Vehicle Detection Project

-Goal:

Process the project video to detect cars and draw a rectangle around them

-Steps:

. Extract HOG features, binned color features and histogram of colors and combine and normalize the features to create the final features for the labeled set of provided image set.

. Train a classifier using the features of the image set

. Implement a sliding-window technique and use the trained classifier to search for vehicles in images.

. Create a pipeline to process each window and 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.

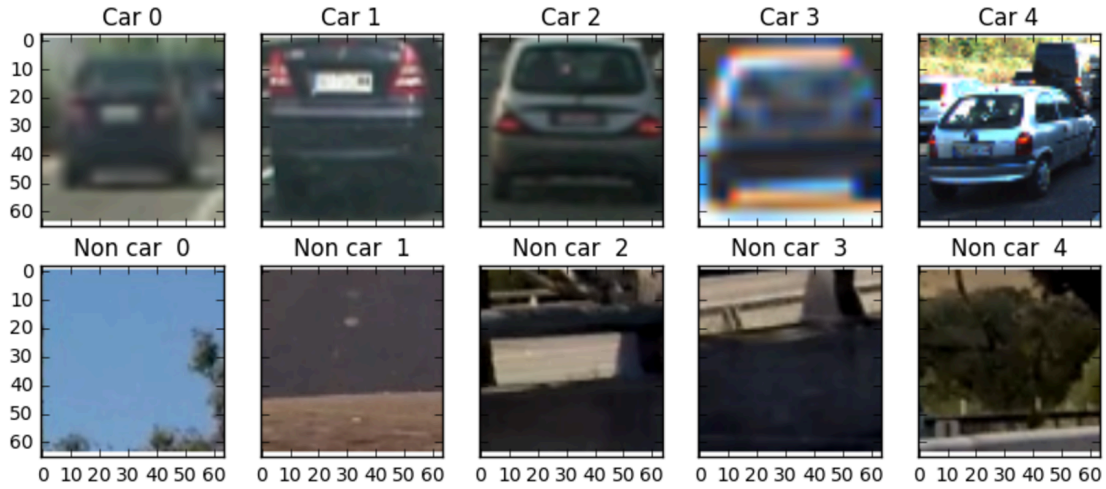
. Estimate a bounding box for vehicles detected.

Project steps:

-In cell 1 of IPython notebook, I added the functions provided from the course lesson to have the functions to calculate the HOG features, bin spatial and color histogram features, a function to combine the features for set of images, a function to get the features of a single image, functions to draw boxes and search an image for input windows.

-First we need to load the images. I did this in cell 2 that adds the path for all the images to 2 list of “cars” and “notcars”.

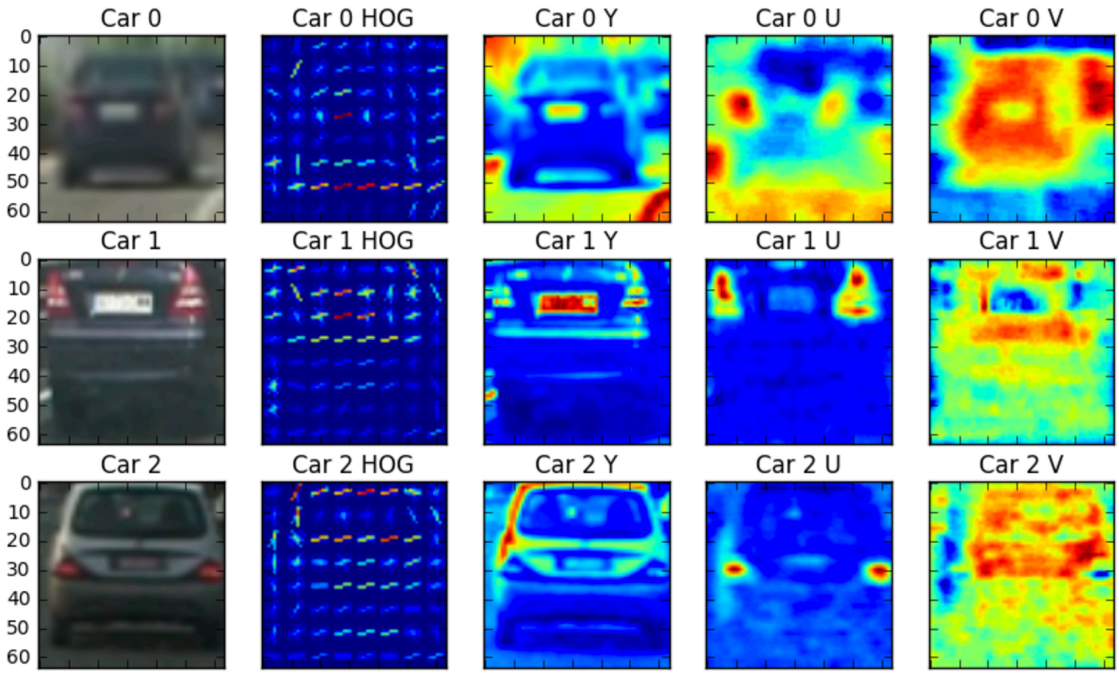
In cell 3 I drew some images of cars and non-cars. Here’s the result:

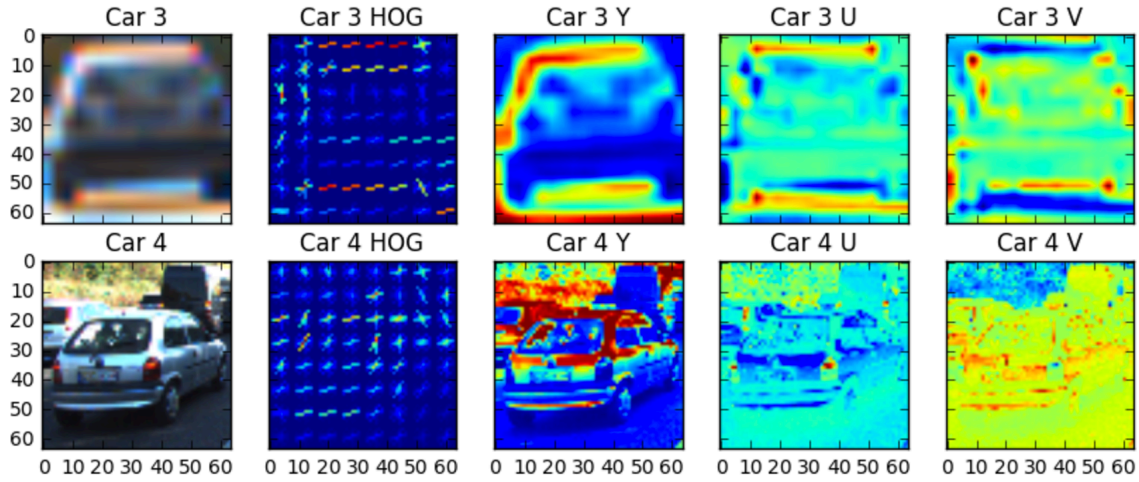


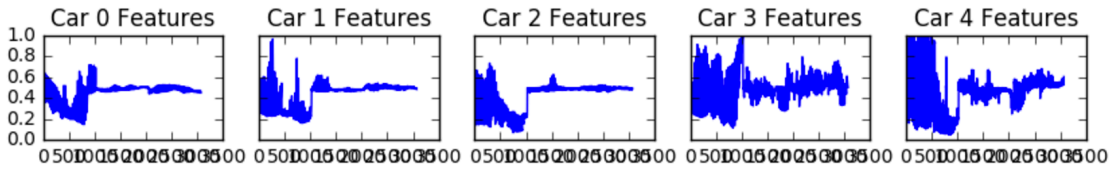
- We need to extract the features. To have some sense of the features and how they could be different or similar between 2 classes I drew some of the features.

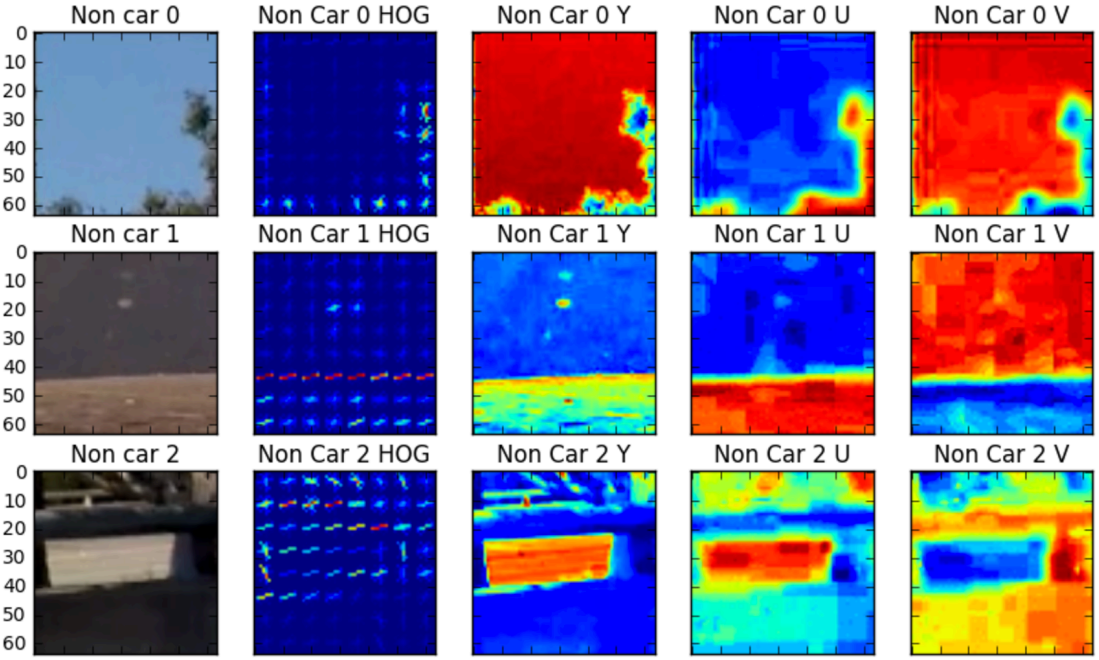
Please note: I tried different features with different color spaces and parameters but what I am adding here are the final ones after I decided to use YUV as color space.

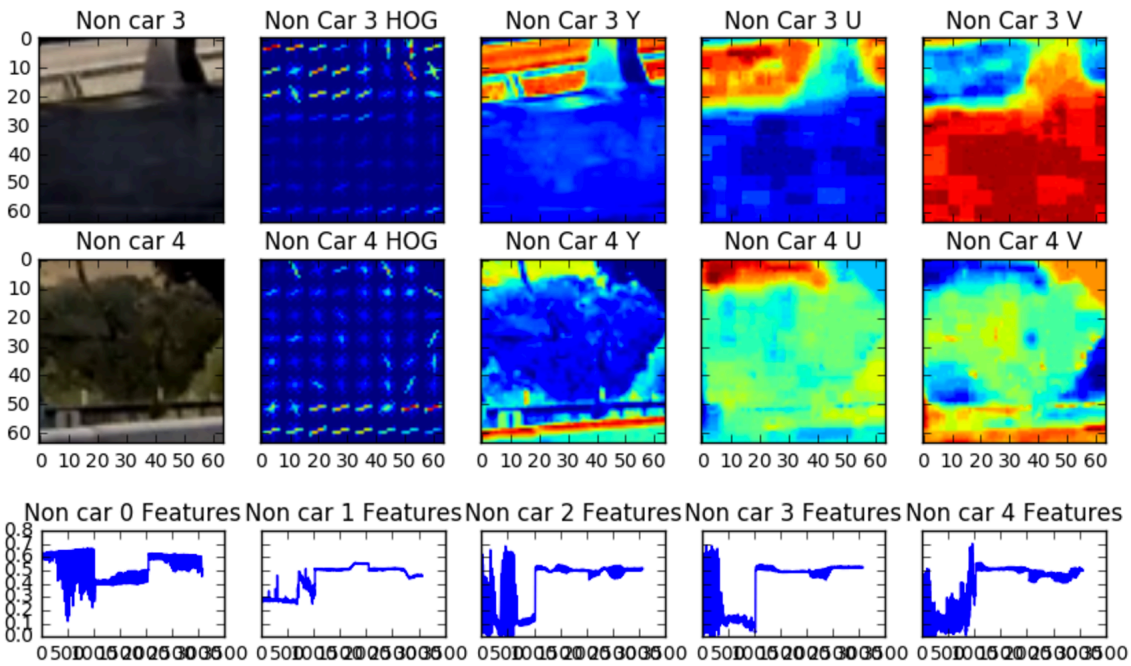
I drew the HOG features and Y, U and V channels for the sample images. Also I plotted the bin spatial features. I did this for cars and non-cars. Here's the result:




Car 3
Car 3 HOG
Car 3 Y
Car 3 U
Car 3 V
Car 4
Car 4 HOG
Car 4 Y
Car 4 U
Car 4 V


Car 0 Features
Car 1 Features
Car 2 Features
Car 3 Features
Car 4 Features


Non car 0
Non Car 0 HOG
Non Car 0 Y
Non Car 0 U
Non Car 0 V
Non car 1
Non Car 1 HOG
Non Car 1 Y
Non Car 1 U
Non Car 1 V
Non car 2
Non Car 2 HOG
Non Car 2 Y
Non Car 2 U
Non Car 2 V



Looking at the sample features from cars and non-cars, it looks like there are visible differences to distinguish between 2 classes and hopefully the classifier can learn the 2 classes well.

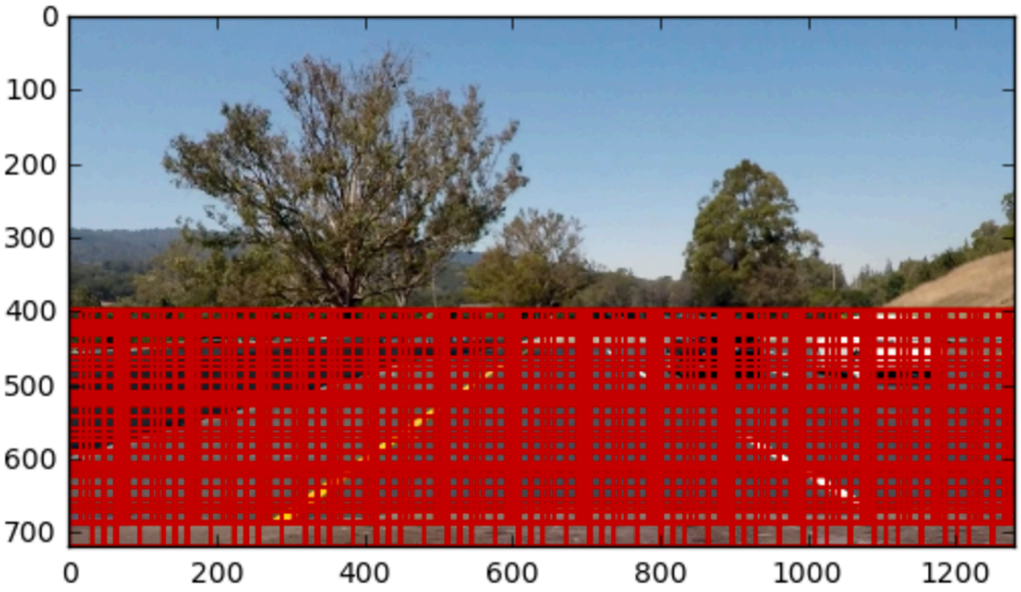
-In cell 8, I used the “extract_features” function to extract the features for cars and non-cars. I used a combination of all 3 features available to me: HOG features, spatial bin features and histogram features. In this cell I load the actual files from the file paths and the real data will be stored in “car_features” and “notcar_features” variables.

-In cell 9, I trained the LinearSVC model. Before training, the data needed to be shuffled and also normalized. I tried other classifiers too like SVC, RandomForestClassifier and AdaBoost. Although each of them looked a little better in some areas of the final video, none of them was much better and all of them took longer time for training and classifying than LinearSVC. So I stayed with LinearSVC.

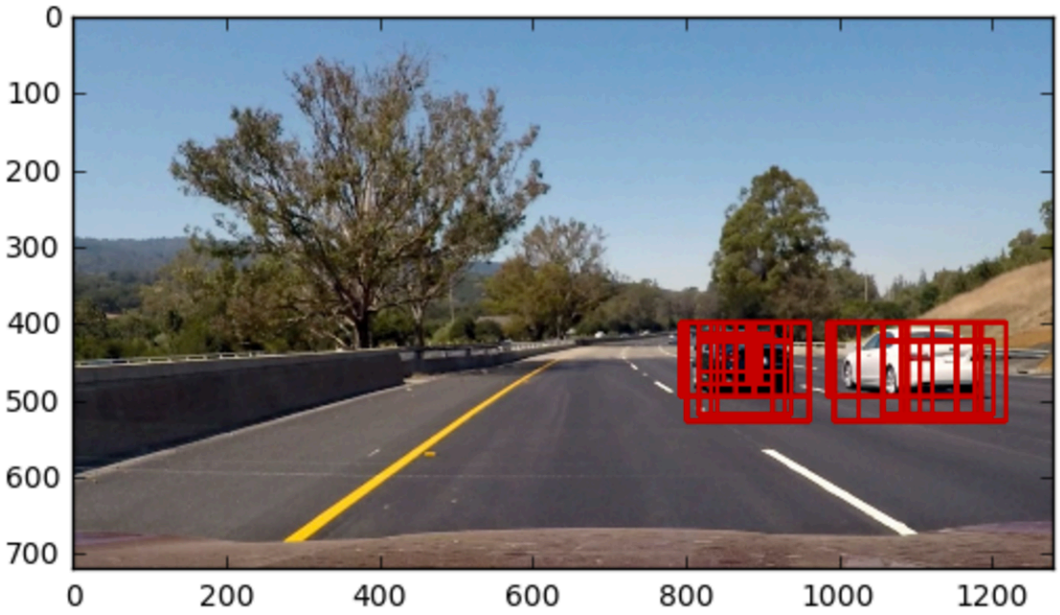
The test accuracy was more than 99% in most of the trials.

-In cell 10 I added the provided functions from lesson to use for heatmap generation. I also added another function “get_all_windows” which generates windows of different sizes and returns a list of windows. Doing many trials I found combination of windows of sizes 64,96,128 and 192 and also overlap of %75 gives me a good result.

-In cell 11 I counted the number of windows and also drew all of them to have some idea of the area that windows scan:

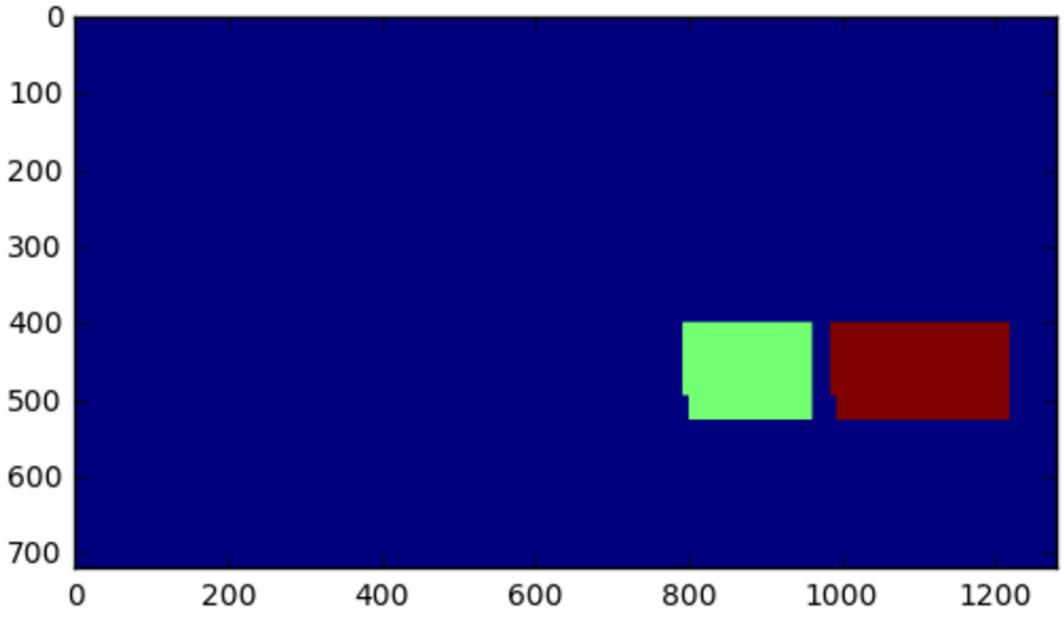


-Now it's the time to search the image for the cars using the windows. I do this in cell 12. The result is this:

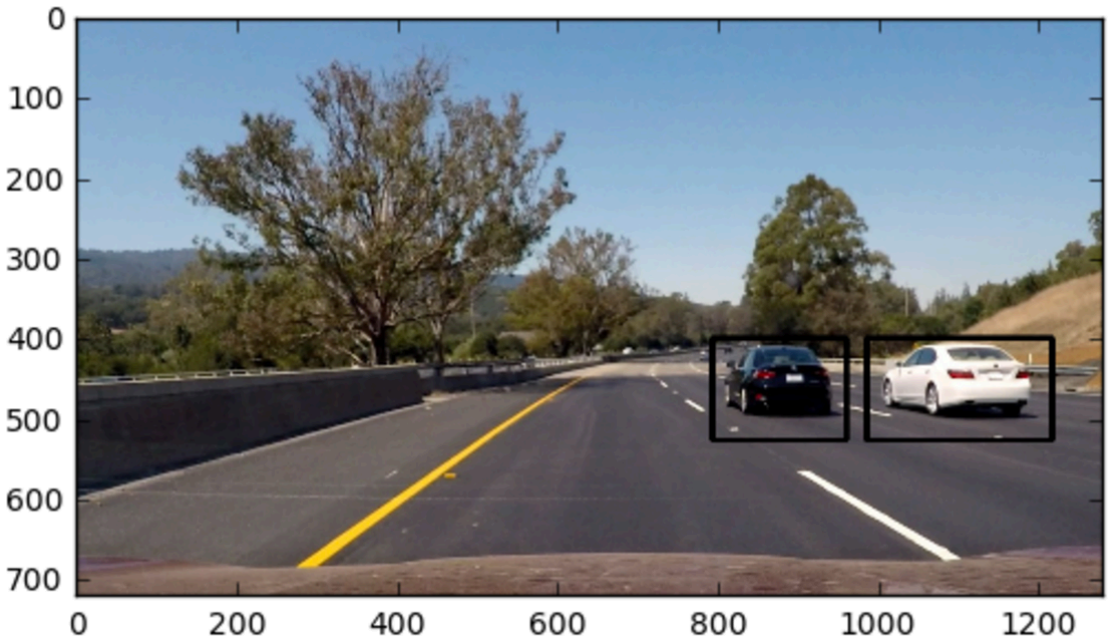


There are several overlapping windows for each car that we need to use heatmap to find one window for each car.

-Using heatmap and threshold and label functions I generated the heatmap image in cell 13 as follows:



And then drew the combined image from the original image and labels as follows:



-Originally, I generated the video using a pipeline using only the above steps. Although it was finding the cars but there were some issues:

1-There were some outliers/false positives detected by the classifier.
2-The bounding boxes for cars didn't look continuous and in some frames they were not showing because classifier didn't detect a car although there was a car.

So we need to add some logic to remove the false positives and also make the true positives smooth by averaging them.

I created a queue that holds bounding boxes for “q_max” number of frames. Also I wrote the function “compare_boxes” which tries to find the boxes that seem to be the same in several consecutive frames by comparing the distance between their centers.

In the pipeline, I get the current image from the video and find the hot windows, labels and bounding boxes and then I find the average bounding box for this frame and previous frames in the queue which pass the “compare_boxes” function criteria. I add the result average bounding box to “final_windows” list and drew the boxes of this list. I also add the boxes from current frame to the queue to be used for averaging in next frames. As a result, I generated a video that has almost no outliers and smooth bounding box moving with the cars.

Discussion

There are many parameters for this project that affect the performance and quality of the final result. In this project my first intention was to “make it work” and get an acceptable result but as I was trying different parameters I could see that by spending more time and effort the result can get more robust:

-The features can be selected smarter to have smaller set of features that make processing faster.
-Also with any set of features selected, the parameters of HOG and color spaces can be tuned to get better result.
-Different classifiers could have better result although may need larger training time.
-The size, number and overlapping percentage of windows can be fine tuned to get optimum result.
-The averaging logic can be developed to be smarter.

I also used the output video of this pipeline to the “Advanced lane line detection” pipeline that created a pretty good result and I sent it with the files of this project.

Although the correct way of generating the combined video of 2 projects would be having a single pipeline to add the lane lines and car detections all together, I didn’t have time to do it because of the deadline.

Thank you for your time.