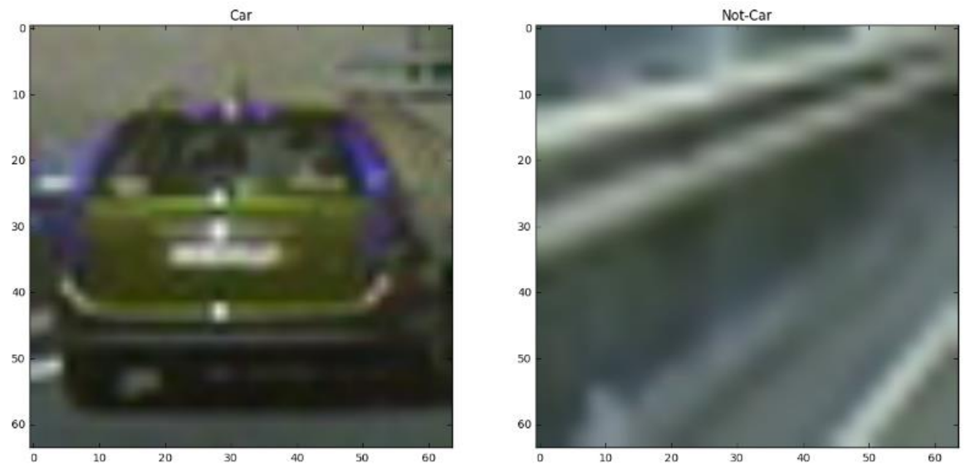
Project 5: Vehicle Detection and Tracking

The objective of this project is to write a software pipeline to identify vehicles in a video from a front-facing camera on a car. The test images and project video were provided in the project repository.

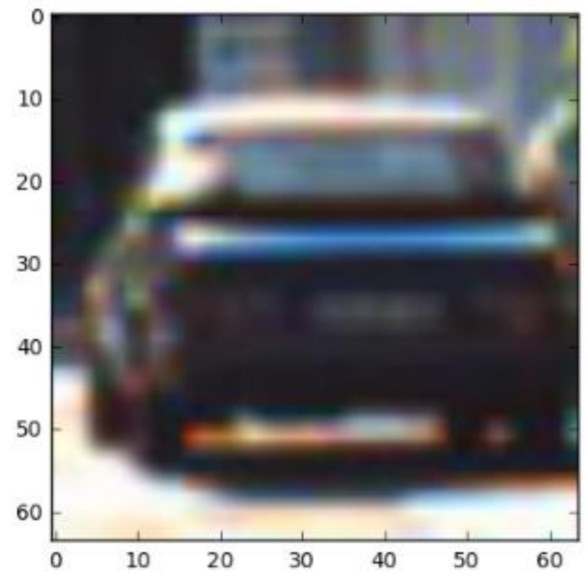
The steps of this project are the following

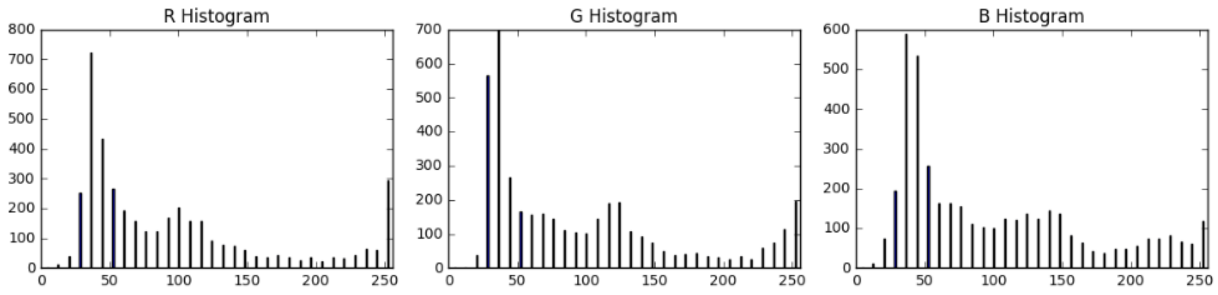
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run the pipeline on a video stream 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.

We started the project by reading the files from the dataset. The images of cars were read into the car variable while the other objects were read into the no car variable. Below is an example of a plotted car and a non-car image.



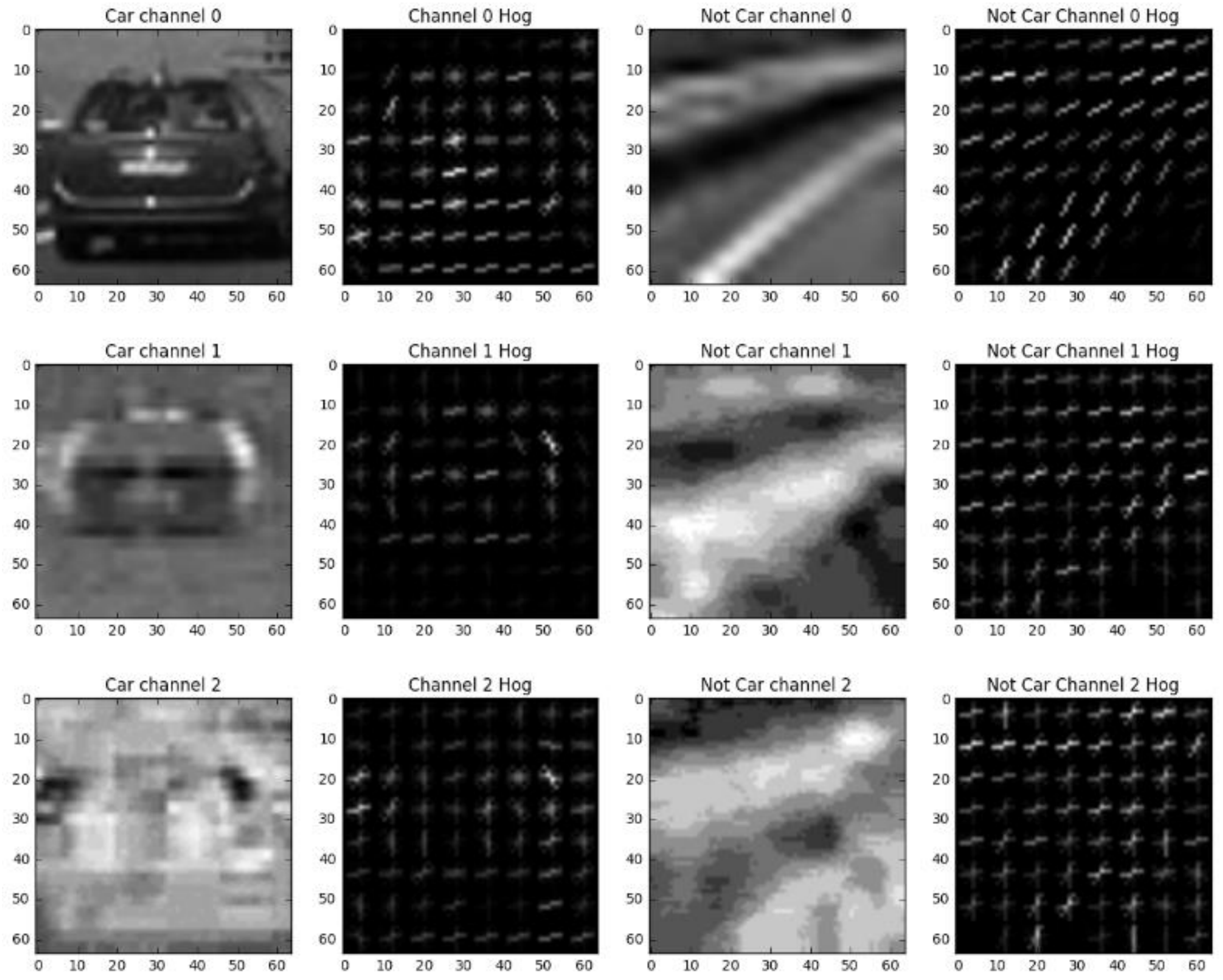
Then I computed the histogram of color values of the car image which can help to identify an area of interest by showing the locations with similar color distributions indicated by close matching.





Histogram of Oriented Gradients (HOG)

The HOG is used to detect objects in images by counting the occurrences of gradient orientation in localized portions of an image. The Scikit-image hog() function is used to compute the HOG features of our images. It takes in a single color channel or grayscaled image as input. Parameters (orientations: 9, pixels_per_cell: 2x2 and cells_per_block: 8x8) for our 64 x 64 images. Below is a plot of histogram of three color channels.



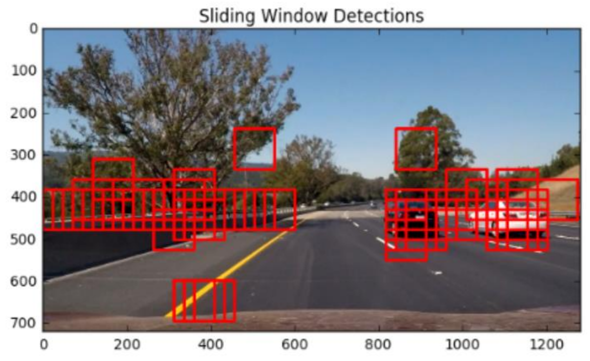
I adopted a combination of color transform features and HOG features. Both approaches represent different quantities, one type of features may dominate the other in the training dataset which can cause a bias so we applied normalization of the color transform and HOG features vectors. The sklearn StandardScaler() function is used to normalize the feature vectors. The get_hog_features() and color_hist() were used generate color features and HOG-spatial features respectively. The extract_features() function combines the both color and spatial transformation by using the get_hog_features () and color_hist() to generate new feature vectors.

Classification of the images

The extracted features were used in the training and testing of the image classifier. The Training samples was (14208, 8460) and Test samples was (3552, 8460). The Support Vector Machine(SVM) algorithm is used to train the classifier, using the sklearn.svm LinearSVC algorithm function with train and test accuracy of 99%

Sliding window technique

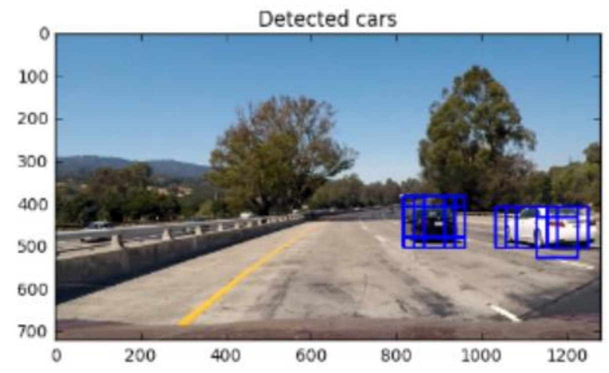
The Sliding window technique is a method for searching for objects in an image. The technique extracts a sub-region of the image, steps across the image in a grid pattern and extract the features that the classifier is trained with from each window. We implemented the sliding window technique using a window size of (64, 64) and an xy overlap of (0.5, 0.5)

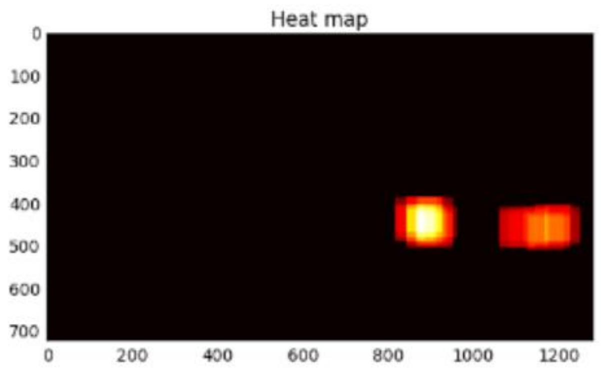


Hog Sub-Sampling Window Search

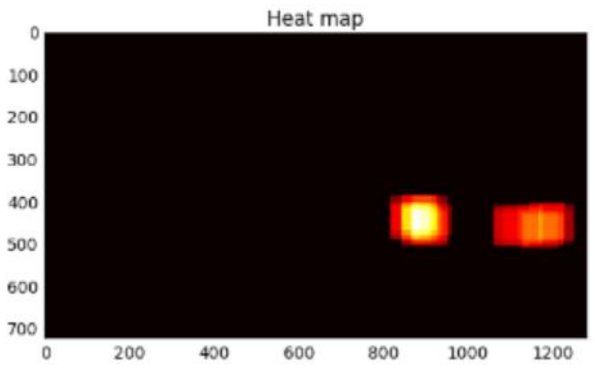
Hog Sub-sampling is a more efficient way for implementing the sliding window technique. The hog features are extracted once and then can be sub-sampled to get all of its overlaying windows. Each window of the image is determined by a scaling factor and the overlap of each window is defined by the cell distance.

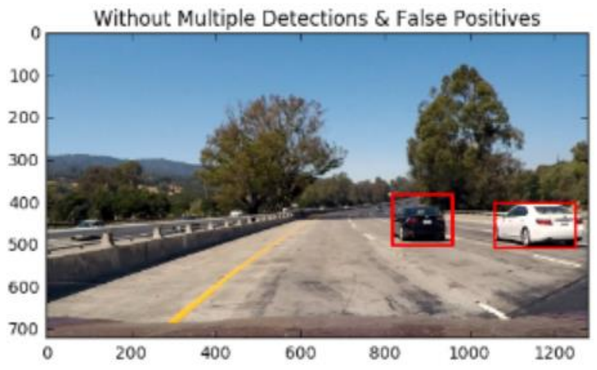
As shown in the image above, there are multiple detections on both cars and some other false positives. We therefore combine the multiple detections and reject the false positives by building a heatmap which adds a "heat" (+=1) for all pixels within windows where a positive detection is detected by the classifier and threshold (threshold was set to <= 2) the heatmap to remove false positives which are detections that do not appear from frame to frame.



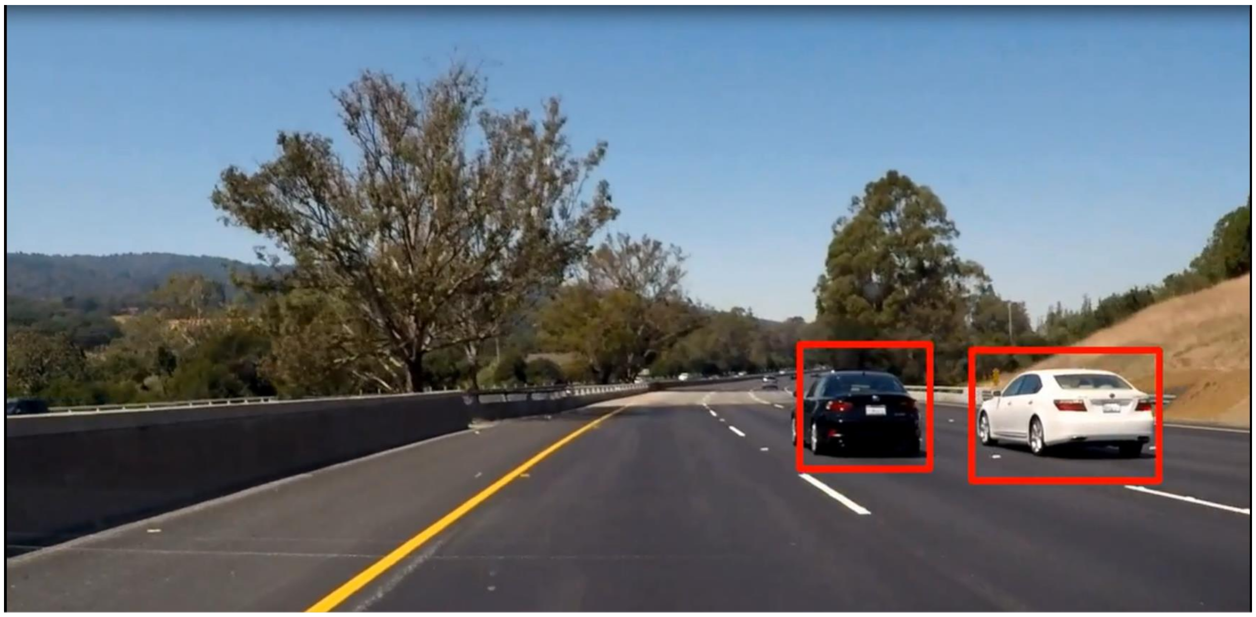


From the image plot above, we can observe the areas with multiple detection in the Heatmap in which areas of multiple detections are "hot", while transient false positives are "cool". We then use the label() function from scipy.ndimage.measurements for the multiple detections and draw bounding boxes around the labeled regions. The output can be seen in the plotted image below





The sliding-window search combined with the classifier is then used to search for and identify vehicles in the sample video. The output video is then generated with detected vehicle positions identified by bounding boxes drawn around them on each frame of video.



Discussion

The pipeline was fairly effective at detecting the vehicles in the provided video. The pipeline detected vehicles that were close to the camera but does not detect vehicles that are further away because of the size of the smaller vehicles and the specified region of interest which excluded objects 400px from the top of video. Speed was not put into consideration in determining what to detect so even though a distance car might not be a concern at lower speed, at higher speed they will be. The algorithm can be improved to be more robust to adjust its detections according to the speed of the vehicle.

After varying HOG parameters and tuning them 8 pixels per cell in each axis, and 2 cells per block in each axis, along with using 9 orientations was observed to have performed the best. We used SVM algorithm to train the extraction classifier because of its fast evaluation but a deep learning approach is worth trying to compare performance. SVMs are known to be great for relatively small data sets with fewer outliers but drop in performance with larger datasets which can be handled relatively better by a deep learning approach.