Histogram of Oriented Gradients (HOG)

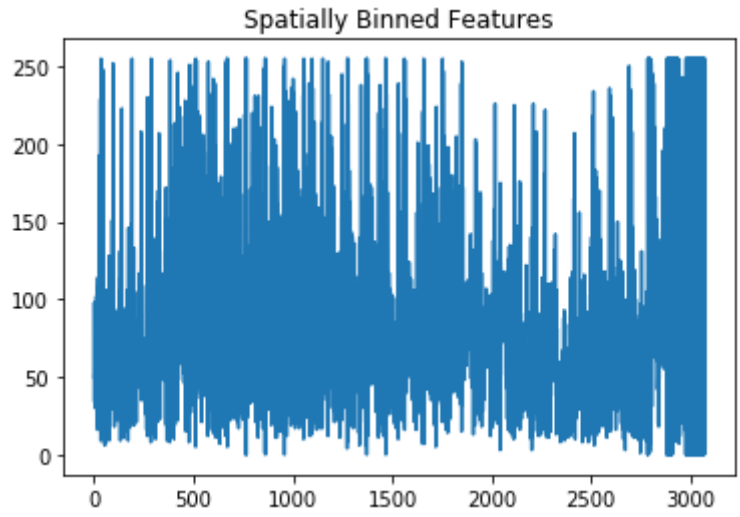
Criteria	Meets Specifications
Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.	Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG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

In order to get the best result, I used color and shape detection methods.

I defined 3 functions (in cell 2 of jupyter notebook):

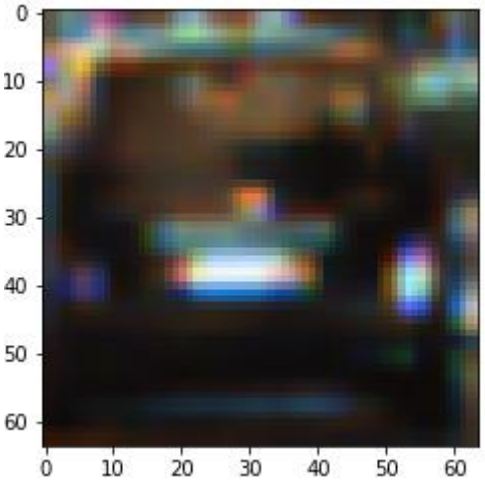
- bin_spatial: separate channels of image, convert image to size of 32x32, then combine them into a single feature vector

an example of spatial binning of an image converted to HSV color space (cell 7)

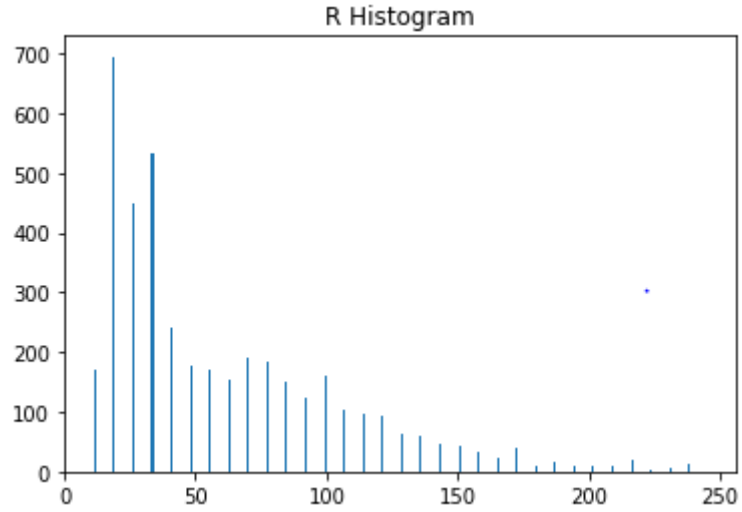


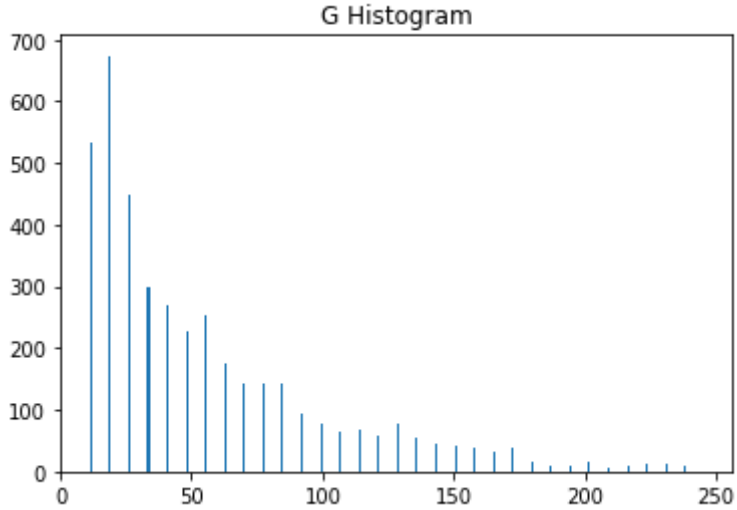
- color_hist: converted image is separated by channel and numpy histogram is applied. Number of bins I used is 32. an example of color histogram on an image (cell 6):

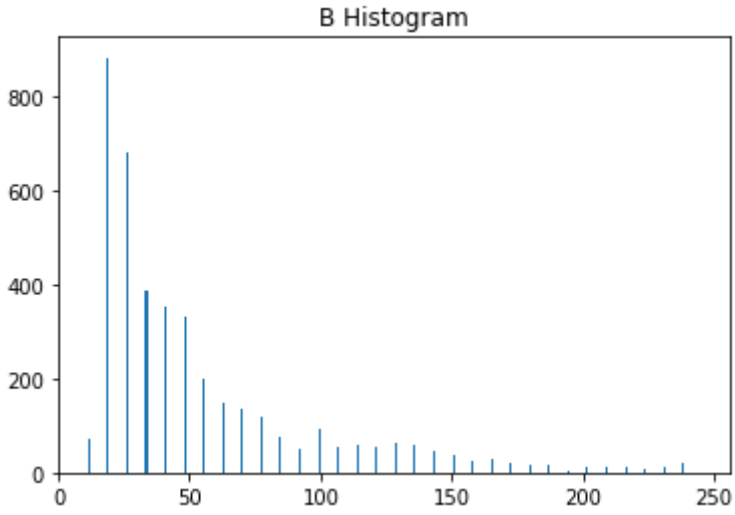
Image:



RGB Histogram:







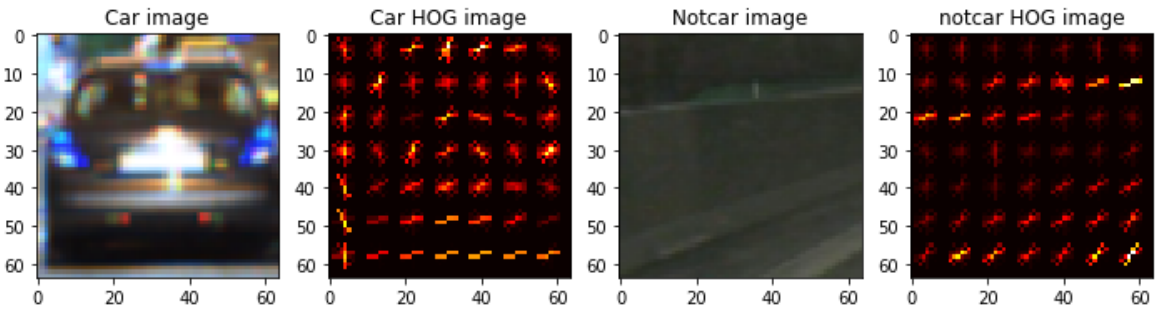
- get_hog_features: in order to get the HOG image I used “HOG” function from skimage.feature. The main input parameters are:
    - image, converted to color space YCrCb
    - orientation: 9 (number of orientation bins, the gradient information is split up into)
    - pixels per cell: 8 (cell size over which gradient histogram is computed. vertically, horizontally the same number of pixels (8x8))
    - cells per block: 2 (number of cells over which normalization is carried out, horizontally, vertically (2x2))
    - visualize: True or False, depends on if we need HOG image as return object from the function
    - feature vector: True or False to generate feature vector

To combine all the methods on images I created 2 functions in cell 2 of jupyter notebook:

- single_img_features: to extract HOG features of a single image
- extract_features: to extract HOG features of training dataset

The color space I used for the detection and training is YCrCb.

an example of HOG image of a car and a not car image from the training dataset (cell 4):

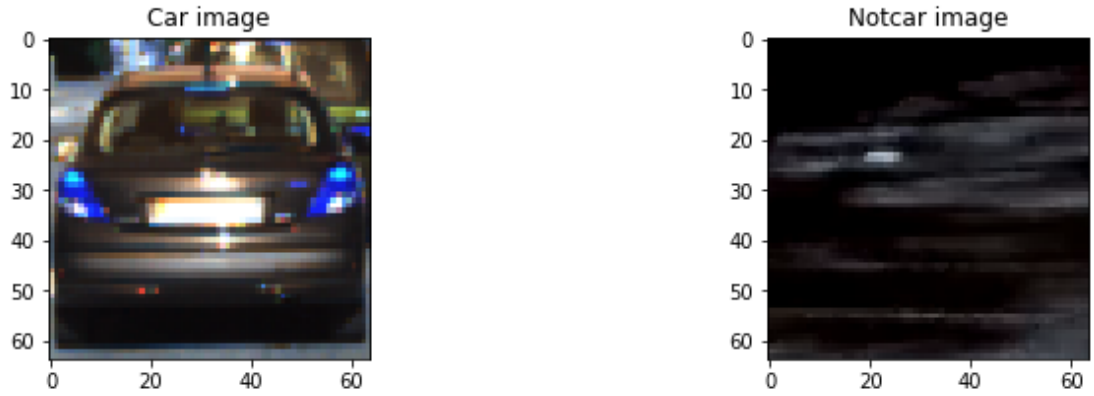


looking at the HOG images the edges of the car and certain parts like blue rear lights or the edges of the window of the car can be detected.

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 used function “extract_features” to extract HOG features of car and not car images to train my model. Some info about size of the training set and an example for a car and a not car image:

```
Function returned a count of 8792  cars and 8968  non-cars
of size:  (64, 64, 3)  and data type: uint8
```



I used the above mentioned parameters (like orientation, pixels per cell etc.) to train my model.

Before training the model I normalized my data (combined array of feature vectors of car and not car images) with “StandardScalar()” (cell 8).

I created a vector with the size of the complete data set. Car images got label value 1, not car image 0.

I split up the whole dataset into train and test dataset by using “train_test_split”. the size of the test dataset is 10% of the whole dataset. I made sure that the dataset is shuffled before splitting it up to avoid any ordering affects.

The classifier I implemented for the training is the Linear vector classifier of SVM.

The result of the training was 99.27% accuracy.

```
158.48 Seconds to extract features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 8460
106.37 Seconds to train SVC...
Test Accuracy of SVC =  0.9927
```

Sliding Window Search

Criteria	Meets Specifications
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.

To detect cars in an image we need to run classifier on sub-regions of an image.

I used sliding window technique to step across an image in a grid pattern and extract the same feature I used for training, then classified the extracted/converted window image to get a prediction.

To do that I defined:

- the region of interest where cars are supposed to show up (like we are not interested in the upper half part of the image where the sky and trees are which can result a false positive). This part is: y(400-720), x(0-1280)
- the size of the sub-region on the image (window size 96x96). I chose this size to better detect cars with bigger size, closer to the camera
- overlapping rate: 0.5, meaning how much windows should overlap while stepping across the image. 0.5 overlapping = moving the window horizontally and vertically by 48 pixels

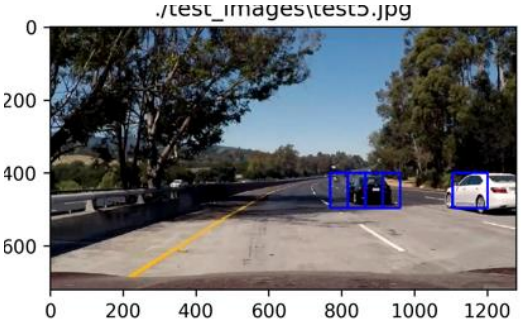
I defined function “slide_window” (cell 2) to determine the x,y coordinates of the window.

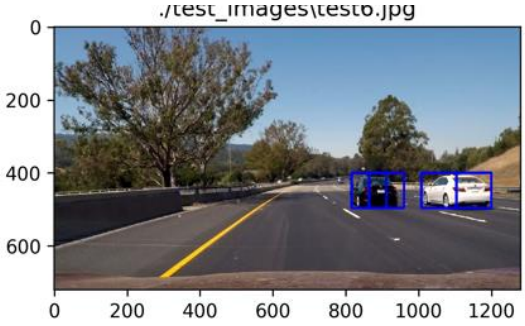
Second step to detect cars on the image is looking at the sub-regions of the image convert that part to a HOG image and try to predict if it is a car. I combined that functionality in function “search_window” (cell 2).

3rd step was drawing boxes around positive predictions.

Car detection logic can be found in cell 9.

Result of detection on example images:





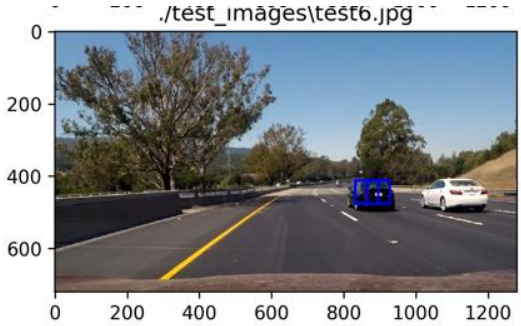
Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?	Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things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

I have tried various parameters for classification to detect cars on images and I found the above mentioned values to give the best result.

One important key is the region of interest to avoid false positive.

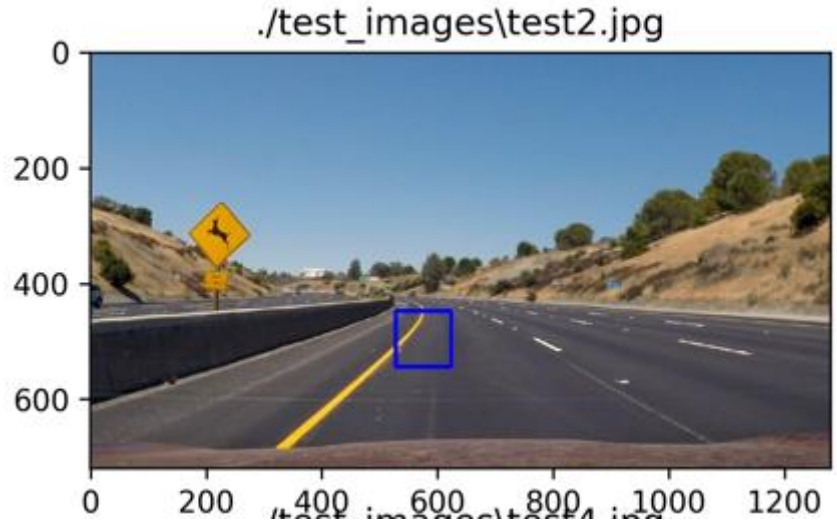
Second key point was the size of the searching window. Too small or high values can result incorrect detections

window size 64x64 did not detect all the cars on the image:



From color transformation point of view I tried to convert the images to different color spaces (like HSV etc.). YCrCb gave the best result so I stick to that one.

False detection with HSV color space (which was not the case with YCrCb):



same image with YCrCb color space:

detection to convert image to HLS:

same image with YCrCb color space:

Video Implementation

Criteria	Meets Specifications
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.

Link to video:

https://github.com/martonistvan/CarND-Vehicle-Detection-master.git - **project_output.mp4**

I implemented a new function ("find_cars", cell 2) to combine all the previously described detection steps, which are:

1, color convert of the whole image

2, scaling of converted image to change resolution by the factor of scale. I set it to 1.5 with windows size 96

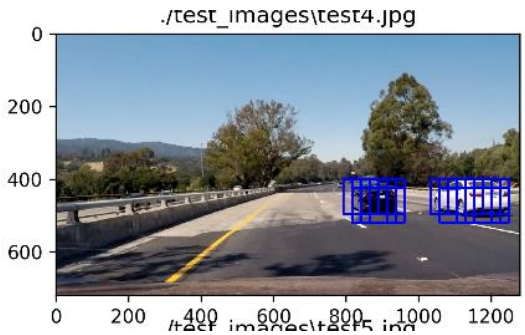
3, calculate the number of windows horizontally and vertically for detection based on parameters like pixels per cell, cell per block etc.

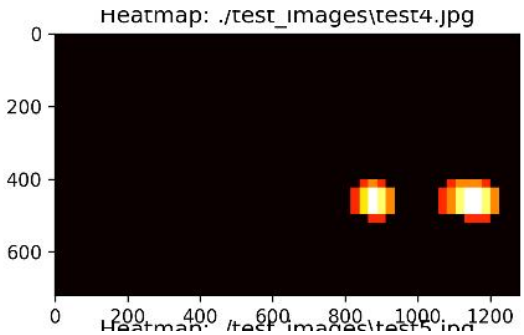
4, compute HOG for the entire image

5, for each search window I:

- generate feature vector (spatial binning, color histogram, HOG)
- run prediction. I used the same classifier as I used for training (Linear SVC)
- in case of successful prediction drawing box

Function returns image with detected cars and heatmap image (cell 10-11).





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 used heat-map in order to combine overlapping detections and ignore false positives (cell 10-11).

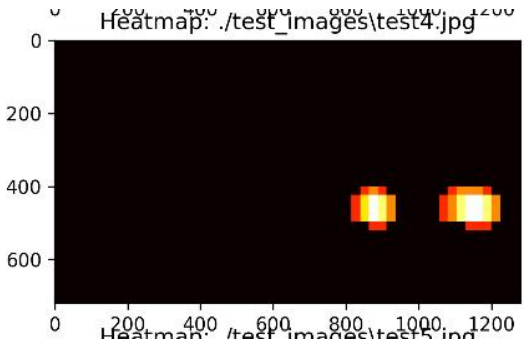
I generate heat-map with function “find_cars” with positive detections reported by the classifier. The “hot” parts of the image are where the cars are.

I implemented “apply_threshold” function to apply threshold to remove false positives.

To figure out how many cars were detected I used “label()” from “scipy.ndimage.measurements” on heatmap image. The function output is a 2-tuple. First item is the number of car detected, the second element is the labeled picture.

I took the output of “label()” to put bounding boxes around the labelled regions to overcome overlapping detections. For that I implemented a new function: draw_labeled_bboxes()

Result of labeling and putting bounding boxes on the image:



Discussion

Criteria	Meets Specifications
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.

It took time to find the best parameters for training and prediction. Which are the best values for orientation, pixs per cells, overlapping, window size etc.

Which color transformation to use, is also important, as by applying the proper one can reduce false positive detections.

There is still some room for improvement. Like there is a car that is not detected which is further from the camera. If I apply a low threshold they are detected but at the same time it results false positive as well.

Another possible improvement I need to work on when car is detected but the box drawn around it is smaller than the size of the car.

It’s been a very good and interesting experience playing around with all those parameters how they affect the training and classification.