Vehicle Detection and Tracking

The goals / 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
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to 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 your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) 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.

Histogram of Oriented Gradients (HOG):

Code for this is contained in code cell #2, and the function name is get_hog_features(). This function is used to extract hog features for given single channel image along with other attributes such as orientation, pixel per cell, etc.,

This same code cell contains other functions to converts the color space of the images, special binning of images and feature extraction based on histograms.

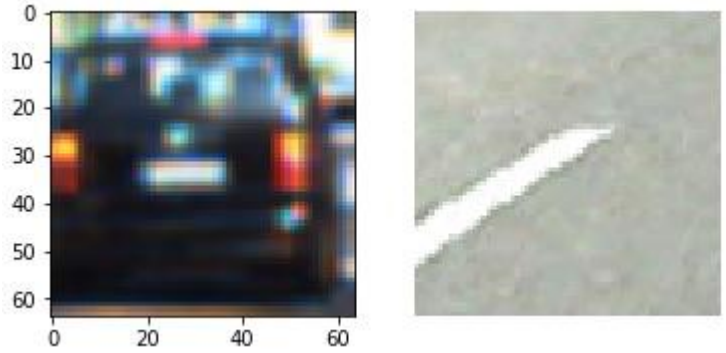
Code cell #3 has a function extract_features() to extract all the features that are needed to train the classifier, based on the parameter values, this function extract features and stack them together to feed it to classifier.

Code cell #4 contains the find_cars() function which is taken from the class lectures and modified slightly to work with the images to do sliding window technique. This function resizes the image based on ‘scale’ parameter and slide the window across the given region to get the sub-image cutouts that are fed to classifier to predict the sub-image class. Based on this result we are keeping track of all the sub-image locations that are cars and using them in later steps to get the tight bounding box for the car.

Code cell #5 I'm just creating a list of all training sample image paths, 'cars' contains list of car image paths, 'notcars' contains list of non-car images paths.

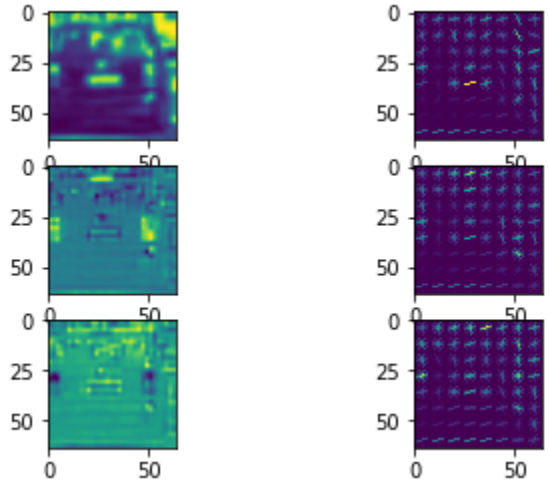
Code cell #6 is to just show a random car and not-car pictures just to make sure that we are reading correct data

Here is an example of random car & noncar image

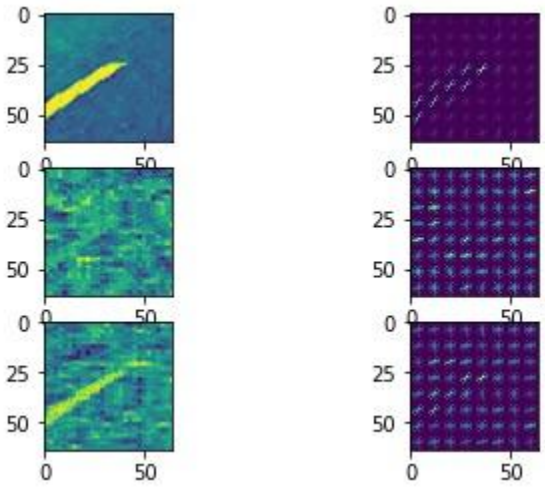


Code cell #7, #8 shows the hog features of car and non-car images for 3 channels for YCrCb color space. I choose YCrCb color space because this color space retails more information about cars shape, and also may other students in the forums has suggested this color space to work with.

Here is an example of Hog features extracted from car image:



Here is an example of Hog features extracted from noncar image:



Code cell #9 is where I'm extracting features from cars and noncar images, I choose to use 'YCrCb' color space for the reasons stated above, orientation = 9, pix_per_cell = 8, cell_per_block = 2, hog_channel = "ALL", special_bin = (32, 32), hist_bins = 32, and using all the three features (hog, histograms, special binning) to train classifier. These numbers are chosen after many random number, because these seems to have a good performance on classifier training.

Once the features are extracted, using StandardScalers' transform function to changes the features distribution with zero mean, unit variance, this is important because without this the classifier will be biased towards the high numbers in features.

Code cell #10, I'm splitting the training and test features using train_test_split to split 20% of data for test set and remaining for training.

In Code cell #11, I choose LinearSVC as classifier because it is proven good when we have binary classes (in our case 0 -> noncar, 1-> car), using train features and train labels, I trained classifier.

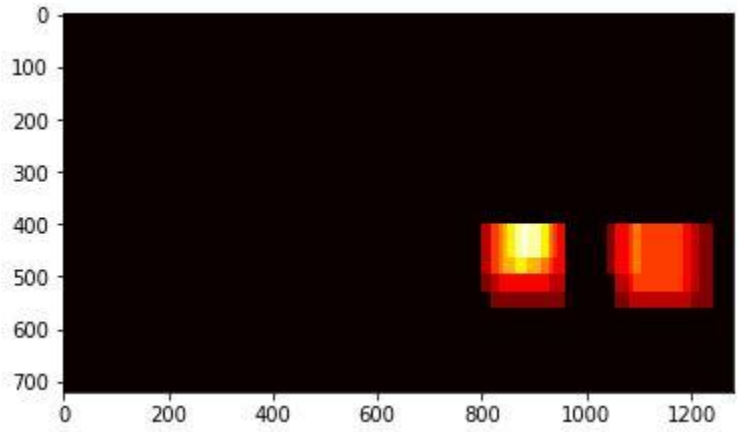
In Code cell #15, get_boundaries() function takes an image and uses different scales and different search boundaries to search for car in the image. I specified different regions with different scales because, cars at horizon looks smaller compare to cars near to the camera, using different scales and different search boundaries, we can reduce the number of search windows effectively improving the algorithm speed.

In code cell #16, I'm plotting original image, searched boundaries along with found cars boundaries, as seen below first column is the test image, second column is the region we search for cars, third column shows the boundaries for cars

Here is the example image that shows the search boundaries and found cars:

In code cell #17 I added functions that are used to remove duplicate boxes found around the car and to draw (draw_labeled_bboxes function) a single tight box for multiple boxes.

Code cell #18 shows the usage of functions from code cell #17 to show tight bounded box.



Code cell #19 is where I'm defining the pipeline for the image, this pipeline takes image as input and then find the car boundaries for the image, by applying heating techniques, it gets a

tight bounding box for duplicate boxes also eliminate any boxes which are less than threshold level. Here I'm also keeping track of last 15 frames information just to smooth out the cars found in new image.

Code cell #20 shows using of pipeline() from code cell #19 and here is the output from pipeline shown in right column

Code cell #21 and #23, I'm using moviepy functions to read the video stream image by image and apply pipeline() on the image and writing it to test_video_output.mp4, project_video_output.mp4 files respectively which is attached in the zip file.