Vehicle Detection Project

The goals / steps of this project are the following:

- Performing a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, applying a color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- Implementing a sliding-window technique and use a trained classifier to search for vehicles in images.
- Running 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.

This submission comprises the following files:

- *Writeup.pdf* - The present file
- *README.txt* - A file summarizing the goals/steps and listing the content of this submission.
- *classifier.py* - python code that creates and train a classifier (SVM) for vehicle detection.
- *find_cars.py* - python code containing the pipeline for vehicle detection.
- *functions.py* - python code containing the functions needed for the pipeline.
- *model.pkl* - file containing the trained classifier plus other parameters necessary for feature extraction.
- *project_video_output.mp4* - video showing the bounding box for the detected cars in the project_video.mp4.
- *teset_video_output.mp4* - video showing the bounding box for the detected cars in the test_progect.mp4.
- *output_images* - folder containing images examples from the vehicle detection pipeline.

Histogram of Oriented Gradients (HOG)

I started by reading in all the “vehicle” and “non-vehicle” images (file “classifier.py”, lines 22-29). Here is an example of one of each of the “vehicle” and “non-vehicle” classes:

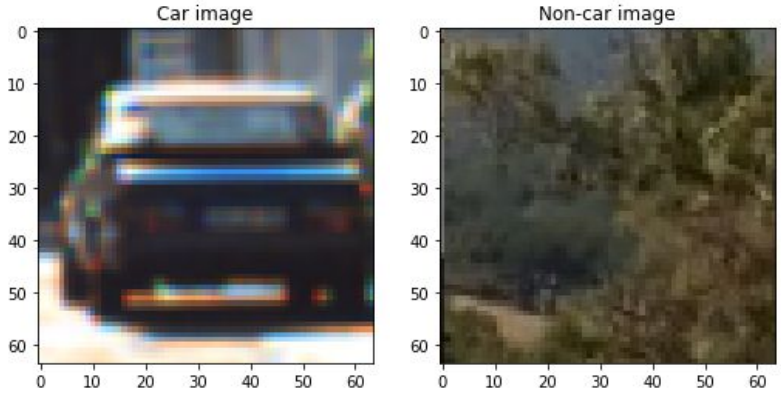

Figure 1 - Example of vehicle (left) and non-vehicle (right) images.

I then explored different color spaces and different parameters for the hog features extraction, such as *orientations*, *pixels_per_cell*, and *cells_per_block*. I grabbed random images from each of the two classes and displayed them to get a feel for what the ‘`skimage.hog()`’ output looks like.

Fig.2 shows a graphical representation of the HOG features extracted from the three channels of a car image previously converted into the YCrCb color-space. The HOG parameters here used are *orientations=9*, *pixels_per_cell=(8, 8)* and *cells_per_block=(2, 2)*:

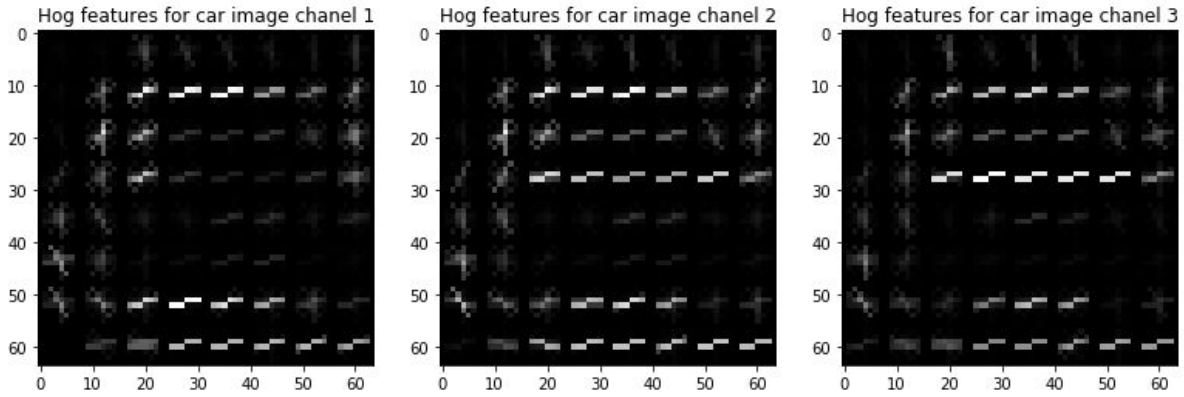

Figure 2 - Graphical representation of the HOG features extracted from images converted into the YCrCb color-space. The HOG features shown here were extracted from the three different channel of the car picture shown in Fig.1, after conversion to YCrCb color-space.

The HOG features were extracted using the function `get_hog_features()`. This function is defined in “functions.py” lines 15-28.

The parameter values that I chose seem to be a good compromise between performance and accuracy of the classifier (see later sections).

Spatial binning and color histogram

Color histogram features were also extracted from each image. Before extracting this other set of features the original images were converted into the HLS color-space and the histogram calculated only on the H channel.
Fig.3 shows the histograms of the H channel of the car and non-car images shown in Fig.1.

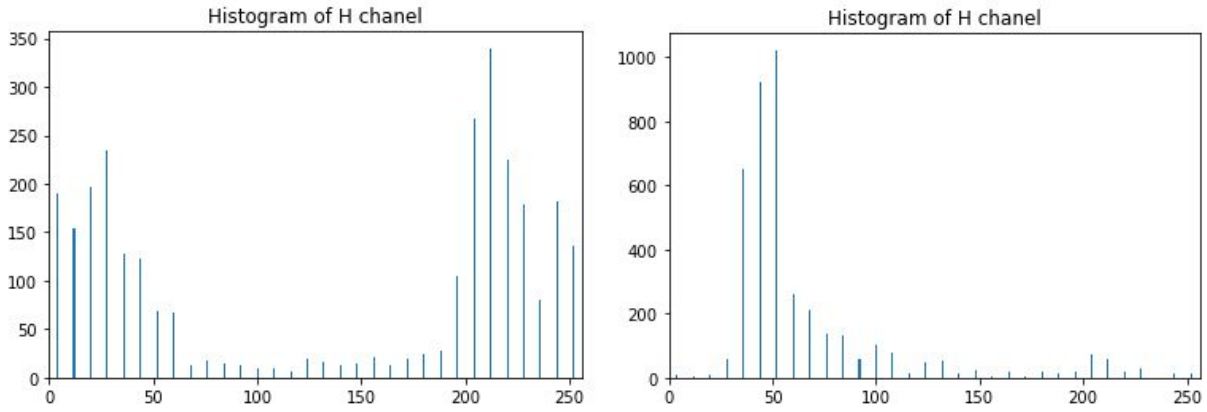

Figure 3 - Histograms of the H channel of the car and non-car images shown in Fig.1.

The code to extract color histogram features is contained in the `color_hist` defined in “functions.py” lines 40-58.

Spatial binning of colors

Raw pixel values were also extracted from each image as an extra set of features. To reduce the number of raw pixel features while maintaining the main visual characteristic of the images, each image was resampled with a spatial binning of 32 x 32.
The code for the spatial binning is contained in the function `function bin_spatial`, defined in “functions.py” lines 32-37.

Fig.4 shows the raw pixel histograms for the car and non-car images shown in Fig.1

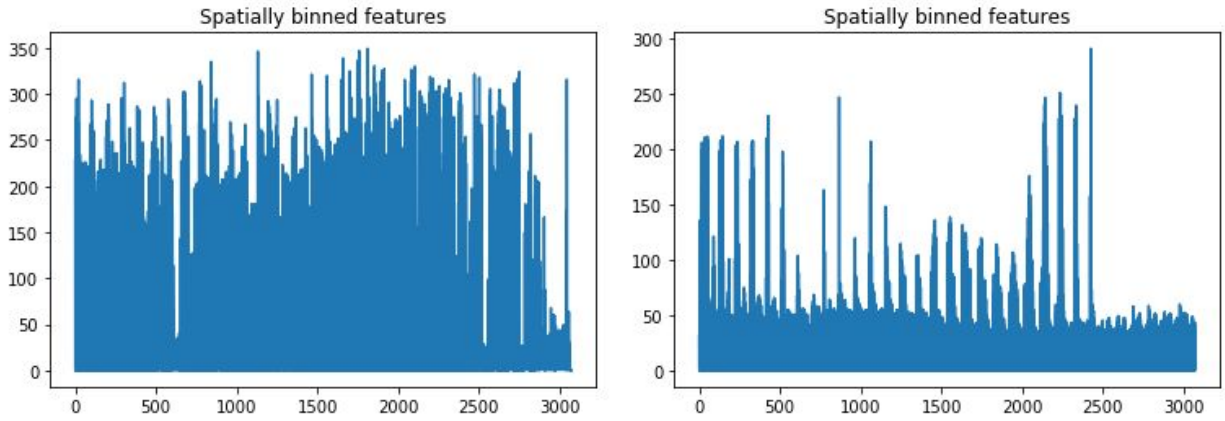

Figure 4 - Spatially binned features for the car and non-car images shown in Fig.1.

Classifier

The classifier trained to distinguish between vehicle and non-vehicles images is a SVM with linear kernel (the C parameter was set to its default value).
The features were scaled to zero mean and unit variance before training the classifier.
The classifier performed with an accuracy of 0.99% on the test set.
The code used to create and train the classifier is contained in the file `create_and_train_classifier.py`.

Sliding Window Search

The sliding window search was implemented by classifying the content of a patch of the image as car / non-car, where the patch (sliding window) was moved systematically from left to right and top to bottom manner.
The relevant parameter value for the sliding window search were set as follows:

- pixel_per_cell = 8 (in both direction)
- cells_per_block = 2 (in both directions)
- cells_per_step = 2 (overlap 75% between adiacent window positions)

Because cars appear in the image in different sizes (due to perspective) it makes sense to have more than one sliding windows, each with a different size, so as to be able to detect cars regardless of how far (small) or close (big) are to the camera.
Rather than changing the size of the window, I introduced a scale factor that change the size of the image. This is effectively the same as changing the sliding window size.
The higher the scale factor, the smaller the image becomes, the bigger the portion of the image covered by the sliding window.
Fig. 5 shows the detection of two cars in a test image using the sliding window technique.

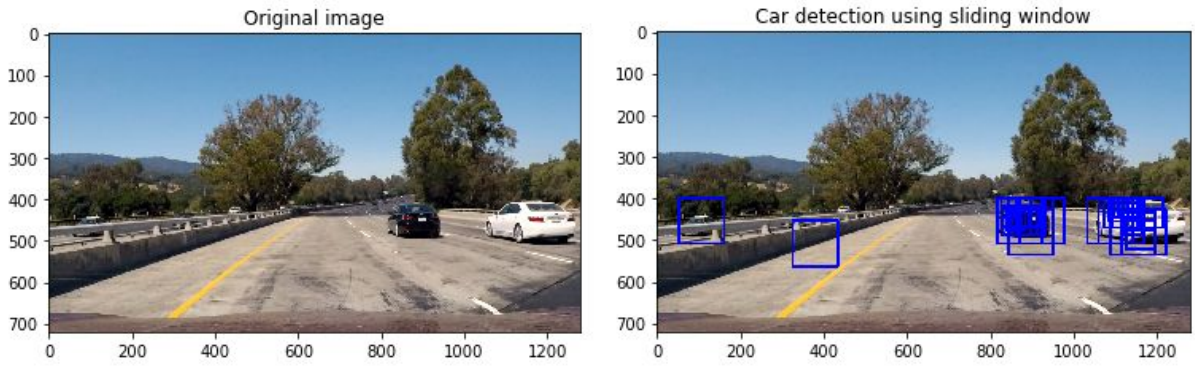

Figure 5 - The original image (left) is processed with a sliding window search. The result of the search (right) shows blue boxes drawn around a region of the image where the classifier identified a car.

False positive and multiple detections

To deal with false positive and multiple detections of the same vehicle, I created a heat map where the intensity of each pixel represents the number of boxes in which that pixel is contained. Subsequently, a threshold was applied so as to remove from the heat map those pixel contained in a number of boxes that was smaller than a given threshold (for example 2).

The result of this procedure is shown in Fig.6

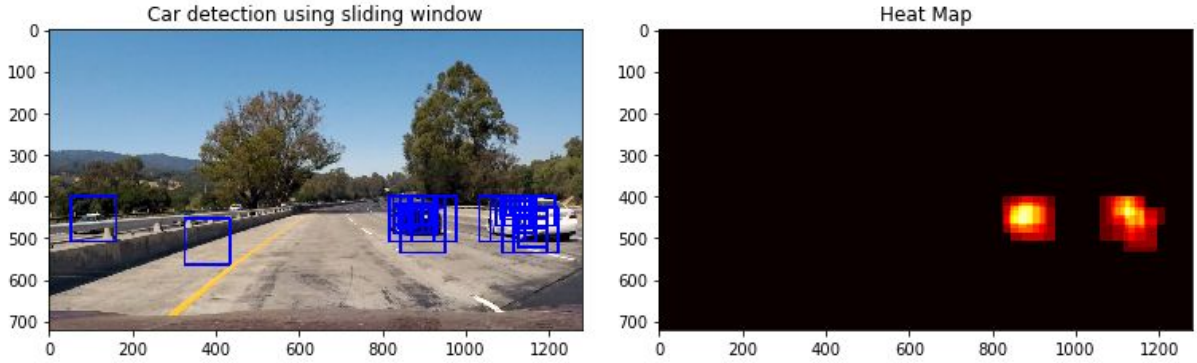

Figure 6 - The result of the sliding window search (left) was used to generate a heat map to filter out false positive.

Finally, to end up with a clean representation of the position of the different cars, I used the functions `numpy.clip()`, `scipy.ndimage.measurements.label()` and `draw_labeled_bboxes` (called in find_cars.py, lines 95, 97, 99 respectively). The result is shown in Fig.7.

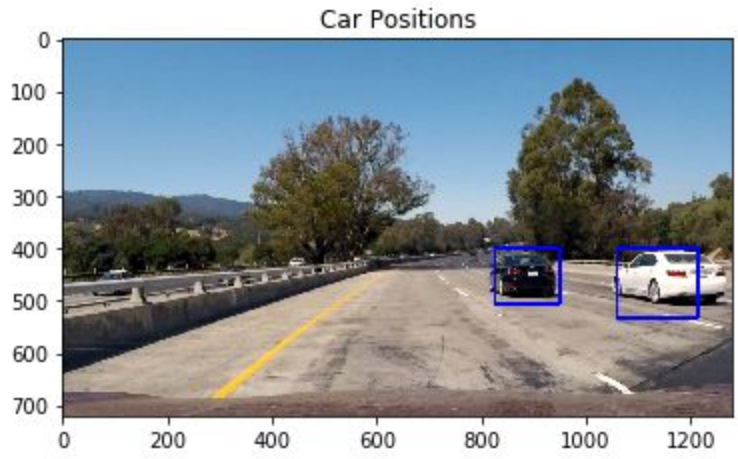

Figure 7 - The heat map shown in Fig.6 (left panel) is used to generate a clean representation of the car positions through specific functions (`numpy.clip()`, `scipy.ndimage.measurements.label()` and `draw_labeled_bboxes`).

Video Implementation

To filter out false positive and make the bounding boxes less wobbly, I implemented the same filter on the heat-map discussed in the previous section. However all the boxes generated by the sliding window search on 10 consecutive frames were considered in the filtering process. The video is provided as part of this submission.

Discussion

There is substantial flickering in the bounding box drawn in the project_video_output.mp4. This flickering is mainly pertaining to the white car. This is possibly due to the color of the car. However including color feature extracted from the HLS color-space should minimize the effect that color has on detections of cars.
This is something I need to investigate more.
Also, finding the right trade-off between the number of frames upon which to apply the heat-map filter and the filter threshold may need some more exploration as it possibly play an important role in smoothing out the bounding boxes flickering.