Project 5: Vehicle Detection

Rubric points

Write-up / README

1. Provide a Write-up / README that includes all the rubric points and how you addressed each one. You can submit your write-up as markdown or pdf. Here is a template write-up for this project you can use as a guide and a starting point.

You're reading it!

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

"P5-Vehicle_detection-Exploration-ver2", **Check cells 3 to 13**

Before extracting the HOG features, I decided to have a look at different color spaces representation of car and non car images as suggested in the learning material videos.

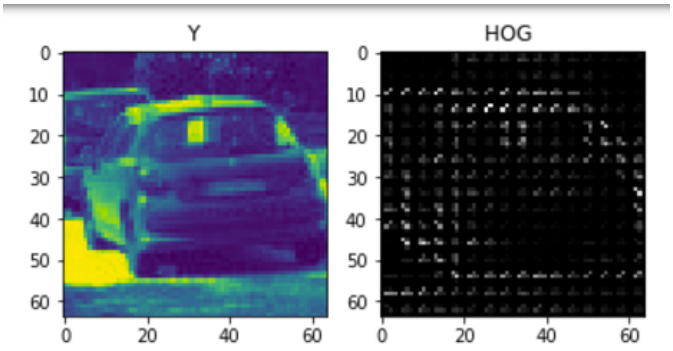
Conclusion RGB line space shows no special patterns or clustering of colors that can distinct cars from non cars images. However the S (Saturation component in the HLS and HSV) and Cr (in the YCrCb) showed clustering of color components in their dimensions.

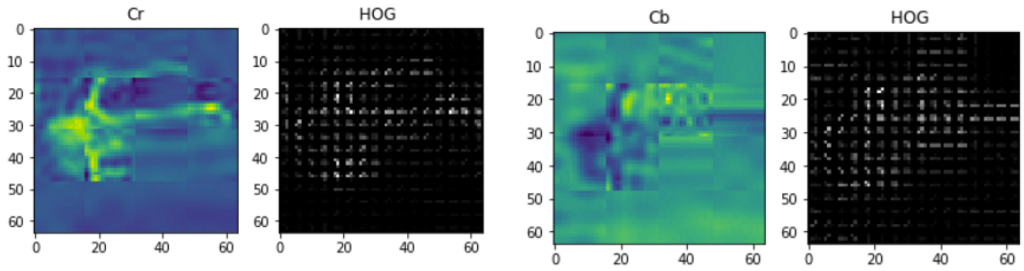
Therefore at that point I decided to consider HLS, HSV or YCrCb as color spaces for HOG feature extraction. And finally chose YCrCb, after an intirative trial and error to get the best performance of the classifier.

Then I used the imported skimage function "hog" to extract the hog features, and defined a function named: get_hog_features, in cell #2. The function takes in the image, and all the hog related parameters, and returns the features + the image in some cases.

I also tried to visualize the hog output of different current spaces in cells #15 -19

I will learn afterwards that including the 3 channels of YCrCb would give higher accuracy to the SVM model.





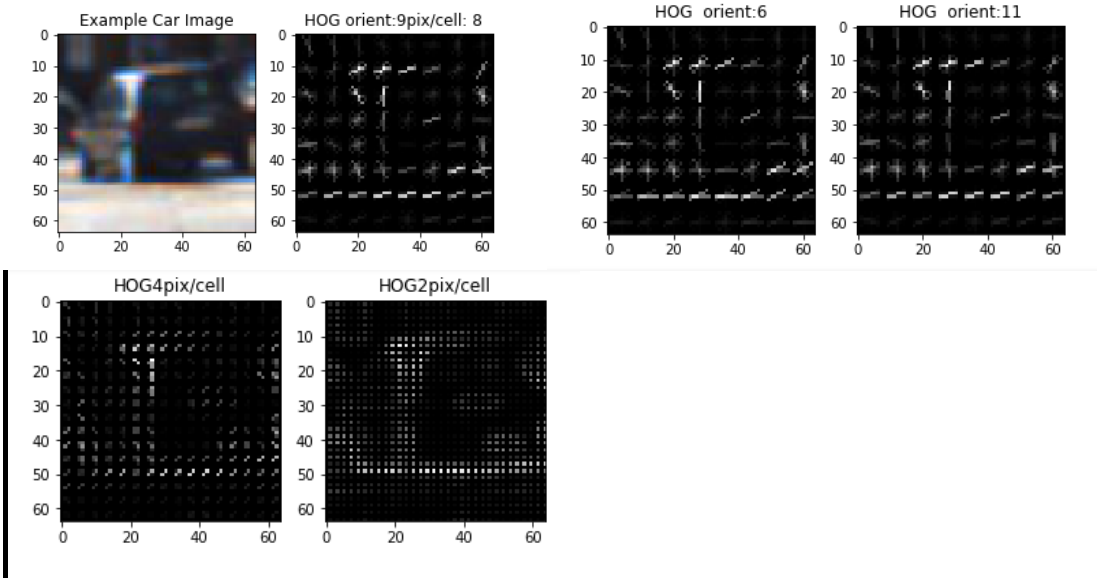
2. Explain how you settled on your final choice of HOG parameters.

"P5-Vehicle_detection-Exploration-ver2", **Check cells 14 to 19**

This first method, was to visualize the HOG features, and try to get some intuition on how to increase the accuracy of the classifier, by feeding an image that looks some how like a car after transformation.

In Jupyter notebook named "P5-Vehicle_detection-Exploration-ver2", you will see how I was testing different parameters to reach a balance between intuition / visually makes sense, and the no of parameters. The tradeoff is, if I achieve higher resolution HOG images, which probably looks more like a real car but with dotted lines, then this means more number of features to process.

Conclusion was, increasing number of orient, decreasing pixels per cell and decreasing cells per block would give a higher resolution HOG feature, but can considerably expand the size of the features per image fed to the classifier, hence more expensive model.



However, this was an iterative process. I was first trying to balance between the size of features used as inputs for the classifier, and the accuracy of the model. But I found myself

continuosly changing the paramters even when I was working on my video pipe-line to reduce the amount of false positives.

For example, I started by using 8 cells per block, and 8 pixel per cell. But at the end I increased the size of features by adjusting it to the below parameters list.

I also included all 3 channels while extracting the hog features to extract hire accuracy from the model

On top of the HOG features, I activated spatial features of size 32x32, and color histogram for YCrCb color space, which showed a unique color pattern as showed in the notebook, and images in this document.

color_space = 'YCrCb'
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL"
spatial_size = (32, 32)
hist_bins = 32
spatial_feat = True
hist_feat = True
hog_feat = True

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

File "P5_VehicleDetection_VideoPipeline", Cells #8

N.B. Normalization visualization from file "P5_VehicleDetection_VideoPipeline", Cell# 20

The classifier used is a support vector machine for classifying cars and non cars images. I first prepared the data used to train the model by using the project related resources. And made sure that the training data is balanced between cars and non-cars.

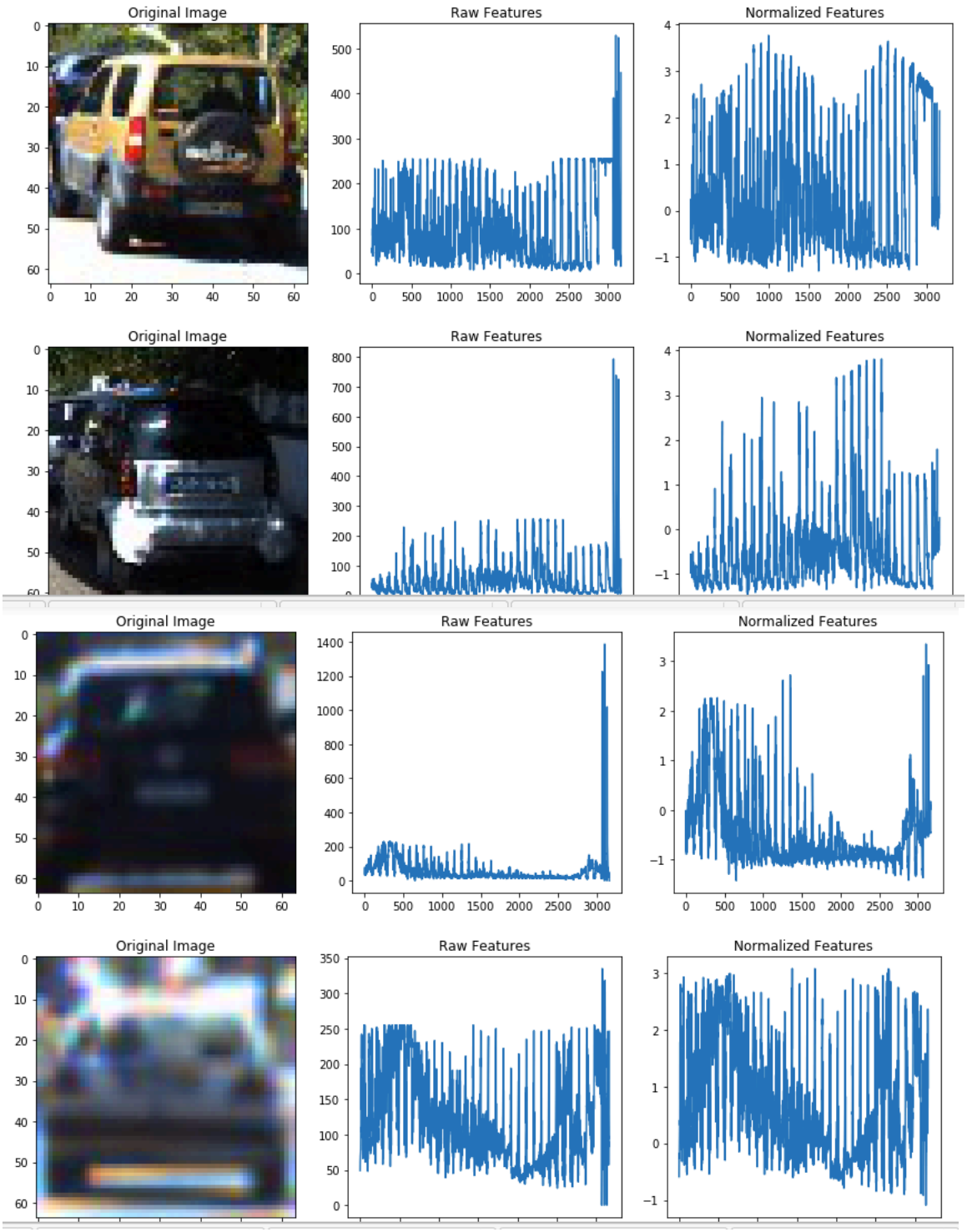
```
number of car images:  1196
number of non-car images:  1125

Using: 9 orientations 8 pixels per cell and 2 cells per block Feature
vector length: 8460
2.48 Seconds to train SVC...
Test Accuracy of SVC =  0.9978
```

I later on suspected that more data would have helped the SVM model to perform better. Especially if I train my model on more photos of cars from a side view, to better detect cars approaching from the right hand side of the car.

Next I labled the data, to cars and non-cars, shuffled them, and split them for training and testing data sets (to measure accuracy of the model). And achieved an accuracy of 99%.

On important step to do before training the model is to normalize the input features, so that no set of features have more influence over the others.



Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

File "P5_VehicleDetection_VideoPipeline"

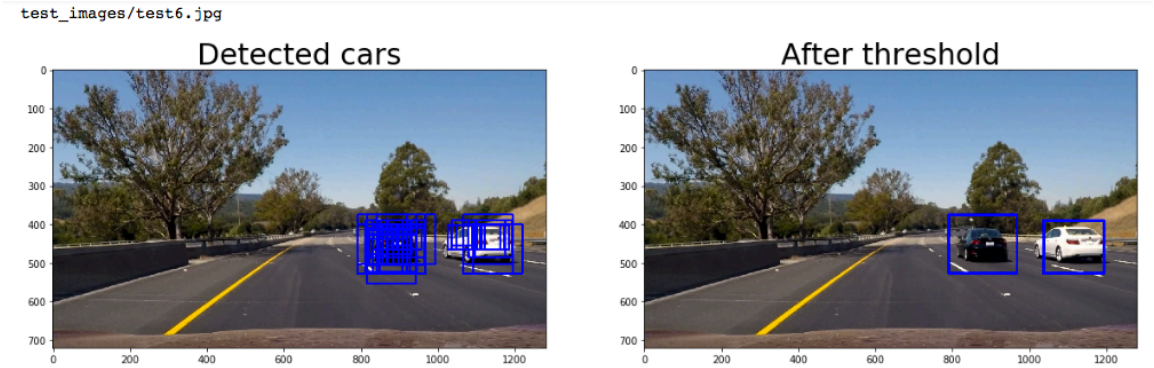
Cells #55: Final implementation
Cell #17: Exploration of different sizes, and start stops for x and y parameters.

I started by exploring the regions and sizes of windows. After many and many experiments I concluded:

- For this exercise, I can safely ignore the left side of the image. i.e. I can start sliding window from the center of the image then slide the windows to the right.
- Y start and stop is determined in order to exclude the sky from running sliding window. Also bigger sized windows, that are suitable for detecing cars closer to car, should have a Ystart point that is lower than those far at the horizon.
- Increasing the overlap will give a tightly wrapped box around the detected car at the end. However the increased overlap places more demand on processing needs. Since I use a normal process, not a GPU, it takes around 3 hours to process the project video, I tried to reduce overlap as much as possible, but to keep the quality.

2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

File "P5_VehicleDetection_VideoPipeline"
Cell# 17



To optimize the classifier I had to do the following:

1- Increase the accuracy of the classifier, to avoid false positives from the start, instead of having to deal with them using heatmap threshold, which can reduce the performance of the algorithm.
2- Adjust the prediction threshold. At the start it was set to 1. But I reduced it to be >=0.5, after testing with many values.
3- Added 3 sizes of windows, and looped the original sliding windows function we learnt in the videos to increase odds for detecting cars that are closer to ours.

Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Here's a link to the final video:

https://www.youtube.com/watch?v=5DUVEqlQl20&feature=youtu.be

If you want to see how my output videos improved over time you can look at my attempts in the below links:

3rd attempt: https://www.youtube.com/watch?v=6OQ-naaAABY

2nd attempt: https://www.youtube.com/watch?v=tgaAW3Ym4G0

1st attempt: https://www.youtube.com/watch?v=lJ40qdUoRTc

filter for false positives and some method for combining overlapping bounding boxes.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

If I would have time, I would implement a less costly algorithm, where HOG features for the whole detection area is extracted once, then sliced to window sizes. This will optimize the code for real time processing.

In the middle of the video, the algorithm fails to detecte the white car on the right, as it goes further down the horizon. Given time and resouces I would do the following:

1- Augment my data with white cars (and non-white), and different sizes, and with photos from the side
2- Experiment more with SVM hyper parameters

Finally, I need to come up with a method to make the boxes look smoother on the screen and to tightly wrap the car boundaries in a clean way. One method to do so , is via a class that can store the exact location or distance form the car of the object at each frame.