Vehicle Detection and Tracking

Write up of work done for Udacity CarND Vehicle Detection and Tracking Project 5, by Mani Srinivasan

---

Project Objectives

Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labelled training set of images and train a classifier Linear SVM classifier
- Optionally, we could apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector. For those first two steps, We should also normalize the features and randomize a selection for training and testing.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the 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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Write-up / README

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

You're reading it!

Note: The code for the entire project was developed in Jupyter and the images and videos are embedded as part of the notebook *'Vehicle Detection and Tracking-P5.ipynb'*. Each step of the code is marked up for easy understanding of the flow, and the pipeline with images to visualize the intermediate results. I used the example code given in the lecture and added my own to complete the work.

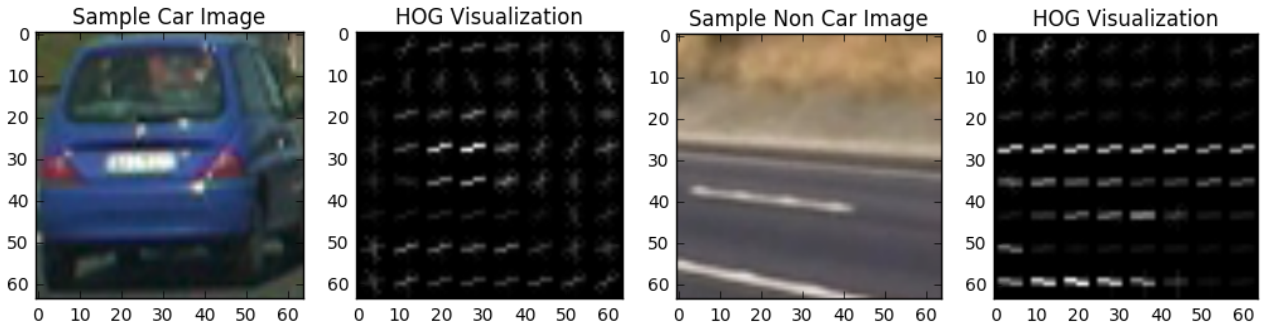
Pl. note that the images in this document could be different from the one on the Jupyter notebook, as every run selects a random set of images.

Histogram of Oriented Gradients (HOG)

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

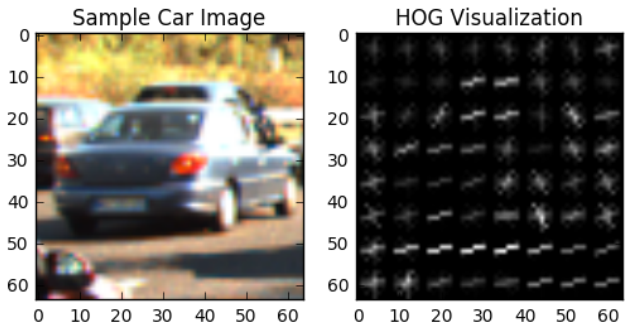
The code for this step is contained in the first code cell of the IPython notebook ("Vehicle Detection and Tracking-P5.ipynb'), under the marked-up section, ***Hog Feature Extraction***

I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:

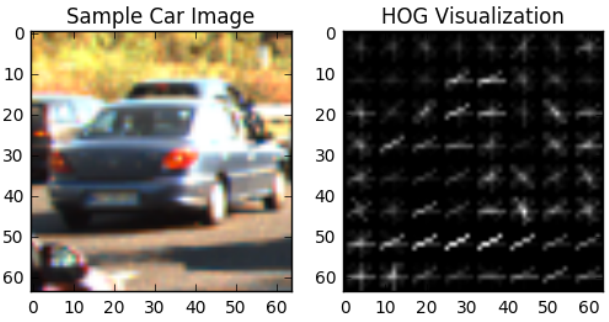


Orientation = 9, Hog Channel = 'ALL'

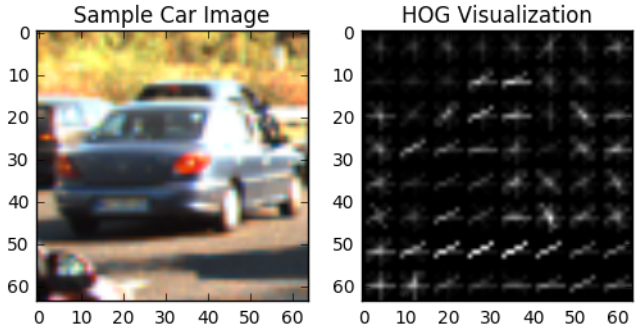
I also tried different vales for Orientation and Hog Channel values.



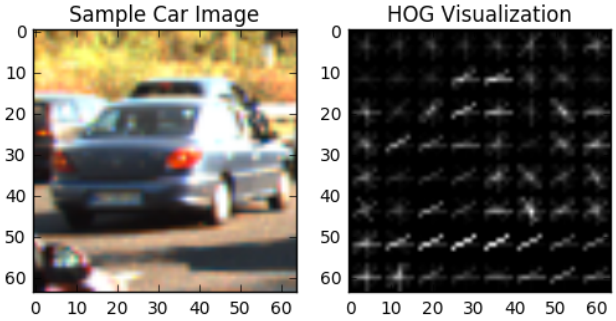
Orientation = 8, Hog Channel = 0



Orientation = 8, Hog Channel = 1

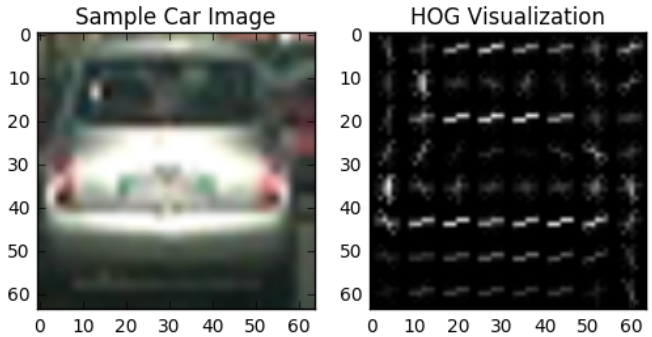


Orientation = 8, Hog Channel = 2



Orientation = 8, Hog Channel = ‘ALL’

I then explored different color spaces (RGB, YUV, YCrCb). Here is an example using the YCrCb color space and HOG parameters, of orientations = 9, pixels_per_cell = (8, 8) and cells_per_block=(2, 2). This are coded in the section marked-up as ***Experiments with Multiple Values for Hog Channel and Orientation*** in the Jupyter Notebook.



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

I tried various combinations of parameters for orientation and hog channel. There was not much of a difference between the various parameters used for HOG from the visual images. And so randomly chose orientation 9 and Hog Channel 'All'. In terms of color spaces, the YCrCb color channel gave the best training and prediction accuracy (> 99%) which showed in the final results as well. The following were the final parameters used.

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).

I trained a linear SVM using bin_spacial, color_hist, and HOG features. I also calibrated the classifier to enable the probability output. I used a normalized training data set and selected 20% as validation data after shuffling. These are contained in code sections marked-up as ***Experiments with Multiple Values for Hog Channel and Orientation*** and ***Train the SVM Classifier***.

```
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)
# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
# Split up data into randomized training and test sets
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=0.2, random_state=rand_state)


print('Using spatial binning of:',spatial_size[0],'and', hist_bins,'histogram bins')
print('Using:', orient, 'orientations', pix_per_cell, 'pixels per cell and', cell_per_block, 'cells per block.')
print('Feature vector length:', len(X_train[0]))
```

```
Using spatial binning of: 32 and 32 histogram bins
Using: 8 orientations 8 pixels per cell and 2 cells per block.
Feature vector length: 7872
```

Train the SVM Classifier

```
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t1 = time.time()
print("Training SVC Classifier...")
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2 - t1, 2), 'seconds to train SVC.')
```

```
Training SVC Classifier...
31.5 seconds to train SVC.
```

The SVM trained very well with an accuracy of 99.13% The code snippet for this is given below, and is under the marked-up section, ***Check the classifier predication accuracy*.**

Check the classifier predication accuracy

```
t1=time.time()
n_predict = 10
print('Classifier Prediction : ', svc.predict(X_test[0:n_predict]))
print('For ',n_predict, 'labels: ', y_test[0:n_predict])
t2 = time.time()
print(round(t2-t1, 5), 'seconds to predict', n_predict,'labels with SVC')
```

```
Classifier Prediction :  [ 1.  1.  1.  0.  1.  0.  0.  1.  1.  1.]
For  10 labels:  [ 1.  1.  1.  0.  1.  0.  0.  1.  1.  1.]
0.02952 seconds to predict 10 labels with SVC
```

```
t1=time.time()
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
t2 = time.time()
print(round(t2-t1, 5), 'seconds to compute the test accuracy.')
```

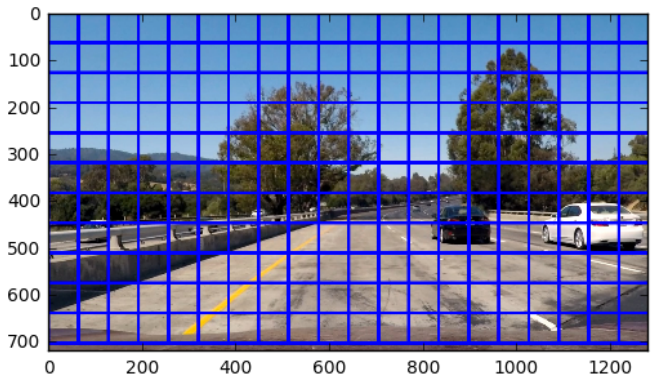
```
Test Accuracy of SVC =  0.9913
0.79592 seconds to compute the test accuracy.
```

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?

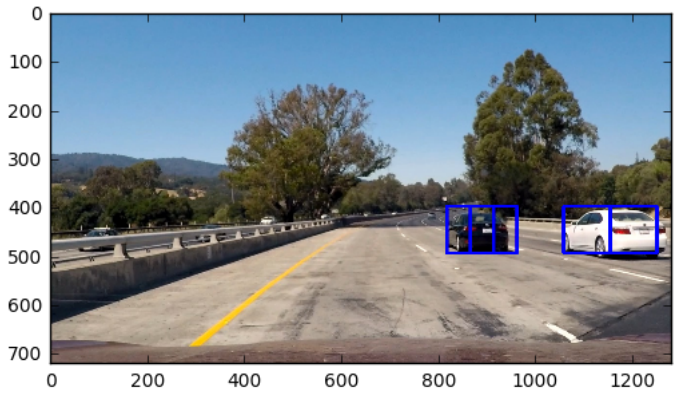
I used the code mostly from the course material for implementing the sliding window and drawing functions. The various functions are given in the section identified under the section marked up as ***Functions for Sliding window and car detection functions defined here*** in the Jupyter Notebook.

The sliding window was checked with a window size of 128x 128 and overlap of 0.5. Here is a representative image (using test1.jpg)



I used the Hog sub sampling window search to implement a more efficient method for doing the sliding window approach. This one allows us to extract the Hog features once. The find_cars only has to extract hog features once and then can be sub-sampled to get all of its overlaying windows.

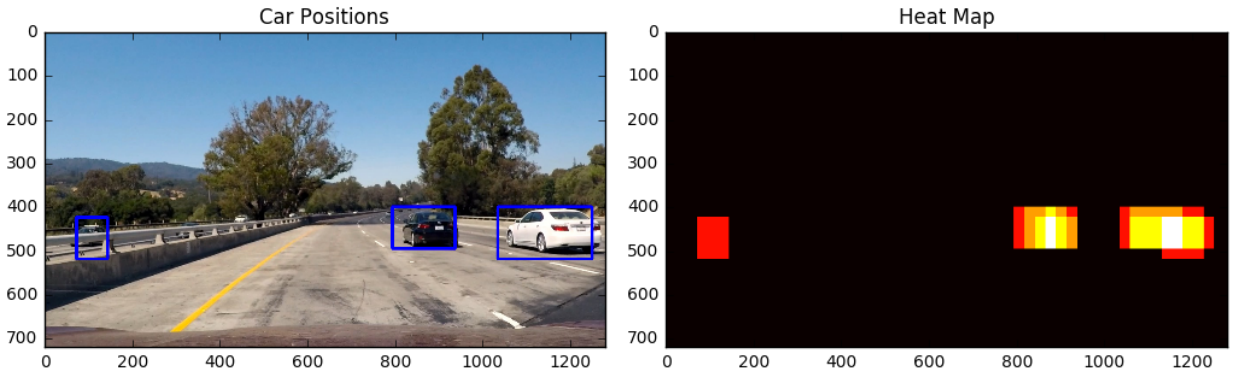
Sample image by using the find-cars is given below:



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?

I tried multiple color spaces, RGB, YUV, YCrCb. RGB tracking was way off. Ultimately, I settled for YCrCb space, which gave the best detection and tracking performance.

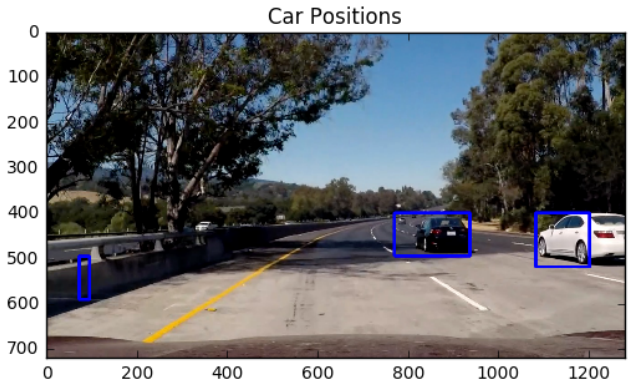
I also used the ‘add heat’ and ‘apply threshold’ functions to remove false positives and overlapping detections. Here are some sample images with the corresponding heat maps.



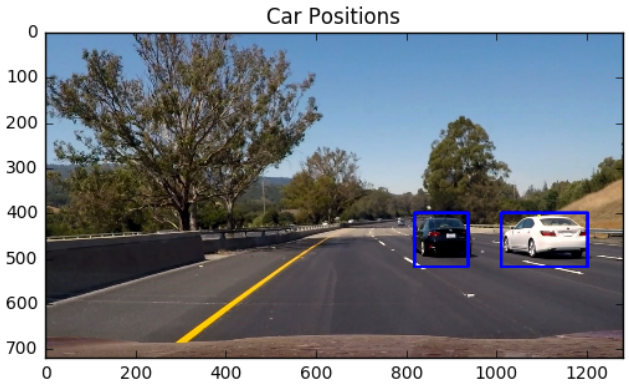
Final output after running the pipeline on some test samples.

Test Image 1

Test Image 3



Test Image 5



Test Image 6

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 the link https://youtu.be/l3cBZL0Tzho to my video output, *project_video_output.mp4,* which is also part of the Jupyter notebook.

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?

I had issues in getting the pipeline working due to incompatible normalization of colors (mostly due to my coding issues) with different libraries. In addition, RGB space was not able to create boxes which were contained only for the object of interest (cars) and finally the use of YCrCb gave the best results. There are still a few false positives but bearable.

Altho' the final results were good, I guess there is a scope for improvement in terms of testing out additional parameters and color spaces for better results.