Table of Contents

Vehicle Identification Project

1. Project Description

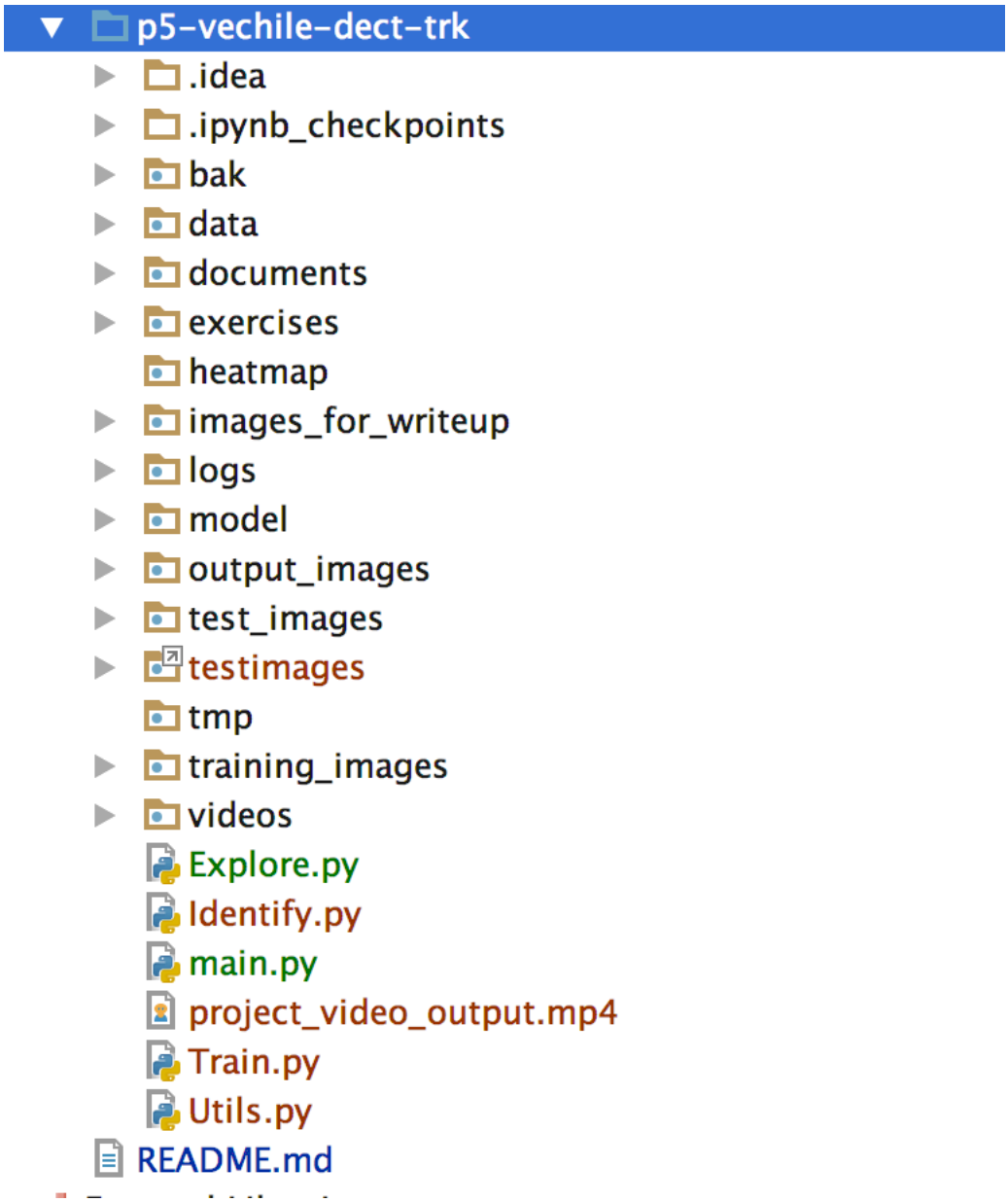
The goals of this project are the following:

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
2. Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
3. Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. Run your pipeline on a video stream and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
6. Estimate a bounding box for vehicles detected.

2. Getting Started

1. Clone the project and create directories `vehicles` and `non-vehicles`.
2. Download images from the links given below and put them into subfolders below `vehicles` and `non-vehicles`.
3. Explore.py splits the data into training, validation and test set and saves them in a pickle file.
4. Train.py trains an SVM to detect cars and non-cars. All classifier data is saved in a pickle file.
5. Identify.py implements a sliding window search for cars, including false positive filtering and applies the classifier to a video

Following is the directory structure of the project



- ./logs directory : contains the logs from the program. You will see the training and search for cars log info.
- Main.py, Explore.py, Identify.py and Utils.py are the key programs
- ./model directory : contains the trained model
- ./test_images : contains test images
- ./testimages : contains latest udacity test images
- ./videos : contains input video stream
- ./project_video_output.mp4 is the processed video stream

3. Data Exploration

3.1. Data Sources

1. Labeled images were taken from the GTI vehicle image database [GTI](http://www.gti.ssr.upm.es/data/Vehicle_database.html), the [KITTI](http://www.cvlibs.net/datasets/kitti/)
   vision benchmark suite, and examples extracted from the project video itself.
2. All images are 64x64 pixels.
3. A third [data set](https://github.com/udacity/self-driving-car/tree/master/annotations) released by Udacity was not used here. In total there are 8792 images of vehicles and 9666 images of non vehicles.
4. Thus the data is slightly unbalanced with about 10% more non vehicle images than vehicle images.
5. Images of the GTI data set are taken from video sequences which needed
   to be addressed in the separation into training and test set.
   Shown below is an example of each class (vehicle, non-vehicle) of the data set.

3.2. Data Exploration

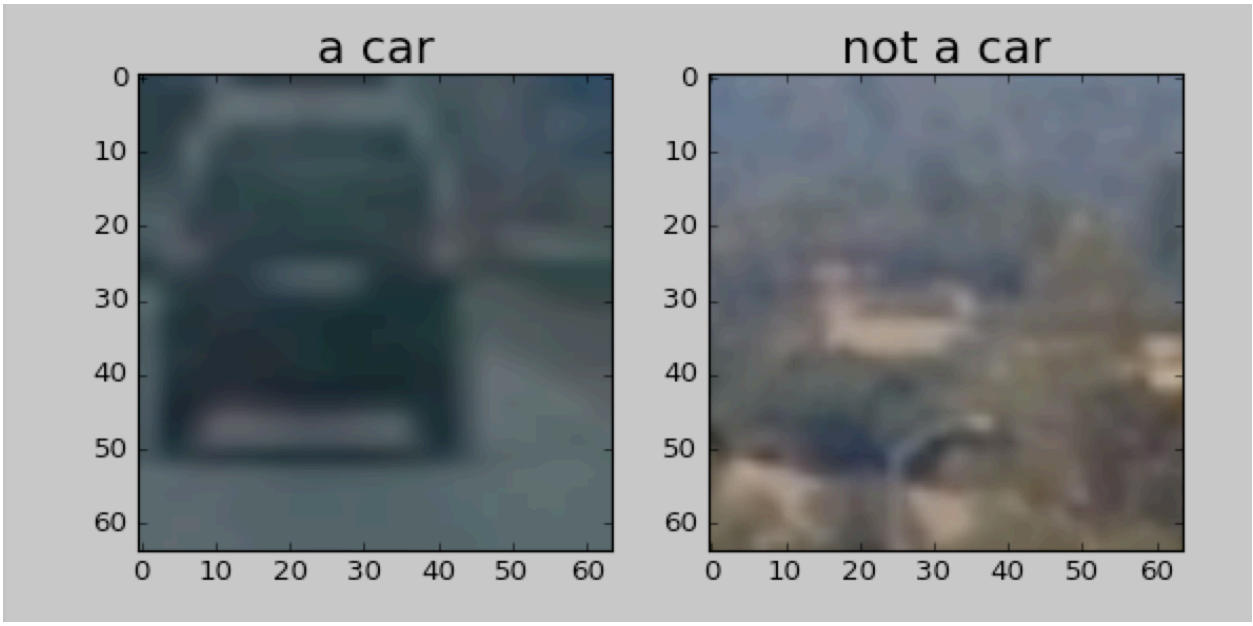
```
print('Number of samples in cars training set: ', len(cars_train))
print('Number of samples in notcars training set: ', len(notcars_train))

print('Number of samples in cars validation set: ', len(cars_val))
print('Number of samples in notcars validation set: ', len(notcars_val))

print('Number of samples in cars test set: ',len(cars_test))
print('Number of samples in notcars test set: ',len(notcars_test))
```

```
# Save the data for easy access
pickle_file = DATA_DIR + 'data.p'
print('Saving data to pickle file...')
try:
    with open(pickle_file, 'wb') as pfile:
        pickle.dump(
            {
                'cars_train': cars_train,
                'notcars_train': notcars_train,
                'cars_val': cars_val,
                'notcars_val': notcars_val,
                'cars_test': cars_test,
                'notcars_test': notcars_test
            },
            pfile, pickle.HIGHEST_PROTOCOL)
except Exception as e:
    print('Unable to save data to', pickle_file, ':', e)
    raise

print('Data cached in pickle file.')
```



4. Training and Modeling

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

The training set was divided as follows:

- the first 70% of any folder containing images was assigned to be the training set, the next 20% the validation set and the last 10% the test set.

- In the process of generating HOG features all training, validation and test images were normalized together and subsequently split again into training, test and validation set. Each set was shuffled individually.

- The code for this step is contained in Train.py : extractFeatures(). I explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).

```
51      def extractFeatures(self,imgs, color_space='RGB', spatial_size=(32, 32),
52                                          hist_bins=32, orient=9,
53                                          pix_per_cell=8, cell_per_block=2,
54                                          spatial_feat=True, hist_feat=True, hog_feat=True):
55
56          print ("Inside extractFeatures")
57          # Create a list to append feature vectors to
58          features = []
59          # Iterate through the list of images
60          for file in imgs:
61              file_features = []
62              # Read in each one by one
63              # orig: image = mpimg.imread(file)
64
65              image = scipy.misc.imread(file)
66
67              # apply color conversion if other than 'RGB'
68              if color_space != 'RGB':
69                  if color_space == 'HSV':
70                      feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
71                  elif color_space == 'LUV':
72                      feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
73                  elif color_space == 'HLS':
74                      feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
75                  elif color_space == 'YUV':
76                      feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
77                  elif color_space == 'YCrCb':
78                      feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
79              else:
80                  feature_image = np.copy(image)
81
82              if spatial_feat == True:
83                  spatial_features = bin_spatial(feature_image, size=spatial_size)
84                  file_features.append(spatial_features)
85              if hist_feat == True:
86                  # Apply color_hist()
87                  hist_features = color_hist(feature_image, nbins=hist_bins)
88                  file_features.append(hist_features)
89              if hog_feat == True:
```

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

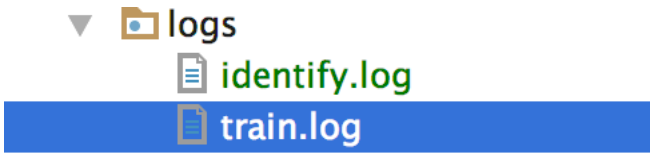
1. I went through a number of different combinations of color spaces and HOG parameters and trained a linear SVM using different combinations of HOG features extracted from the color channels.

2. I have captured the training log in ./logs/train.log. I used following params for training

```
CELLPERBLOCK_LIST   = [2, 4]
ORIENT_LIST         = [9, 12]
HISTBINS_LIST       = [32, 48]
```

Training with Cell per block 4. orient 12. hist_bins 48 yielded the best test accuracy. You can look at the accuracy of others params in the log file.

```
Training with Cell per block 4. orient 12. hist_bins 48
Inside extractFeatures
Inside extractFeatures
Using:  12 orientations 8 pixels per cell and  4 cells per block  48 hist_bins =>
feature vector length:  15312
Seconds to train SVC:  72.67
Train Accuracy :  1.0
Test  Accuracy:   0.9938
Prediction time :  0.0009100437164306641
Label  1 Prediction [ 1.]
Prob [[ 0.03029128  0.96970872]]
```



3. For HLS color space the L-channel appears to be most important, followed by the S channel. I discarded RGB color space, for its undesirable properties under changing light conditions. YUV and YCrCb also provided good results, but proved to be unstable when all channels were used.

4. There was relatively little variation in the final accuracy when running the SVM with some of the individual channels of HSV,HLS and LUV.

5. I finally settled with Cell per block 4. orient 12. hist_bins 48. Using larger values of than `orient=9` did not have a striking effect and only increased the feature vector. Similarly, using values larger than `cells_per_block=4` did not improve results, which is why these values were chosen.

####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).

1. I trained a linear SVM using all channels of images converted to HLS space. I included spatial features color features as well as all three HLS channels.
2. The final feature vector has a length of 1836, most of which are HOG features. For color binning patches of `spatial_size=(16,16)` were generated and color histograms were implemented using `hist_bins=48` used. After training on the training set this resulted in a validation and test accuracy of 98%.
3. The average time for a prediction (average over a hundred predictions) turned out to be about 3.3ms on an i7 processor, thus allowing a theoretical bandwidth of 300Hz. A realtime application would therefore only feasible if several parts of the image are examined in parallel in a similar time.

Using just the L channel reduced the feature vector to about a third, while test and validation accuracy dropped to about 94.5% each.

Unfortunately, the average time for a prediction remained about the same as before. The classifier used was `LinearSVC` taken from the `scikit-learn` package.

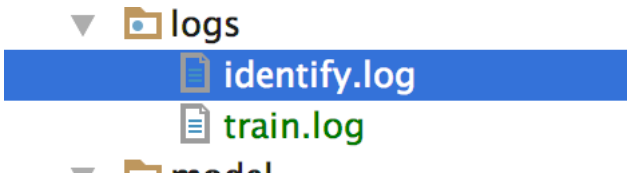
Despite the high accuracy there is a systematic error as can be seen from investigating the false positive detections. The false positives include frequently occurring features, such as side rails, line line markers etc. Shown below are some examples that illustrate this problem.

5. Sliding Window

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?

1. In the file Identify.py, I have segmented the image into 4 partially overlapping zones with different sliding window sizes to account for different distances.

2. The window sizes are 240,180,120 and 70 pixels for each zone. Within each zone adjacent windows have an ovelap of 75%, as illustrated below.




```
Process the video stream ...
full search ...
frame_number:  1  car_count:  0  search region: ( 1280 720 ), (  0 0 )
[MoviePy] >>>> Building video project_video_output.mp4
[MoviePy] Writing video project_video_output.mp4
  0%|          | 0/1261 [00:00<?, ?it/s]car_count:  0 perform_search:  False
search_count:  1
Skip this frame ... no grid search
frame_number:  2  car_count:  0  search region: ( 1280 720 ), (  0 0 )
car_count:  0 perform_search:  True search_count:  0
full search ...
frame_number:  3  car_count:  0  search region: ( 1280 720 ), (  0 0 )
  0%|          | 2/1261 [00:04<47:56,  2.28s/it]car_count:  0 perform_search:  False
```

```
search_count:  1
Skip this frame ... no grid search



frame_number:  1102  car_count:  2  search region: ( 814 384 ), (  1260 516 )
 87%|████████▋ | 1101/1261 [28:12<07:19,  2.75s/it]car_count:  2 perform_search:  True
search_count:  1
short search:
frame_number:  1103  car_count:  2  search region: ( 814 384 ), (  1268 516 )
 87%|████████▋ | 1102/1261 [28:15<07:13,  2.72s/it]full search ...
frame_number:  1104  car_count:  2  search region: ( 814 384 ), (  1268 516 )
 87%|████████▋ | 1103/1261 [28:20<08:43,  3.31s/it]car_count:  2 perform_search:  True
search_count:  1
short search:
frame_number:  1105  car_count:  2  search region: ( 814 384 ), (  1269 516 )
 88%|████████▊ | 1104/1261 [28:22<08:09,  3.12s/it]car_count:  2 perform_search:  True
search_count:  1
short search:
frame_number:  1106  car_count:  2  search region: ( 814 384 ), (  1272 516 )
 88%|████████▊ | 1105/1261 [28:25<07:42,  2.96s/it]car_count:  2 perform_search:  True
search_count:  1
short search:
frame_number:  1107  car_count:  2  search region: ( 814 384 ), (  1273 516 )
 88%|████████▊ | 1106/1261 [28:28<07:27,  2.88s/it]car_count:  2 perform_search:  True
search_count:  1
```

####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to try to minimize false positives and reliably detect cars?

1. The final classifier uses four scales and HOG features from all 3 channels of images in HLS space. The feature vector contains also spatially binned color and histograms of color features.
2. False positives occurred much more frequently for `pixels_per_cell=8` compared to `pixels_per_cell=16`.
3. Using this larger value also had the pleasant side effect of a smaller feature vector and sped up the evaluation. The remaining false positives were filtered out by using a heatmap approach as described below. Here are some typical examples of detections

False positives occur on the side of the road, but also simple lane lines get detected as cars from time to time. Cars driving in the opposite direction also get detected in so far as a significant portion is visible.

6. Pipeline

####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.)

Please find the final result of the video processing pipeline

./output /processed_project_video.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.

1. In the file Identify.py, I have implemented a logic to save the last `n` frames.

2. For every frame the (possibly empty) list of detected bounding boxes gets added to the beginning of the queue, while the oldest list of bounding boxes falls out.

3. This queue is then used in the processing of the video and always contains the bounding boxes of the last `n=20` frames. On these a threshold of 20 was applied, which suppresses also false positives from detected lane lines.

7. 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?

1. I started out with a linear SVM due to its fast evaluation. Nonlinear kernels such as `rbf` take not only longer to train, but also much longer to evaluate.
2. Using the linear SVM I obtained execution speeds of 3 FPS which is rather slow. However, there is ample room for improvement.
3. At the moment, the HOG features are computed for every single image which is inefficient. A way to improve speed would be to compute the HOG features only once

for the entire region of interest and then select the right feature vectors, when the image is slid across.

4. Some false positives still remain after heatmap filtering. This should be improvable by using more labeled data.
5. Another very interesting avenue would be to use a convolutional neural network, where it is easier to incorporate scale invariance.