**Vehicle Detection Project**

The goals / steps of this project are the following:

* Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
* Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
* Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
* Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
* Run your 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.

###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 utils.py  ( in lines 9 through 27).

I started by reading in all the `vehicle` and `non-vehicle` images.  The images were extracted from the following links
vehicle : https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip
non-vehicle :https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip

Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

Figure 1
Car class Image

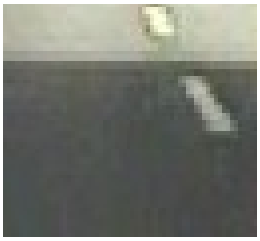


Figure 2
Non-car class Image

The code for extracting images for the corresponding folders in in vehcile_detection.py (in lines 224-242).

Total of 8792 images for vehicle class and 8968 from non-vehicle were obtained.

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).  I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `YUV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:
with hog performed on channel 0. The HOG output for Figure 1 is :

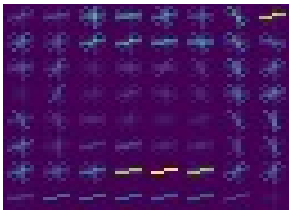


Figure 3
HOG output for Figure 1.

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

I tried various combinations of parameters as shown in the table and based my result based on best test set accuracy when trained on 500 samples : I only played with 2 color spaces namely HSV and YUV and HOG was performed on channel 0 in case of YUV and channel 1 in case of HSV.

| num of orient_bins | pixel_per_cell | Cells_per_block | Color Space | Accuracy |
|---|---|---|---|---|
| 9 | 8 | 2 | HSV | 0.955 |
| 12 | 8 | 2 | HSV | 0.895 |
| 9 | 16 | 2 | YUV | 0.955 |
| **9** | **8** | **2** | **YUV** | **0.973** |
| 9 | 4 | 4 | YUV | 0.965 |

####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 played a bit with my classifier different models generated are in model file.
1.
models/linear_svm.plk
I used sklearn.model_selection.GridCV function to find my parameter c in range 1-10 and kernel from rbf , Linear here are the stats for this classifier:
best params : c = 10 kernal = rbf
test_acc : 0.9692
training time : 2746.32 seconds.
This was with
orient  = 9
8 pixels_per_cell
2 cells_per_block
hist_bins = 32
spatial_size = 16
color space = hsv ,channel: s

2.
models/linear_svm1.plk
I used Linear SVC with parameters

```
color_space = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
```

```
hog_channel = 0 # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 16    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
y_start_stop = [None, None] # Min and max in y to search in slide_window()
```

the stats were
training time : 135.16 sec to extract features + 12.3 seconds to train scv(NOTE: this was done on a beefed up system hence these timings)
test accuracy: 0.9764

3.
models/linear_svm2.plk
I again used Linear.SCV() but with parameters:

```
# parametsrs2 for different feature extraction techniques
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 32  # 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 = (32, 32) # Spatial binning dimensions
hist_bins = 32    # Number of histogram bins
spatial_feat = False # Spatial features on or off
hist_feat = False # Histogram features on or off
hog_feat = True # HOG features on or off
y_start_stop = [None, None] # Min and max in y to search in slide_window()
```

This gave me accuracy of 98.03 which is slightly better but was not performing as well which i'm not sure why. The Training time was similar to the previous case.

The actual classifier implementation is in Vehicle_detection (in lines 257-316)

###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?

The sliding window operation is performed in vehicle_detection.py from lines 170-182 and actually searching is done in function search_windows in vehicle_detection .py from lines 98-127.

The the final size of window search was based upon last of experimentations.
The final results are as tabulated

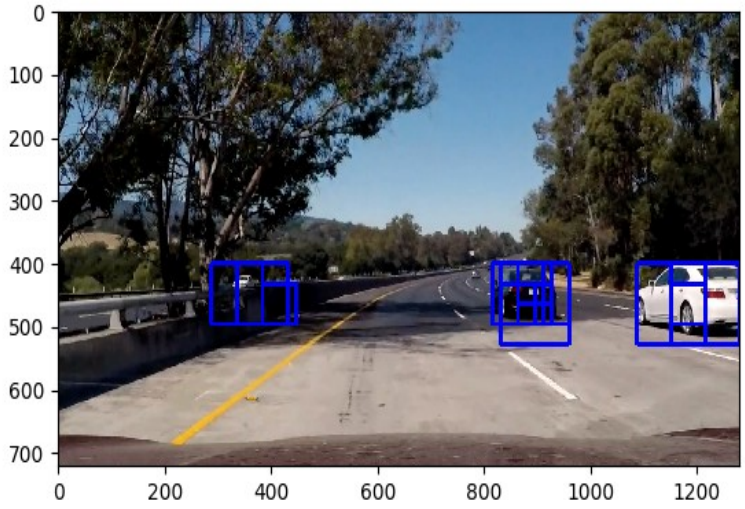
| window_size | Y start | Y stop | overlap |
|---|---|---|---|
| 64 | ceil(height*.555) | ceil(height*.694) | 0.5 |
| 96 | ceil(height*.555) | ceil(height*.694) | 0.5 |
| 128 | ceil(height*.555) | ceil(height*.802) | 0.5 |
| 192 | ceil(height*.555) | ceil(height*.923) | 0.5 |

* where height id the height of the image on which operation is performed.
Basically smaller windows were searched for possible match near the horizon and larger near to the car on which the camera was mounted.

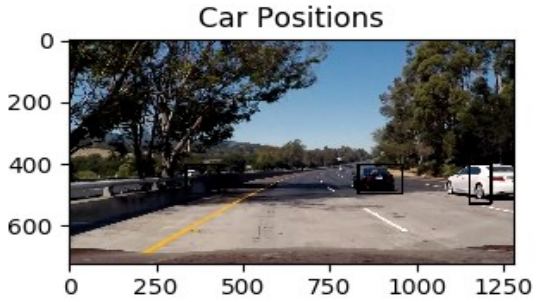
####2. Show some examples of test images to demonstrate how your pipeline is working.  What did you do to optimize the performance of your classifier?

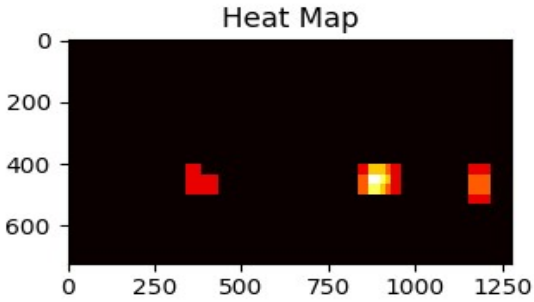
Not much optimization was done , its in future work for the project, to extract features and classify at the same time to reduce the computational cost and time. The pipeline for image in test_images/test5.jpg is demonstrated below:

original image




windows_of different scale




Heatmap and detection

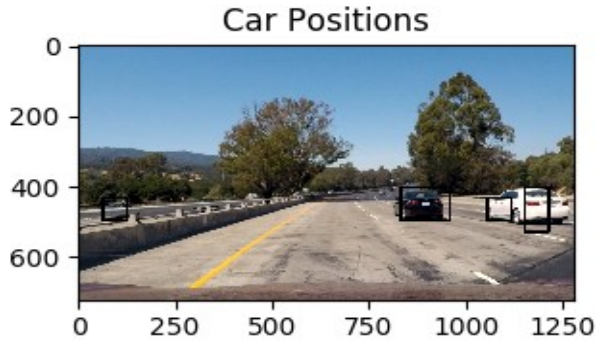
All the pipeline images are in output_images/ for further inspection. If you look carefully the detection can identify cars on opposite side of the road too.

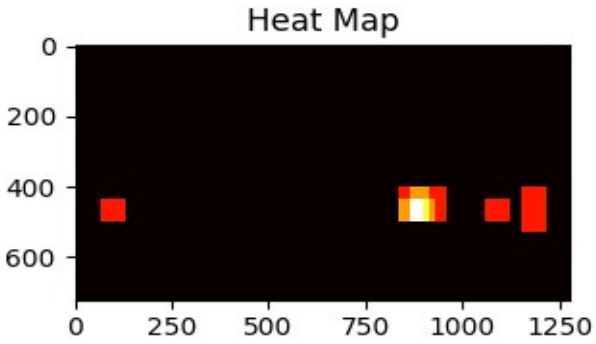
### Video Implementation

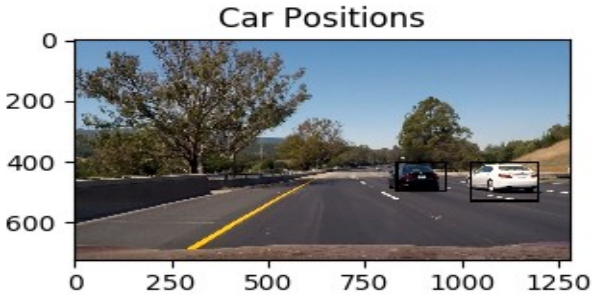
####1. Provide a link to your final video output.  Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)
The output video is called project_video_output.mp4.

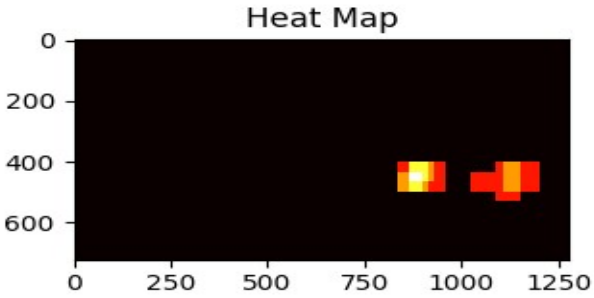
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of positive detections in each frame of the video.  From the positive detections I created a heat map and then thresholded that map to identify vehicle positions.  I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap.  I then assumed each blob corresponded to a vehicle.  I constructed bounding boxes to cover the area of each blob detected.  The code to do the same is in vehicle_detection (in lines 199-208) . I use a threshold value of 1.


Car Positions
0
200
400
600
0
250
500
750
1000
1250


Heat Map
0
200
400
600
0
250
500
750
1000
1250


Car Positions
0
200
400
600
0
250
500
750
1000
1250


Heat Map
0
200
400
600
0
250
500
750
1000
1250

---

###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?

Problems Faced and potential Pitfalls:
There are still some misclassification is there more robust feature are needed and have to find a way to reject outliers.
Training took a lot of time, was not able to experiment with different classifiers.


Future work:
1.optimize the pipeline
2. Use ensemble classifier scheme for feature section before feeding to svm a few classifiers that can be tried are Decision tree and Neural Networks.