Project 5. Vehicle Detection. Writeup.

Goal of the project: Build a pipeline for detecting the vehicles in the image and use it for vehicles detection in the video file/stream.

The steps of this project are the following:

- Generate a training data set by performing a feature extraction from a labeled data set of images using following technics:
  - Histogram of Oriented Gradients (HOG)
  - Histogram of colors
  - Spatial binning of colors
- Select a proper classifier for given use case and train it on extracted feature set
- Implement a sliding-window search technique and use trained classifier to search for the vehicles in image.
- Estimate the bounding boxes for detected vehicles.
- Implement filtering of false positive detections.
- Run implemented vehicle detection pipeline on provided test video file **project_video.mp4**

Project resources

My project includes the following required files:

- **pipeline.ipynb** - Jupiter notebook file containing the implementation of pipeline for detecting vehicles in a video file
- **output images/** - folder containing the images that visualize the results of each step of detection pipeline execution. Description of each image (as well as a copy of image itself) is provided in a writeup report.
- **writeup.pdf** - summarizing report of the work done in scope of this project (this document)
- **project_video_out.mp4** - video recording of applying of implemented vehicle detection pipeline to provided **project_video.mp4** test video file.

All above mentioned project recourses are can be found here.

All visualizations used in this write-up document are implemented with means of matplotlib.pyplot library.

1. Training data exploration

For this project the labeled dataset is provided within two sets of images (vehicles and nonvehicle) and has the following structure:

Size of data set: 17760 images

- Images with car: 8792
- Images without cars: 8968
- Shape of each image is 64x64x3 (64 pixels height/width x 3 (rgb) channels)

Here is an exploratory visualization of the loaded data set (block **2. Load the dataset** in pipleline.ipynb):

- random selections of the images representing 'car class:

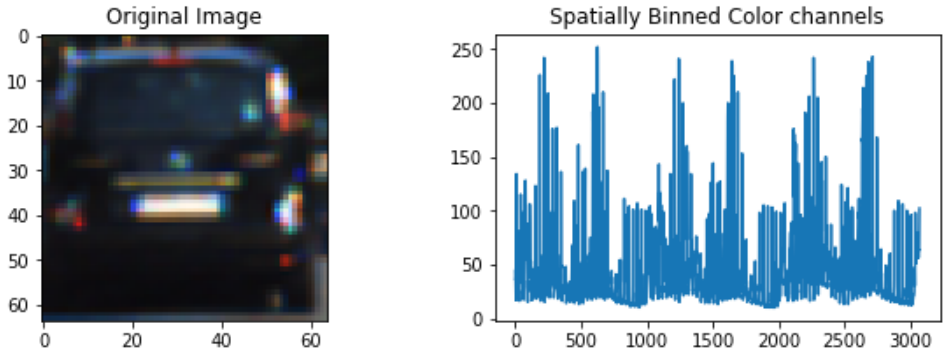
- random selections of the images representing ‘not a car class:

2. Feature extraction

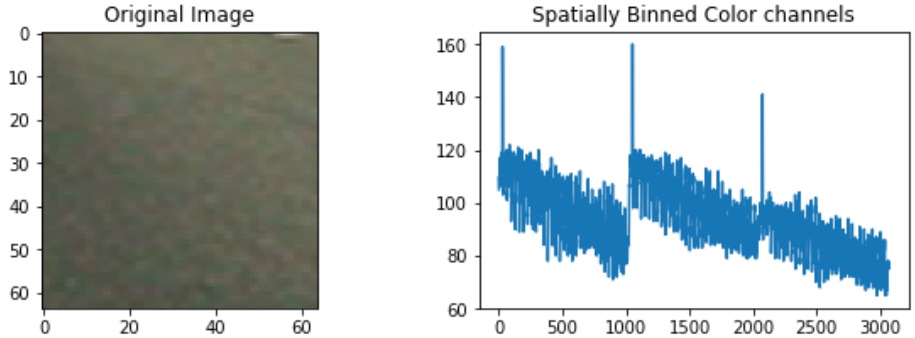
For extracting a feature for training from provided data set I will use following technics:

- **Spatial Binning of Color**. This method image (block **3. Generating feature vector from images using Spatial Binning of color** in pipleline.ipynb) allows to get the vector of features using following processing sequence applied to the:
  - image is resized to smaller resolution
  - resulting image is splatted into separate color channels
  - pixel value matrix of each color channel is flattened into 1D array
  - flattened arrays are then concatenated into single 1D feature vector

Following is an example of extracting the feature vector from the random ‘car’ image of dataset using a Spatial Binning of Color technic :



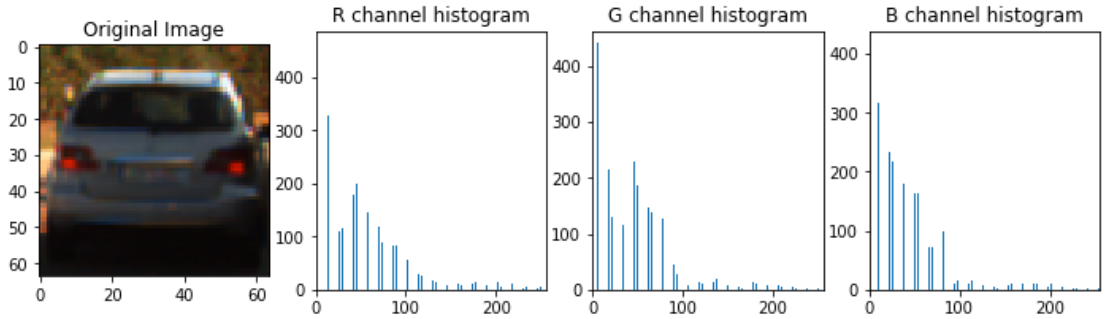
And same type of visualization, but for the random image from ‘not a car’ image dataset:



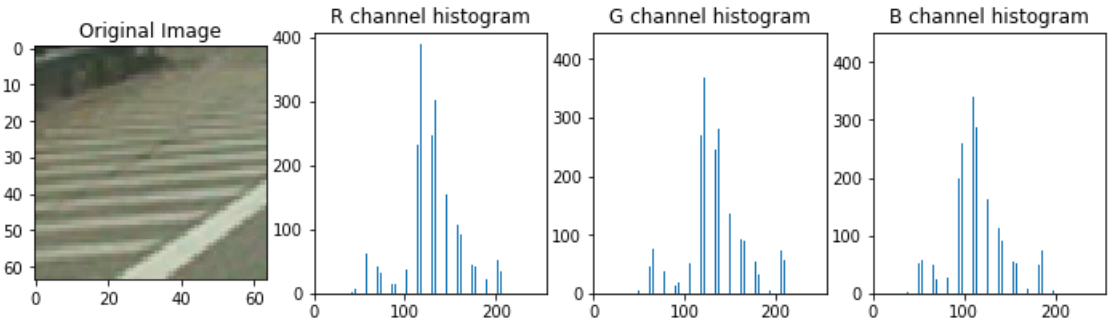
As it can be seen from the visualizations there is a clear difference in a pixel value ranges of feature vectors extracted from ‘car’ and ‘not car’ images. Using this specific characteristic by including corresponding feature vectors into training dataset will allow to improve the accuracy of the classifier during detection of the cars displayed on images.

- **Histogram of colors.** This method (block **4. Generate feature vector from images using Histogram of Color channels** in pipleline.ipynb) allows to generate the feature vector for training the classifier using following processing sequence applied to the image:
    - image is splatted into separate color channels
    - pixel value range of each color channel is split into N bins
    - pixels are assigned to corresponding bins based on their values**.** Resulting histogram forms a 1-D array with size N were each element represents a corresponding histogram bin and element value represent the number of image pixels in this bin
    - corresponding color channel histogram 1D arrays are concerted into a feature vector of size n_channels X n_bins

Following is an example of extracting the feature vector from the random 'car' image of dataset using a Histogram of Colors technic (number of histogram bins: 64):



And same type of visualization, but for the random image from 'not a car' image dataset (number of histogram bins: 64):

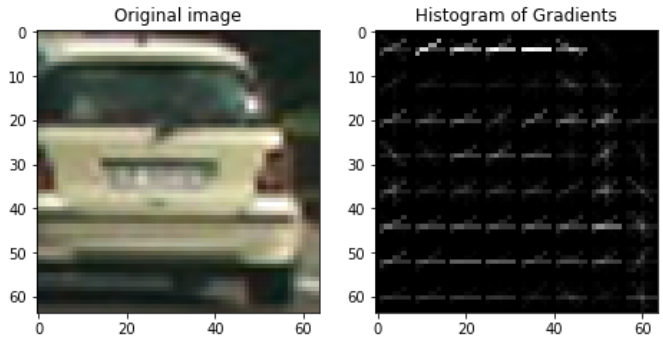


The above visualizations of generated feature vectors show a clear difference in a pixel values distribution across the images of 'car' and 'not car'. Thus using this characteristic by extending the training dataset with corresponding feature vectors should additionally increase the accuracy of the classifier during detection of the cars displayed on images.

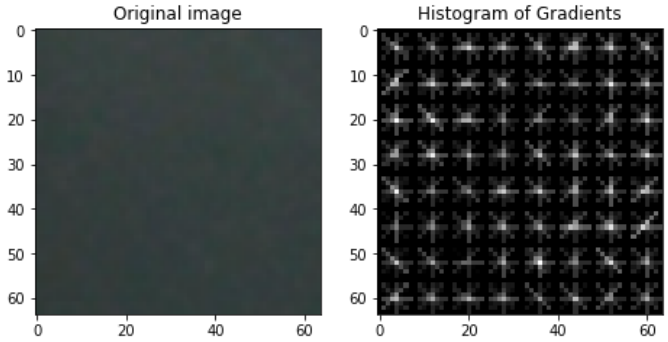
- **Histogram of Gradients (HOG).** This is the main feature vector generating technic of this project. This method (block **5. Generate feature vector from images using Histogram of Gradients** in pipleline.ipynb) allows to generate the feature vector for training the classifier using following processing sequence of the image (search window): [in accordance to scikit reference]
    - normalize the image to reduce the influence of illumination effects (not used in my implementation)
    - compute the first order image gradients by x and y orientations. They capture the contour, silhouette and some texture information, while providing the resistance to illumination variations.
    - generate the gradient histograms. For this divide the image into 'cells' (using *pixels_per_cell* input parameter). For each cell accumulate local 1D histogram of gradient (edge orientations over all the pixels in the cell). This combined cell-level 1D histogram forms the "orientation histogram". Each such histogram divides the gradient angle range into a defined number of bins (using *number_of_orientations* input parameter). The gradient magnitudes of the pixels in the cell are used to 'vote' into the orientation histogram.

- normalize cells across the image blocks (the local groups of cells calculated using *cells_per_block* input parameter).Normalization introduces better invariance to illumination, shadowing, and edge contrast, and is performed by accumulating a measure of local histogram 'energy' over blocks. The result is used to normalize each cell in the block. Usually each individual cell is shared between several blocks, but its normalizations are block dependent and thus different. The cell thus appears several times in the final feature vector, but with different normalizations.
- concatenate flattened gradient histograms of blocks into resulting feature vector

Following is an example of extracting the feature vector from the random 'car' image of dataset using a Histogram of Gradients method (orientation bins: 8, pixels per cell: 8x8, cells per block: 5x5, color channel: grayscale):



And same type of visualization, but for the random image from 'not a car' image dataset:



As it can be seen from the above visualizations the histogram of gradients of 'car' image produces the recognizable contour and silhouette of the car, thus corresponding feature vector should help the classifier to more accurately detect the corresponding the car on the image. On other hand the histogram of gradients of 'not a car' image represents the smooth uniform texture, thus it should also not be hard for classifier by using corresponding feature vector to 'decide' that corresponding image does not contain any car-like object. For generating the feature vectors of training dataset I will use following parameters for Histogram Of Gradient calculation function: color channel: all 3 channels, number of orientation bins: 12, number of pixels per cell: 8x8, number of cells peers block: 6x6. These values of parameters were calculated empirically by testing various combinations of HOG method parameter values on a small subset of original dataset (also provided for this project with vehicles and non_vehiles image sets) and comparing the accuracy of prediction of the classifier trained using corresponding generated feature vectors. Following are top 10 results of this 'brute force' parameter tuning (full set of my HOG parameter tuning results can be found here):

classifier accuracy score	extraction time, sec	training time, sec	prediction time, sec	HOG parameters
1	5.579999924	0.340000004	0.001	('YUV', 9, 8, 6, 'ALL')
1	5.789999962	0.389999986	0.001	('YCrCb', 12, 8, 6, 'ALL')
1	5.860000134	0.330000013	0.001	('YUV', 8, 8, 5, 'ALL')
1	6.039999962	0.479999989	0.001	('YCrCb', 8, 7, 5, 'ALL')
1	6.519999981	0.439999998	0.001	('YCrCb', 8, 7, 4, 'ALL')
1	6.670000076	0.25	0.001	('YUV', 8, 8, 3, 'ALL')
1	6.690000057	0.74000001	0.001	('YCrCb', 8, 6, 5, 'ALL')
1	6.760000229	0.319999993	0.001	('YCrCb', 10, 8, 3, 'ALL')
1	6.809999943	1.210000038	0.001	('LUV', 11, 6, 6, 'ALL')

3. Training data preparation

To create the training dataset from provided image dataset for training the classifier I generated the set of feature vectors for each image from the dataset using all above described methods and concatenated the generated method specific feature vectors into resulting image feature vectors (block **6. Generate a training and test feature and label sets from loaded dataset** in pipleline.ipynb).
Before feeding each image into a feature vector generation functions I converted the image color space from RGB to YCrCb, since using this color space in combination with above mentioned values of HOG calculation function parameters has shown one of the best results during HOG calculation parameters tuning experiment and also worked very well for detecting the cars on the bunch of test images (see **output_images/** folder).
Applying the feature extraction methods to the provided image dataset allowed me to generate the set of feature vectors with following characteristics:

- `spatial binning features vector shape: (3072,1)`
- `color histogram features vector size: (96,1)`
- `HOG features vector size: (7056,1)`

`Combined feature vector size: (10224,1)`

To minimize the bias of the classifier to the features from one feature vector over the other I had to normalize the features in a combined feature vector to a zero mean and unit variance. For this I have used the StandardScaler class from sclearn.preprocessing package.

The final shape of the feature set generated from provided image dataset: `(17760, 10224)`
I have shuffled it and split into a training and test features sets in a ratio 80%-20%:

- `Training feature set size: (14208, 10224)`
- `Test feature size: (3552, 10224)`

4. Training the classifier

For my implementation I have decided to try a Linear SVC, which is one of the types of Support Vector Machine classifier. It was suggested in a lesson as the one providing a good tradeoff between the speed of training/prediction and accuracy of predictions for this specific use case (block **7. Train the SVC classifier using generated training set** in pipleline.ipynb).
Trained classifier gave me a `0.9941` prediction accuracy score on a test dataset, which is pretty good result in my opinion. Thus I decided to stick with this one and look into other possible options like Decision Trees or KNN or maybe even some ensemble of classifiers like AdaBoost only if the final vehicle detection results would not satisfy me.

5. Sliding Window Search

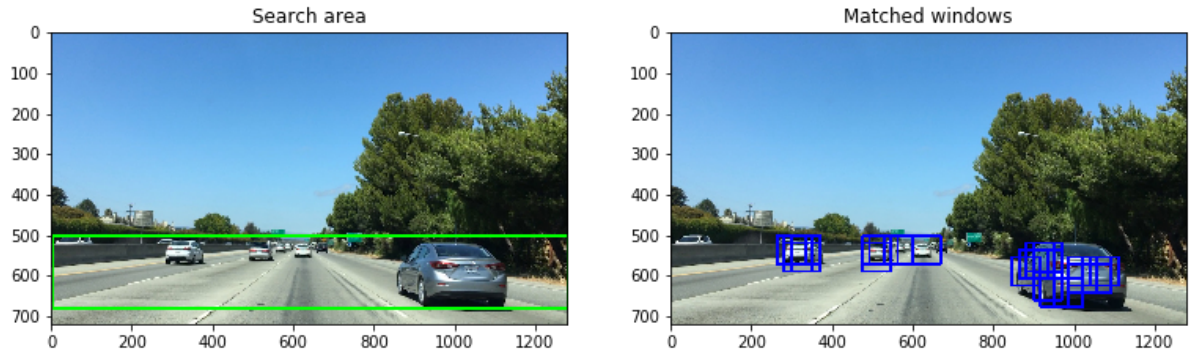
My version of sliding windows search algorithm consists of following steps (block **8. Sliding windows search algorithm** in pipleline.ipynb):

- define a size of search area on the image (only vertical a top and bottom boundaries, since full width of image used for all search areas).
- convert the search area to required color space (YCrBr in my case)
- rescale the search area using preconfigured scaling factor. This together with defining various specific sizes of search area allowed to look for the vehicles of different size (mean distance from ego vehicle) in a different expected locations in relation to ego vehicle (for my implementation I have empirically selected 11 different combinations of search area size + scaling factor, that allowed to precisely detect all displayed vehicles on the various test images captured as screenshots from the target video file project_video.mp4 (see **output_images/** folder))
- fetch the HOG features for each of the color channels of full search area image. This way I will omit computing the hog features for each of the sliding windows, but will extract them as a corresponding subset of full feature set, extracted once for dull search area image.
- Based on 'pixels_per_cell', 'cells_per_block' configuration parameters calculate the number of HOG blocks in the search area, number of extracted features per block and additionally (using another parameter windows_size) the number of HOG blocks per sliding window. Also using 'cells_per_step' configuration parameter calculate the amount of sliding windows in a search area image (number of steps in horizontal and vertical directions).
- Using all above calculated parameters cyclically fetch the subset of features from previously extracted full HOG feature vector that matches the corresponding sliding window size and location.

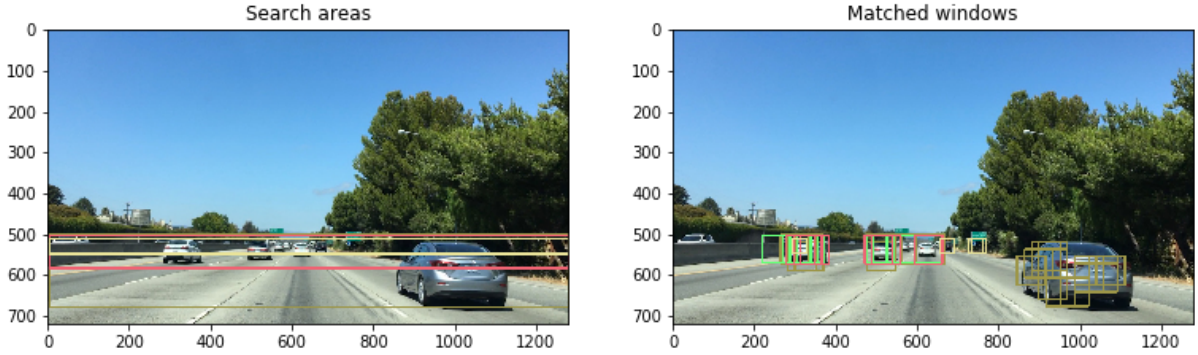
- Use this subset of HOG features in combinations with spatial binning and color histogram features extracted for corresponding part of search area image (matching the current sliding window) to make a prediction about the presence of the car on the current sliding window image. For doing the predictions, previously trained classifier (Linear SVC) and fitted feature set normalizer (StandardScaler) must be used.
- If classifier detected the car on the current sliding window – add corresponding window perimeter rectangle to resulting list of bounding boxes

Following is the visualization of running the above described detection algorithms against the test image (of size 720x1280) with following search are configuration parameters:

- vertical upper limit (in pixels): 502
- vertical lower limit: 680
- scale: 1.1

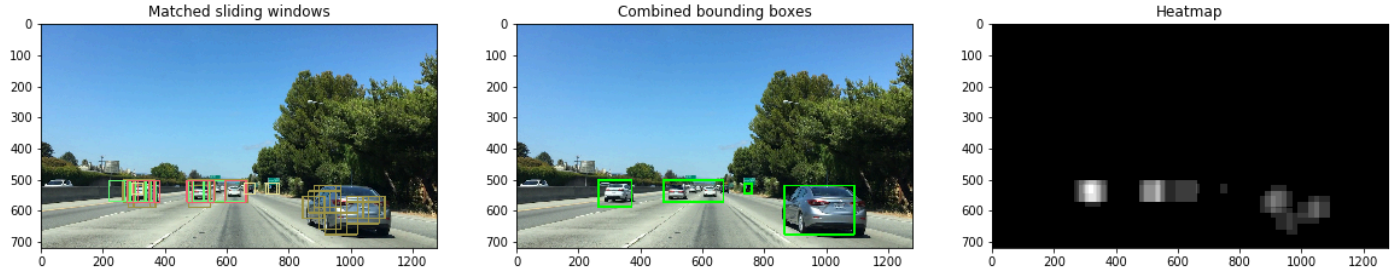


As it can be seen from the above detection result, not all of the cars were detected, e.g. small cars (located relatively far away from ego car) were ignored by classifier. To handle this issue I run my detection algorithm with 4 (in case of this specific test image) different combinations of search areas size and scaling factor (see block **9. Using for multiple search space configurations for improving detection accuracy** in pipleline.ipynb), what produced the following detections (results of each search configuration detections as well as corresponding search areas are overlaid on the original test image with a different color):



6. Estimation of bounding boxes for detected vehicles

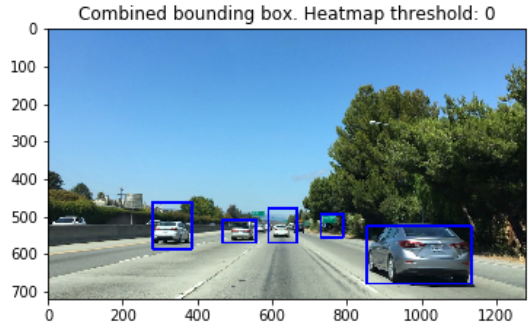
Now to group all the matched windows that cover each specific detected vehicle into a corresponding bounding boxes (one per detected car) and by this also compensate the inconsistency of detection surface for some detected (usually big or relatively close) vehicles, I used the technic of calculating the heat map for each vehicle based on the number of overlapping matched windows. (block **10. Group matching detection windows using heat map with filtering** in pipleline.ipynb). Applying this method gave me the following result:



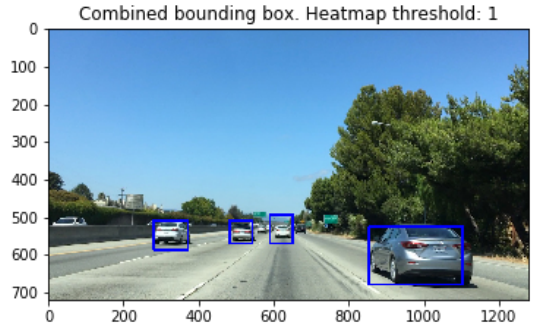
7. Filtering of false positive detections

As it can be clearly observed in above visualization, in some detection cases appears the side effect of overlapping the matched detection windows for adjacent closely displayed cars or miss detecting one or several phantom (non existing) cars (false positive detections). To cope with such side effects I have introduced following filtering mechanisms:

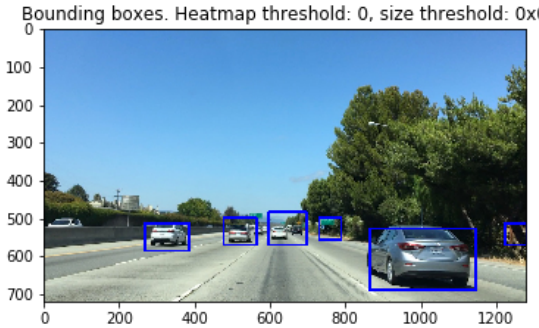
- thresholding the heat map with configured minimal required amount of matched windows for each specific detection (selected empirically for each specific detection case scenario (see **apply_threshold_to_heatmap** function in block **10. Group matching detection windows using heatmap with filtering** in pipleline.ipynb). Here are the examples of running the single search space configuration detection algorithms with threshold value equal 0:



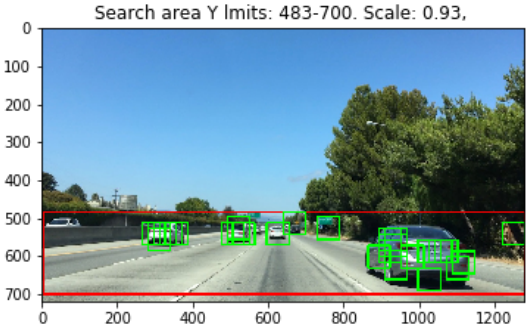
and equal 1 accordingly:

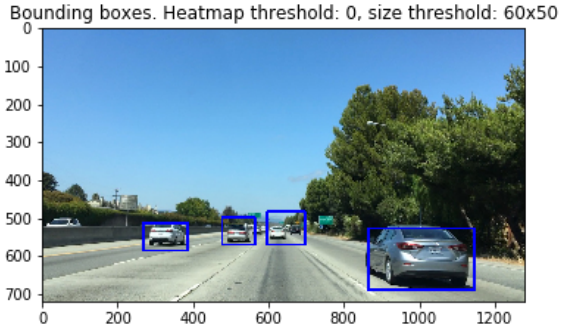


- Thresholding the resulting bounding boxes by setting the minimal width and height of bounding box, in order to filter out ‘strong’ false positives, when these are miss detected with multiple overlaid matched windows. But this type of filtering can work only until certain limit, since the configured threshold values should not exceed the minimally expected size of bounding box for real detected vehicle. Thus such filter is usually suitable for dealing with relatively small ‘strong’ outliers, and its values should also be empirically defined in each specific use case. Here is the example of vehicle detection result for the image with presence of small size ‘strong’ outlier:



and the same detection, but filtered by minimal size (60pxx50px) for detection bounding box:



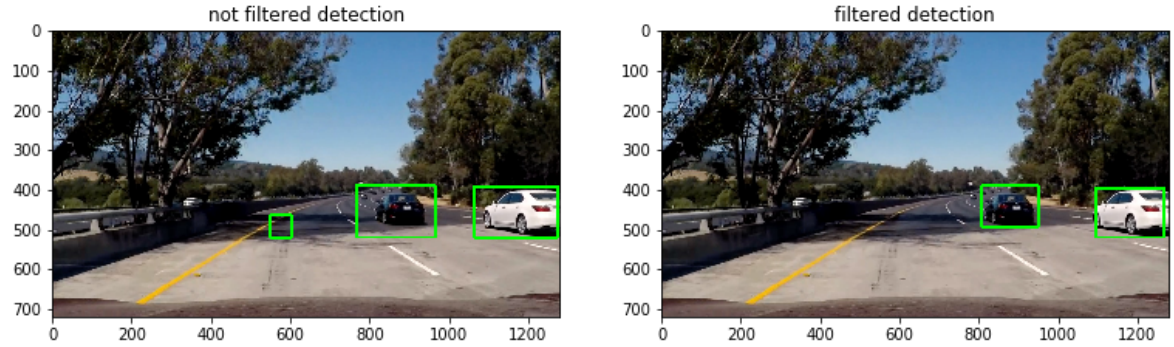


8. Applying the detection pipeline to target video file

To successfully detect the vehicles in provided for this project test video file **project_video.mp4** I first had to define all sort of configuration parameters for my detection pipeline which would best fit to this exact type of road environment. For that I did the initial run of the pipeline with dummy parameters for search space and 'outlier' filtering values, and identified several spots where the pipeline had issues with correct detections. So I used the snapshots of these 'challenging' cases for fine-tuning the correct combination of search space and filtering parameters. Following are the fine-tuned values of configuration parameters for execution of my pipeline:

- Search space configurations: [(395,488,1.32), (431,518,1.35), (405, 499, 1.20), (405, 499, 1.06), (401, 499, 0.93), (392,522,2.01), (393,510,1.83), (415,499,1.0), (389,511,1.34), (401,489,1.33), (392,608,0.9)], where first value is y_min, seconds value is y_max and third value is search window scaling factor.
- Heat map threshold: 2
- Bounding box min size: 55x55 px

And an example of execution of pipeline without and with filtering for 'challenging' case from test video file:



Additionally, in order to make the detection of the vehicles in a video stream more smooth (in terms of keeping the same size of bounding box for detected vehicle across multiple frames) and robust (in terms of persistence of detected vehicle bounding box across multiple frames and additional minimization of 'phantom' 'short duration' detections) I decided to introduce the detection smoothing mechanism for the use case of execution of pipeline for video stream (see block **13. Detecting cars in the video stream with multiframe detection smoothing** in pipleline.ipynb).
Main idea of it is to cache the last N detected bounding boxes for each car and build a combined heat map from these. This technic allows to filter out inconsistent (lasting across one or very few sequential frame) 'phantom' detections by multiplying the initially configured (single frame heat map threshold by number of cached frames (a size of cache). Fine-tuning of this parameter allowed me almost completely remove all sort of false positive detections for the use case of running the pipeline against video stream. The final result of applying implemented pipeline to provided video file **project_video.mp4** can be seen in the captured output video file project_video_out.mp4.

9. Conclusions

Fine-tuning of parameters for HOG features extraction and search space size and scaling factor together with threshold for filtering of false positive detections allowed me to achieve very good results in detecting the vehicles in target video file. Pipeline was able to filter out almost all outliers and correctly detect all of vehicles moving in the same with ego vehicle direction on adjacent lanes. The detection bounding boxes were more-less smoothly changing in size over whole video without any interruptions.
The pipeline was not able to filter out only one false positive detection of very short duration (several frames) and has completely ignore the vehicle which were moving in opposite to ego vehicle direction behind over the rail guard, though detecting of these vehicles was not among the requirements and anyway would bring more noise to the visualization than any positive value.
Taking into account the accuracy score of the classifier 99,41%, as one of further improvements of the pipeline implementation I would mention the improving even more of the classification precision for given dataset (e.g. trying out other than Linear SVC classifiers and splitting the total generated feature set to training and test sets more accurately instead f dummy shuffle and 80-20 split, in order to prevent having the same or almost the same images in both training and test sets (a side effect of generating the image dataset from the video with a high framerate, when each second of video produces a big amount of almost identical images into resulting dataset)).
As one of the main limitations of implemented model I would point out the need to fine-tune multiple runtime parameters (e.g. search area size, sliding window scaling factor, filtering thresholds) depending on the road environment represented in video stream. I suspect that used values of parameters were a best fit only for this specific video file. Other video streams the pipeline may handle not so effectively.