Writeup

Vehicle Detection Project

The goals of this project is the following:

In this project, my goal 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 are available in the project repository. There is an writeup template in the repository provided as a starting point for your writeup of the project.

Rubric Points Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters. Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.

As taught in the course, I introduced the hog tool from skimage.feature, so hog method can be easily handled with just lines of code. In order to apply it to images, get_hog_features() is defined.

In this function, below tuning parameters are considered.

color_space = 'YCrCb' # Can be GRAY, RGB, HSV, LUV, HLS, YUV, YCrCb

hog_channel = ‘ALL # Can be 0, 1, 2, or ‘ALL’
above 2 parameters are aligned. And I tried loads of time to tune it until a better performance achieved. Those

setups are more preferred: L channel in HLS and LUV, V channel in HSV, Y channel in YUV and YCrCb. In general, lightness is better than hue and saturation.
If I choose all 3 channels, the computational load will double, so I stick to single channel for further training to achieve a quicker trail and error loop with other modules.

orient = 9 # HOG orientations. represents the number of orientation bins that the gradient information will be split up into in the histogram. Which in range 6 and 12 is preferred, I choose the mid-value

pix_per_cell = 8 # HOG pixels per cell, parameter specifies the cell size over which each gradient histogram is computed. I just tried with 8 and 16, to make comparison, I Assume a smaller number will leads to an better off accuracy. Meanwhile to sacrifice the computational power. And that's my approach

cell_per_block = 2 # HOG cells per block, it specifies the local area over which the histogram counts in a given cell will be normalized. and generally leads to a more robust feature set. I did not pay much attention to tune it in my case.

spatial_size = (32, 32) # Spatial binning dimensions. Spatial size is quadratically related with the feature size, so 64*64 is a bad choise for me. And 32*32 is a compromise w/o loosing too much information.

hist_bins = 32   # Number of histogram bins, default value without too much of the considerations. It is a RGB color space inside the color_hist(), I did not trust in it to contribute most of the weight in all the features, so 3*32 could be reasonable.

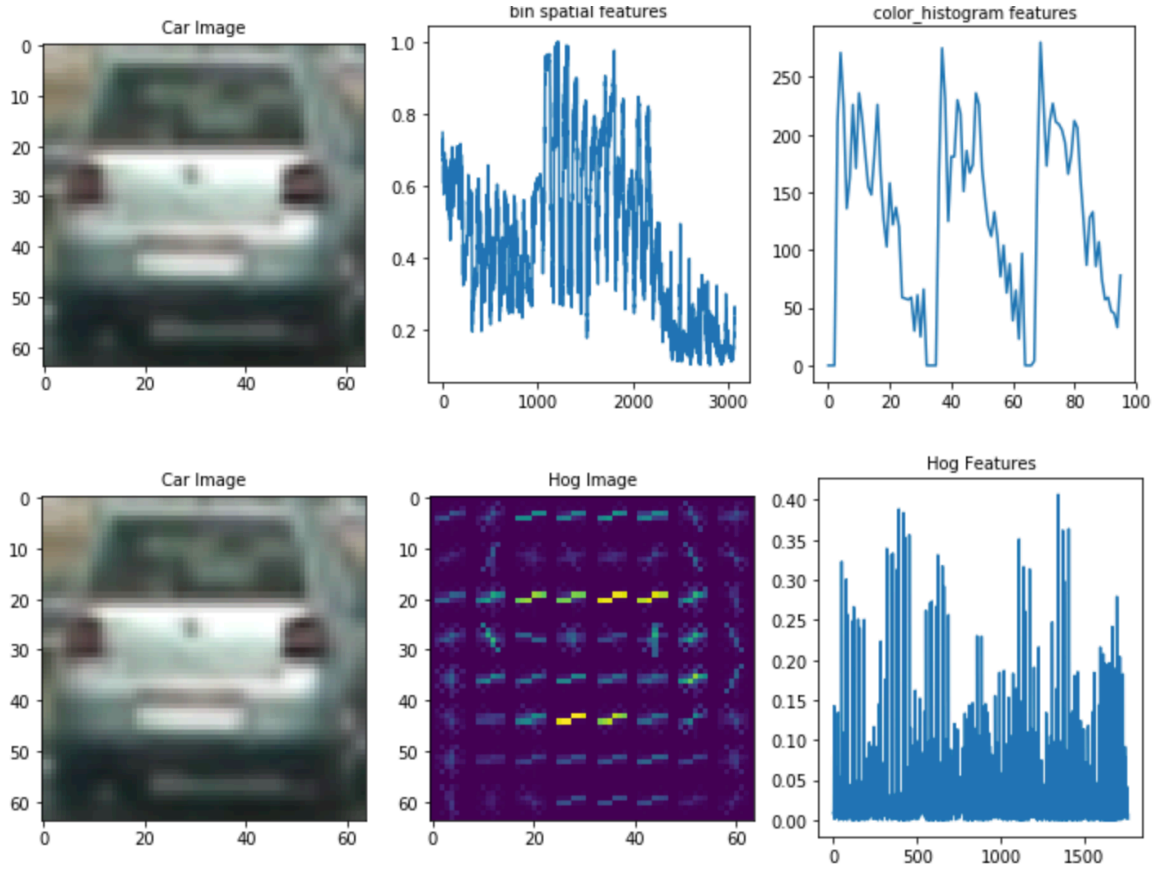
spatial_feat = True # Spatial features on or off.   I make it On

hist_feat = True # Histogram features on or off   I make it On

hist_range = (0,256) no need to tune any more

hog_feat = True # HOG features on/off I make it On.

in general, I evaluate the performance of those parameters in a later part at the SVM optimizer section. Where computational results is given to the matching rate.



note: above setup is my first submit, afterwards I tuned a bit of those parameters in my 2nd submit. Details are attached in the end of this writeup.

2.Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them). The HOG features extracted from the training data have been used to train a classifier, could be SVM,

Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.

I'm using the default starter code in the lesson, as in the video it mentioned, SVM is a good start to try.

```
from sklearn.preprocessing import StandardScaler
# Create an array stack of feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)
```

Key point here is to normalize the data to a zero mean and unit variance is ensured with this StandardScaler(), as in the last checkpoint I visualized all 3 feature sets, and they have a totally differed scale. From decimal to hundreds.

3. 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? A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.

slide_window() do it for me. The scale for a search is defined inside multi_slide_windows() . where I introduced 4 different sized scanning blocks

(80*180,128*128,96*96,80*80), for each kind of window, I

assigned differed scanning area, as a prior knowledge that a perspective rule regulates a fixed horizon(vanishing point), so one min_y can fit all scales, and the far the object, a smaller max_y needed.

I define the overlap with (0.75,0.75), main purpose is to get an accurate profile, if 0.5,0.5 applied, always the block intersect with the car.

4. Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier? Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)

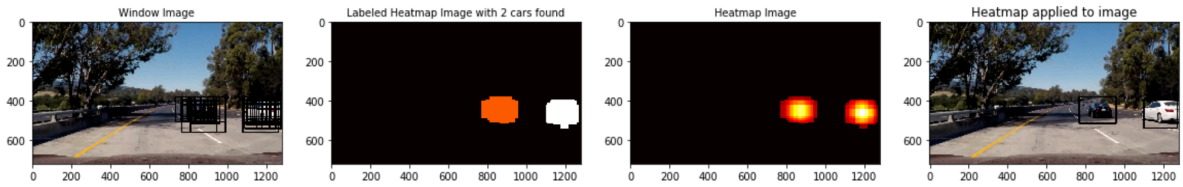
in order to improve my classifier, I pick out main parameters and focus myself mainly on the color space and channels. I activate all 3 classifiers including bin_spatial, color_histogram, and hog transformation.

Those methods are totally unlike each other, being equipped with such a triple fail-safe mechanism, I can maximize the possibility to find true positive features. Here I have to balance the my computational power and accuracy. If a higher resolution setup is used, the run time rocketing.

In the end, I choose YCrCb color space, hog_channel set to 2, instead to put it 'ALL', I sacrifice no the resolution. With this combination, I can achieve a pretty fine performance with my test images and the video. 5 hours consumed in this case for video processing.

```
    windows = multi_slide_windows(img)
```

```
    hot_windows = search_windows()

    window_img = draw_boxes()
heatmap = add_heat(heat, hot_windows)
    heatmap=apply_threshold()
    heatmap = apply_threshold()
    labels = label()
    applied_image = draw_labeled_bboxes()
```



5. 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.) The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

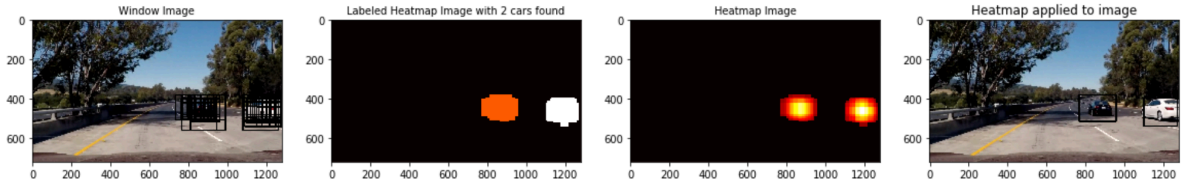
`Project_video_ouput.mp4 is the output video.`

6.Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes. A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

I make the false positive checks by introducing below several functions to group a pipeline.

The key is to extract sampling area from the original image and use the SVM classifier to screen out proper images including car features. Generally a car will be sampled several times in the same area with different sized blocks. Those overlapping blocks creates a heatmap that highlights the target we're looking for.

But the false positive case acts the opposite, they appears occasionally and may not locates in its adjacent frames. And would have no lots of blocks clustered together, so we add a numerical weights to a specific area if we identify a positive, and we count for the overall weight, if the weight pass some threshold we defined, we deem it as a real car, or else we will obsolete the findings.



Discussion

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? Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

My pipeline was not fully turned, due to frustrated run time. If I have a powerful hardware, I will try more parameter combinations, including each channels for each color space, hog transformation setups, or even find a fourth method to extract image features like boundary detection,

Another approach is to extract the correlations info from adjacent images by using of history labels. To filter out the false positives more intuitively.

In my understanding, this computer vision approach can give decisions on Yes/No detections, whereas not flexible enough to predict “what” is the object, and deep learning approach can do it with a much more flexible way(for sure it’s still a black box decision maker).

Updated Writeup of 2nd submit.

==========

Due to missing below code block. I refined my algorithm to have a 2nd try.

A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

in this case I introduced deque mothod to record historical heatmap info. To make it more robust and smooth. And it works.

```
# #set number of history frames used for further classifier.

# history_frames = 10

# history = deque(maxlen=history_frames) # history_frames is number of his
tory frames stored

def pipeline(img):

    ...

    heat = add_heat(heat, box_list)

    history.append(heat)

    heatmap = apply_threshold(sum(history)/n_frames, heat_threshold)

    ...
```

Meanwhile I'm not satisfied with my former performance, which processing lead time can go up to 5 hours, it's not reasonable enough in practice.

So I did lots of tests and create a log to record all those results.

1. Investiage on hog channels in different color channels. Whereas left other para. unchanged.

   Spatial features(length at 3072) and Hog channel features(lengh at 1764) counts the majority mass of the combined features(length at 4932),

   for HOG features, applying a single channel will save loads of time instead of all channels.

I found the light channel generally have a better performance on accuracy. So I stick to V channel in HSV.

```
test log:  hog_features = True, hist_features = False, spatial_features =
False



YCrCb Color space. Validation Accuracy of SVC =  0.9583;Test Accuracy of S
VC =  0.9055;9.56 Seconds to train SVC...

YUV Color space.   Validation Accuracy of SVC =  0.9541;Test Accuracy of S
VC =  0.8982;9.05 Seconds to train SVC...

HSV Color space.   Validation Accuracy of SVC =  0.9657;Test Accuracy of S
VC =  0.9083;8.97 Seconds to train SVC...

HLS Color space.   Validation Accuracy of SVC =  0.9609;Test Accuracy of S
VC =  0.9111;9.48 Seconds to train SVC...

LUV Color space.   Validation Accuracy of SVC =  0.9792;Test Accuracy of S
VC =  0.9348;11.27 Seconds to train SVC...

GRAY Color space.  Validation Accuracy of SVC =  0.9823;Test Accuracy of S
VC =  0.9753;4.02 Seconds to train SVC...



test log(Color space = HLS):



hog channels = 2: Validation Accuracy of SVC =  0.9105; Test Accuracy of S
VC =  0.8971; 7.56 Seconds to train SVC...

hog channels = 1: Validation Accuracy of SVC =  0.984;  Test Accuracy of S
VC =  0.9775; 3.91 Seconds to train SVC...

hog channels = 0: Validation Accuracy of SVC =  0.8829; Test Accuracy of S
VC =  0.8166; 6.81 Seconds to train SVC...



test log(Color space = HSV):



hog channels = 2: Validation Accuracy of SVC =  0.9842;Test Accuracy of SV
C =  0.9781; 3.5 Seconds to train SVC...

hog channels = 1: Validation Accuracy of SVC =  0.9001;Test Accuracy of SV
C =  0.8937; 6.58 Seconds to train SVC...
```

```
hot channels = 0: Validation Accuracy of SVC =  0.8798;Test Accuracy of SV
C =  0.8183; 6.73 Seconds to train SVC...



test log(Color space = LUV):



hog channels = 2: Validation Accuracy of SVC =  0.8925; Test Accuracy of S
VC =  0.8487; 6.79 Seconds to train SVC...

hog channels = 1: Validation Accuracy of SVC =  0.8902; Test Accuracy of S
VC =  0.8335; 8.73 Seconds to train SVC...

hog channels = 0: Validation Accuracy of SVC =  0.9817; Test Accuracy of S
VC =  0.9747; 3.95 Seconds to train SVC...



test log(Color space = YUV):



hog channels = 2: Validation Accuracy of SVC =  0.8458; Test Accuracy of S
VC =  0.8093; 5.36 Seconds to train SVC...

hog channels = 1: Validation Accuracy of SVC =  0.8562; Test Accuracy of S
VC =  0.8043; 6.54 Seconds to train SVC...

hog channels = 0: Validation Accuracy of SVC =  0.9831; Test Accuracy of S
VC =  0.9719; 3.81 Seconds to train SVC...



test log(Color space = YCrCb):



hog channels = 2: Validation Accuracy of SVC =  0.8559; Test Accuracy of S
VC =  0.8121; 5.24 Seconds to train SVC...

hog channels = 1: Validation Accuracy of SVC =  0.8393; Test Accuracy of S
VC =  0.7897; 7.32 Seconds to train SVC...

hog channels = 0: Validation Accuracy of SVC =  0.9823; Test Accuracy of S
VC =  0.9753; 3.98 Seconds to train SVC...
```

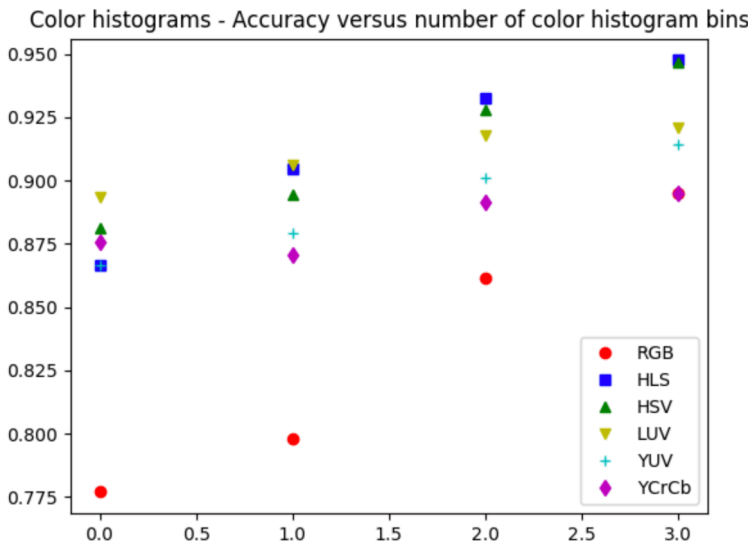
2. I randomly try other parameters as I see there are a lot to tune, so I walk around randomly and for each succeeding step I change only 1 parameters in order to find a potential correlation quickly.

I found even the accuracy is good, but the visional result may be not.
And YCrCb performed best for actual images.
So I changed my strategy. Stick to YCrCb color space.

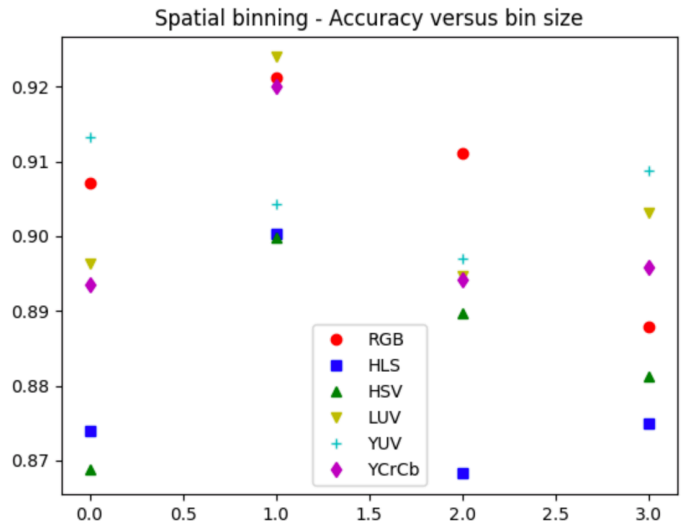
```
Test log:

hog_features = True, hist_features = True, spatial_features = True

color_space = HSV; hog channel = 2;Validation Accuracy of SVC =  0.9825;Test Accuracy of SVC =  0.9584

color_space = GRAY; Validation Accuracy of SVC =  0.9766;Test Accuracy of SVC =  0.9415;

color_sapce = HSV, hog channel = 'ALL'; Val Accracy = 0.984, Test Accu = 0.964

color_space = YCrCb, hog channel = 'ALL'; VA = 0.982; TA = 0.964

color_space = YCrCb, hot channel = 1, VA = 0.9665; TA = 0.946

color_space = YCrCb, hot_channel = 0; pixel_per_cell = 16, VA = 0.9769; TA = 0.9438

color_space = HSV, hot_channel = 2; pixel_per_cell = 16, VA = 0.9809; TA = 0.9629

color_space = LUV for HOG and Spatial, HLS for Color_hist, 'ALL'; VA = 0.9823, TA = 0.955

color_space = YCrCb for HOG and Spatial, HLS for Color_hist, 'ALL'; VA = 0.98; TA= 0.9567
```

later I found a priori knowledge, in below I posted:

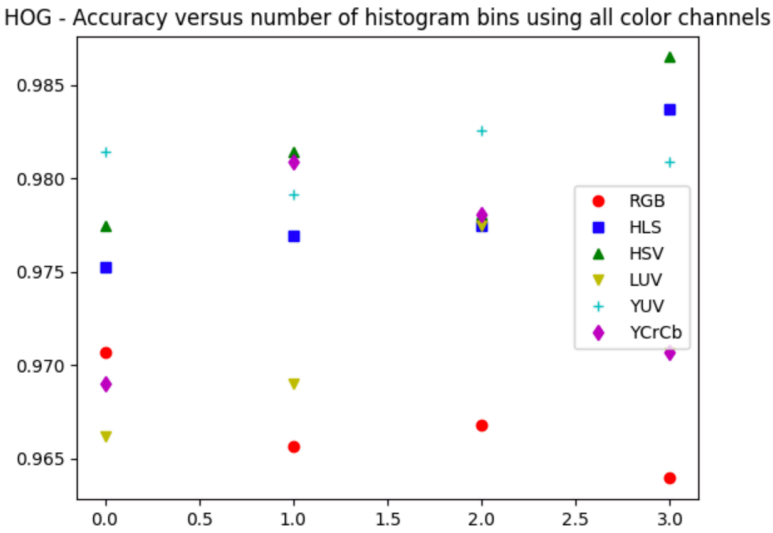
Color histogram features for bin sizes: 8, 16, 32, 64 (on x-axis)



Spatial binning for sizes: (8, 8), (16, 16), (32, 32) & (64, 64) (on x-axis)



HOG features for orientation histogram bins: 7, 8, 9, 10 (on x-axis)



I noticed several things:

- In general 8 orientation in HOG outperformed 9 orients with less data.
- Size (16,16) outperformed (32,32) in spatial spinning with less data.
- YCrCb color space outperformed in almost all the graph except for the color_hist.
- HLS performed best in color_hist

Base on above conclusion, I set all para. As below:

```
color_space = 'YCrCb' # Can be GRAY, RGB, HSV, LUV, HLS, YUV, YCrCb

orient = 8  # HOG orientations

pix_per_cell = 16 # HOG pixels per cell

cell_per_block = 2 # HOG cells per block

hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"

spatial_size = (16, 16) # Spatial binning dimensions

hist_bins = 32   # Number of histogram bins

spatial_feat = True # Spatial features on or off

hist_feat = True # Histogram features on or off

hist_range = (0,256)

hog_feat = True # HOG features on or of
```

I set hog_features and spatial_features in YCrCb color space.
And set Color_hist_features in HLS color space.

In output video I see no difference in identifying vehicles. So I confirmed this combination.

Further more I even tried spatial_size at (8,8), output video is also fine.

Then I have the confidence to introduce all the channels in YCrCb color space.

```
spatial_size = (8, 8) # Spatial binning dimensions
```

afterall, features length at 1152 achieved. Whereas in the beginning it is 4932.

Video Processing cycle time come to around 2s/frame(in the beginning is 10s-14s/frame)