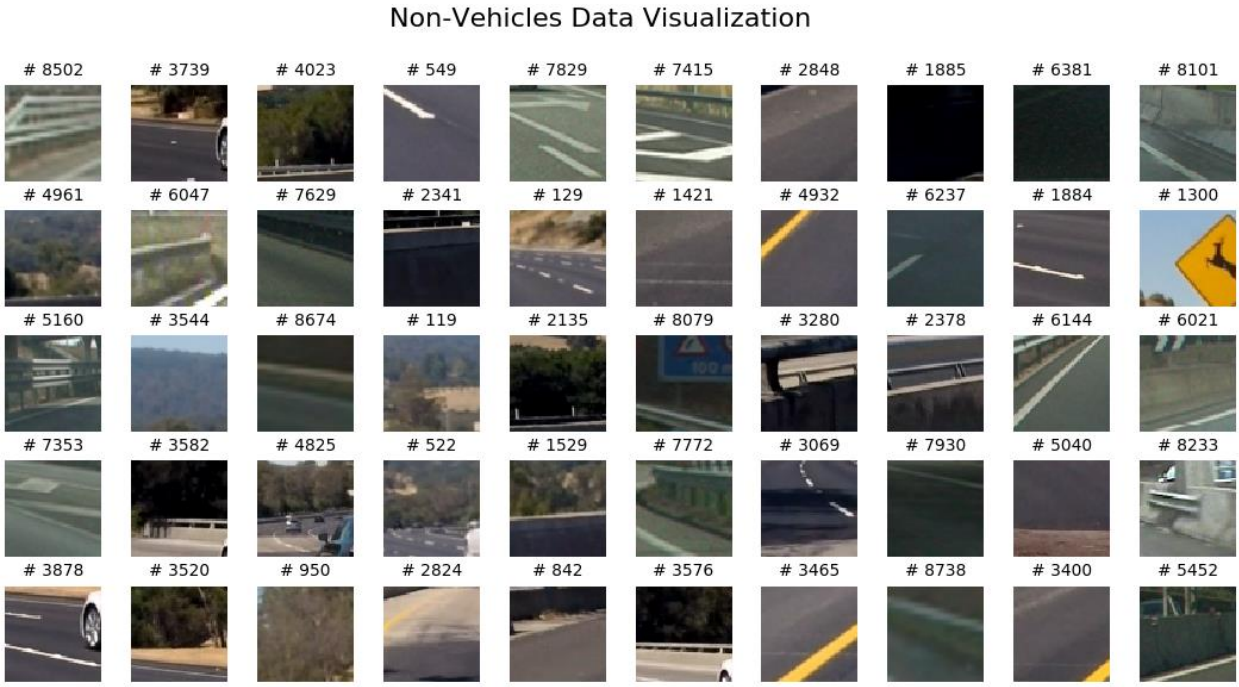


Figure 2. Visualization of 50 randomly selected non-vehicle images.

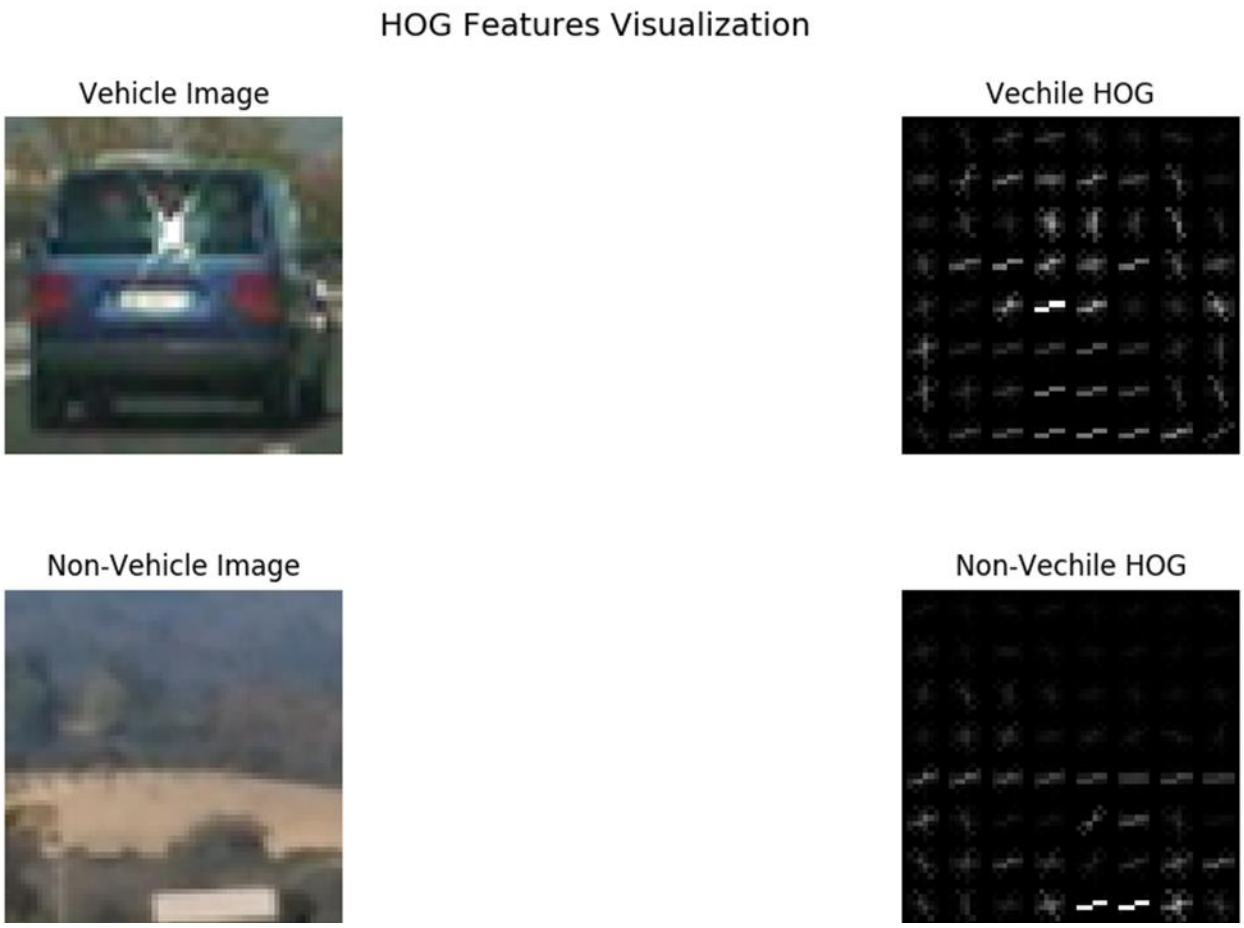


Figure 3. Visualization of HOG features for vehicles and non-vehicle images.

3. Features Extraction

The following steps describes the implemented images feature extraction functions in order of execution:

1. **Color Spatial Features**: the function “bin_spatial()” that can be found in “P5 VehicleDetect8.py” lines 205 => 209, is used to extract the contribution of different color channels in each image. Or in other words, to compute the binned color features. The channel of each image is resized to (32, 32) and then raveled.
2. **Color Histogram Features**: the function “color_hist()” that can be found in “P5 VehicleDetect8.py” lines 211 => 221, is used to compute the histogram of each color channel in each image with a designated number of pins (using the numpy “histogram” function), and then concatenate them.
3. **HOG Features**: the function “extract_hog_features()” that can be found in “P5 VehicleDetect8.py” lines 258 => 300, is used to compute the histogram oriented gradients of each image channel separately and then append them together if the option “ALL” is selected.

The previously implemented “get_hog_features()” function is used in the implementation if this function, which implicitly use the SciKit-Learn function “hog”.

4. **Combining All**: The above feature extraction functions produce the following feature vectors:

    a) Using the “bin_spatial()” function and “spatial_size=(32, 32)” results in a feature vector of 32x32x3 = 3072 elements.
    b) Using the “color_hist()” function and “hist_bins=32” results in a feature vector of 32x3 = 96 elements.
    c) Using the “extract_hog_features()” function, “orient=9”, “pix_per_cell=8”, “cell_per_block=2”, “hog_channel='ALL'” results in a feature vector of 7 x 7 x 2 x 2 x 9 = 1764 x 3 = 5292 elements.
    d) If all the above functions are used the resulting feature vector will be of the following length: 3072+96+5292 = 8460 elements

4. Training the Classifier

The following steps are used to build up and train the vehicle/non-vehicle classifier:

1. Compiling a training data set “X” of “35,520 x 8,460” size which includes 35,520 vehicle/non-vehicle feature vectors of length 8,460 each. These training set represents the input to the classifier.
2. The feature sets must be scaled; before combining them together; using the SciKit-Learn “StandardScaler().fit()” function. Figures 4 & 5 show the visualization of raw and normalized feature vectors for two vehicle images.
3. Compiling an output training set “Y” of “35,520 x 1” size which each element is of a Boolean value of 1=>vehicle or 0=>non-vehicle.
4. Shuffle the training sets randomly and split them to 80% for training and 20% for testing using the SciKit-Learn “train_test_split()” function.

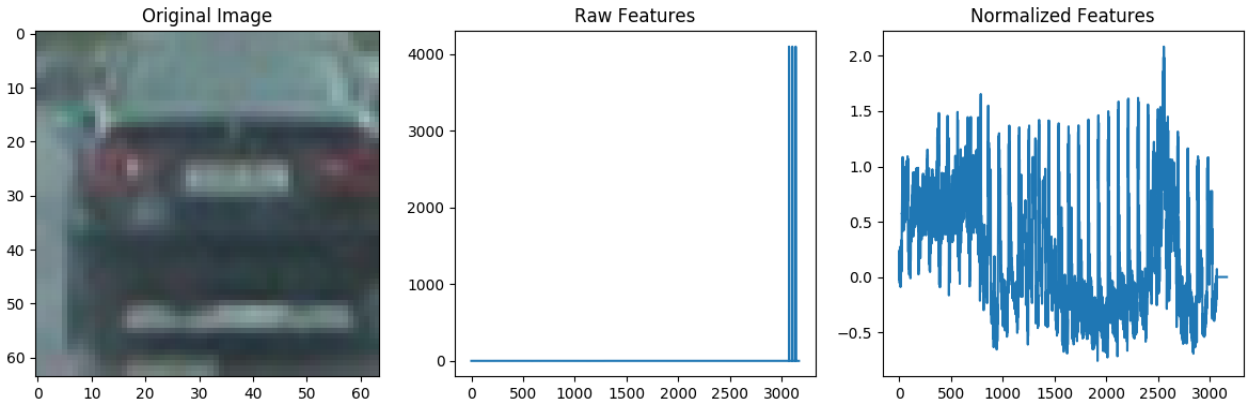


Figure 4. Visualization of feature vectors for vehicles images.

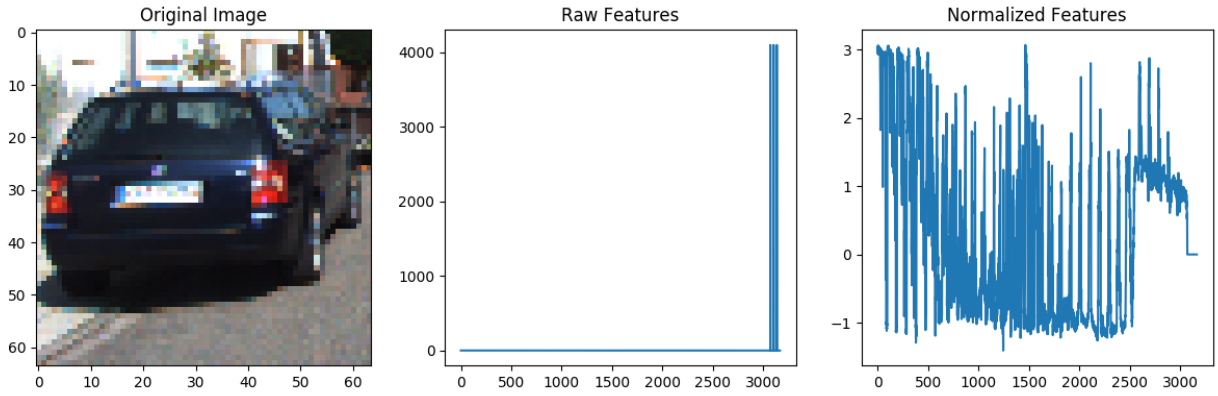


Figure 4. Visualization of feature vectors for vehicles images.

5. Using a Linear Space Vector Machine Classifier function “LinearSVC()” of the Sci-Kit Learn library, the model got trained with high accuracy (above 97.7%) in almost all the selected parameters combinations. Then the trained model is tested on the test images supplied by Udacity. The results were not good in several cases. Extreme experimentations have been done with many parameters combinations, however, the results still were not acceptable.
6. After several trials and errors, it is found that the color spatial features are taking significant portion of the feature vector length (>36%) without adding a real value (sometimes even confusing element) to the distinction of vehicles / non-vehicles features. Moreover, the color histogram features are of a very insignificant contribution (~ 1.1%) of the feature vector as well as to the distinction between vehicles / non-vehicles.
7. Therefore, both the color special and histogram features have been removed from the feature vector and keeping only the HOG features. By doing that, this results in the reduction in the length of the feature vector from 8,460 to 5292 features only. This is off course simplifies the training and the real-time application of the algorithm, and results in a huge reduction of processing and training time.
8. The new Linear SVC classifier with training data set of size = 35,520 x 5292, is constructed using several color spaces with the training results shown in Table 1.

Table 1. LinearSVC training results.

Color Space	Training Time	Prediction Time for 10 Labels	Test Accuracy
RGB	19.5 Seconds	0.01563 Seconds	0.9716
HSV	8.94 Seconds	0.001 Seconds	0.9865
HLS	8.83 Seconds	0.0015 Seconds	0.9831
LUV	8.79 Seconds	0.002 Seconds	0.9876
YCrCb	7.76 Seconds	0.002 Seconds	0.9899
YUV	8.34 Seconds	0.003 Seconds	0.9918

LAB	5.7 Seconds	0.0 Seconds	0.9916

9. Almost all the color spaces produced comparable result except the “RGB”. The “LAB” color space produces the fastest performance in both training and prediction with second to highest accuracy behind the “YUV”. However, while tesing on test-images “YUV” produced false positives more than “LAB”. Therefore, “LAB” color space is selected for the next steps.

5. Vehicle Detection and Tracking Pipeline

The following steps have been used to detect and track vehicles, and presented in order of execution. The function “process_image()”; implemented in “P5 VehicleDetect8.py” lines 594 => 758; is the main procedure that executes all the necessary routines to successfully detect vehicles:

1. **Finding lane lines:** calling the advanced lane line finding function “process_image_lane()” imported from “P4_AdvLane3.py” file.
2. **Detecting vehicles by sliding windows technique:** by calling the “find_cars()” function with the following parameters:
    a) “orient = 9” defining the number of histogram bins per cell and it is used for the HOG feature extraction for images or video frames.
    b) “pix_per_cell = 8” defining the number of HOG pixels per cell.
    c) “cell_per_block = 2” defining the number of HOG cells per block.
    d) xstart, xstop, ystart, ystop: these 4 parameters define a rectangular area on the image or frame that represents the region of interest (ROI) in which the function searches for a car in it by the sliding windows technique.
    e) “step_size = 2” defining the how many cells to step (or to slide) to construct a new search window that will overlap with the previous search window.
    f) “Scale_Step = 0.25” defining the step at which the search window sizes increments from one search scan to the next.
    g) Scale_Multiplier_Start, Scale_Multiplier_End: two parameters defining the starting and stopping of the windows sizes increment while scanning of the ROI area.

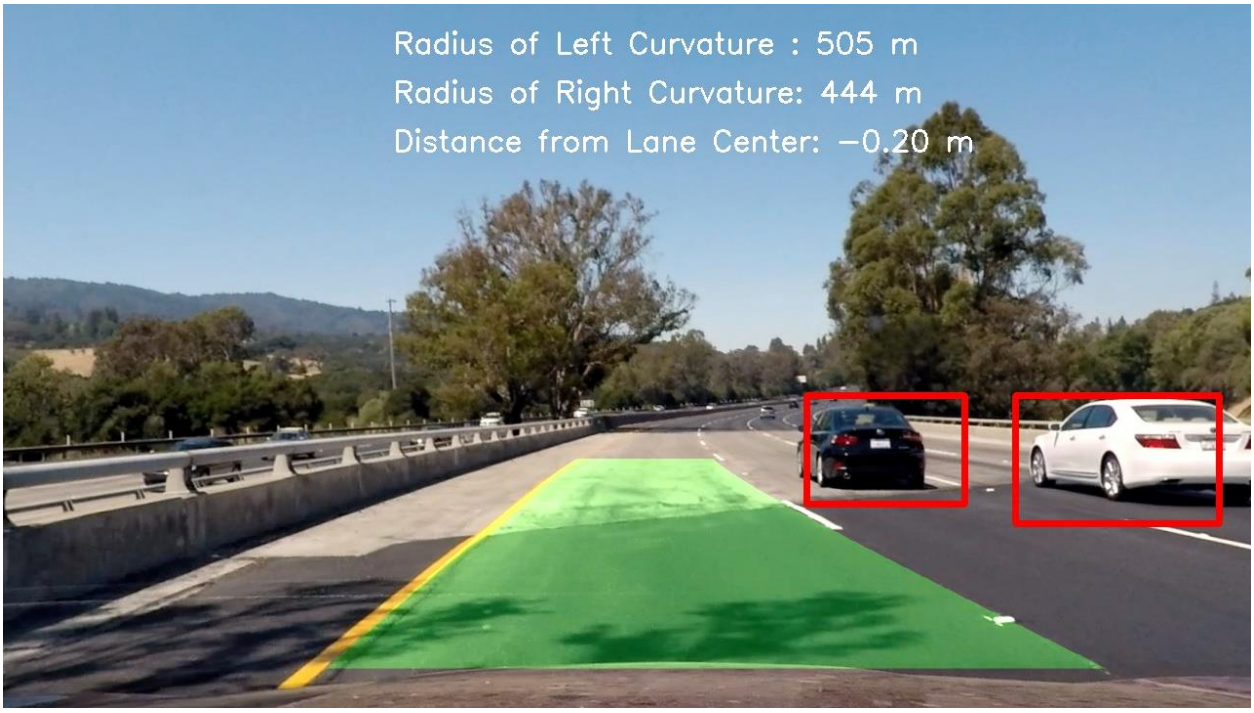
    The function uses the trained SVC classifier model and applies it on each constructed search window. Sliding windows with different sizes is being constructed to cover the defined ROI as shown by the code snippet “P5 VehicleDetect8.py” lines 617 => 637.

    The “find_cars()” function is applied several time with different set of “a => g” parameters based on the need as shown in the code snippets “P5 VehicleDetect8.py” lines 639 => 655 and “P5 VehicleDetect8.py” lines 657 => 673.
3. **Building active heatmaps:** The goal is construct a heatmap for each found car box during the search of sliding windows scan. This heatmaps as used to filter out (try to minimize) the false-positive boxes. The parameter “HEAT_THRESHOLD” is used to only pass the car boxes with multiple hits (based on its value). Functions “add_heat()” and “apply_threshold()” are used to carry out the false positive technique as shown in “P5 VehicleDetect8.py” lines 506 => 613 and “P5 VehicleDetect8.py” lines 517 => 523 respectively.

4. **<u>Labeling car boxes:</u>** the overlapped true-positive car boxes are then grouped and in bigger boxes and labeled using the “label()” function from the Sci-Kit Learn library as shown in “P5 VehicleDetect8.py” lines 728 => 729.
5. **<u>Drawing the labeled car boxes:</u>** as a final step, the labeled boxes are drawn on the original test image or video frame using the “draw_labeled_bboxes()” function found in “P5 VehicleDetect8.py” lines 524 => 588.

Figures 5 & 6 shows examples of the results after the execution of the above pipeline on the supplied test images by Udacity.

Figures 5 Results after the execution of the image processing pipeline on test_image3.



Figures 6 Results after the execution of the image processing pipeline on test_image4.

6. Solution Approach

The following points shed some light on some technical tricks and aspects that have been tried or implemented in the described pipelines:

1. **Color spaces:** around 7 different color spaces have been tried on both images and videos. From our experimentation, both HSV and LAB produced the best results in both car finding and lower false positives. The other color spaces like YUV, LUV, YCrCb, HLS produces comparable results. However, RGB produced the worst results among them by far. Therefore, both project videos based on HSV and LAB are presented.

2. **Decision function:** After applying the trained SVC model on every constructed sliding widow to search for cars, the decision function (from the SciKit-Learn library) is used instead of simple prediction function. The decision function returns the probability of the object being a car or not. So, positive probabilities means that the object is at least 50% a car, and accordingly negative probabilities mean it is more that 50% non-car object. By defining a new parameter "Confidence_score" which is identifying the confidence for an object of being a car. The higher the positive value the higher the confidence for the object of being a car. Using decision function helped reducing false positives significantly. Please refer to "P5 VehicleDetect8.py" lines 480 => 488.

3. **Heatmaps filtering:** heatmaps calculated on each frame are not used directly, however, it will be been filtered using a constructed FIR filter ("P5 VehicleDetect8.py" lines 710 => 720). The FIR uses the current and the last values of the previous four frames, before applying the "apply_threshold()" function. This helps smoothing out the found car windows and helps to reduce false positives.

4. **Car Box Vertices Filtering:** Likewise the heatmaps filtering, the found car boxes are also filtered out using FIRs ("P5 VehicleDetect8.py" lines 538 => 573). The calculated vertices are not drawn directly, but got filtered first using the calculated values of the previous three frames. This helped reducing the jitter of the position and size of the identified final car boxes for each frame.

5. **Identification of the regions of interest:** the "find_cars()" function has been modified to include the identification of the search ROI by both x and y axis. This helped to more accurately identify search areas, reduces the search time, improve search performance and eliminates unrequired false positives.

6. **Frame sampling:** throughout the experimentation, it is found that it is not necessary to search for cars every frame at the current sampling rate of the project (25 fps) video as the movement of cars from frame to frame is not that fast. Therefore, the active search for car is restricted to every other frame, which reduces the video processing time by half ("P5 VehicleDetect8.py" lines 605 => 608) and almost didn't affect the result at all.

7. **Sanity checks:** some sanity checks are used to improve the identified car boxes like:
    a) Car Box size: the identified car box size is being measured and checked out before it is being drawn to the image or video frame. This is done by measuring the diagonal of the identified box and compare it with a certain threshold as shown in "P5 VehicleDetect8.py" lines 573 => 582.
    b) Car Box Position: some check are added to check the position of the identified car boxes. For example, in the supplied images, can boxes can't be found lower than "y = 400" as in "P5 VehicleDetect8.py" lines 578 => 582.

7. Shortcoming of the implemented approach

The following list summaries the identified shortcomings:

1. False positives: the false positives are still a problem for the current implementation. There are several causes for this problem as follows:
    a) Not enough training data with false positive images: the training data is lacking many cases that identify false positives like images of pure pavements with saturated colors, pure green areas like trees … etc. These kind of images need to augment the data with big number.

b) Not enough training data with cars with some colors: as an example the training data is lacking cars with white colors at different orientations.
c) Not enough training data with partial cars: most of the training data includes full cars, however the data needs to be augmented with images of partial cars (e.g. the front part only …etc.).

2. Color spatial and histograms of colors: actually, it is noticed that the color special features and the histogram of colors are not adding enough or even any value the identification of cars in images. Actually, it is noticed that even contributing to the false positives. Actually, this is logical as cars can come in any color, nothing actually unique in this regard. Moreover, the saturation of colors in cars can be easily misidentified with trees and road portions which has high saturation of colors.

3. Parameter tuning: there many parameters to tune, you need a lot of trials and errors. There is no consistent way to make sure that you have selected the right value for a certain parameter.

8. Suggested Improvements

The following list summaries the suggested improvements:

1. Use higher order of the FIR to smooth-out the transition heatmaps and car boxes. The current used on is of order 3, I believe we can try-out even to order 10.

2. Need to investigate more into the color space issue. Eventhough, we have experimented a lot with 7 different color space, however, we didn't do experimentation with mixing channels from different color spaces (like combining the S channel from HLS and HSV or combining the L channel from LUV and LAB … etc.)

3. Using color spatial and color histogram features in a different way and not combining them in the same feature vector with HOG feature as being done. For example, I am propose using them as sanity checks.

4. Adding more sanity checks to the pipeline like predicting the label car box positions in next video frames correcting the identified ones accordingly.