Project Report

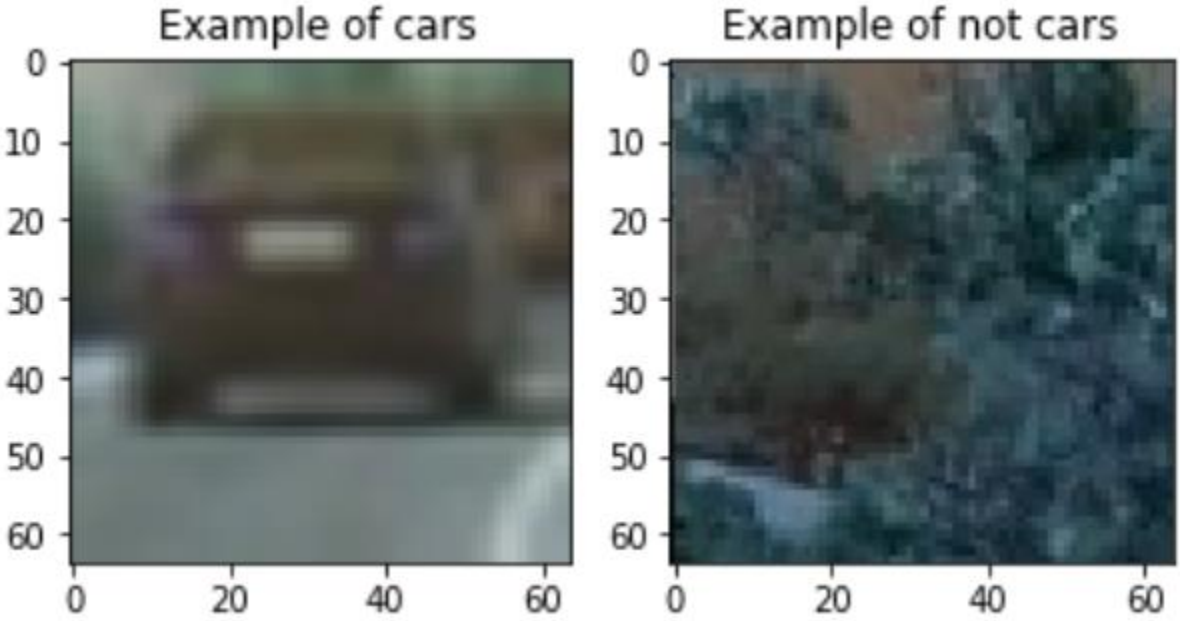
Vehicle Detection and Tracking

Name: Zeyu Tang

The goal is to write a software pipeline to identify vehicles in a video from a front-facing camera on a car. Here are my steps.

Prepare training data

From given folders '*vehicles*' and '*non-vehicles',* I import training images with their labels. Here is an example of image with cars and an example of image without cars.



Prepare feature vectors

Extract features

Based on multiple trials, I choose to use YCrCb color space, and combined features of color histogram, binned color features and HOGs. Here are details of these features.

Color space	YCrCb
Number of HOG orientations	9
Channels of HOG	CH3
HOG pixels per cell	8
HOG cells per block	2
Spatial binning dimensions	32 * 32

Number of histogram bins	16

When tuning parameters, my principle is to find values of parameters so as to make the classifier works well on all testing images in folder ‘test_images’ while to minimize the computing time.

For each image, extracted features are stored in a feature vector.

Preprocessing on feature vectors

Then all feature vectors, along with their labels are randomly divided into training set and testing set.

After that, all feature vectors in training set are normalized to 0 mean and small variance, and feature vectors in testing set are scaled with the same scaler factor.

Train a classifier

I choose to use Linear SVC as the classifier. Because the differences between features vectors of images with cars and images without cars, and the data set looks not big.

I feed all shuffled feature vectors of training set in to SVC, and get fitted model. Then I test the model on testing set. Here is the testing result.

```
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 4884
15.27 Seconds to train SVC...
Test Accuracy of SVC =  0.969
```

Build pipeline

Set sliding window search

To set sliding window search, I define several functions in the first cell of my codes.

slide_window() function is used to generate lists of windows positions, based on input images and other settings about windows.

draw_boxes() function is used to draw bounding boxes on input images according to input windows positions

single_img_features() is used to define a function to extract features from a single image window. And *search_windows()* is to search all windows and get windows with

positive positions. The parameters about extracting features are the same as above. The function is only used to visualize the process by generating images with all windows with positive detections. However, extracting HOG in each window require pretty much computing effect.

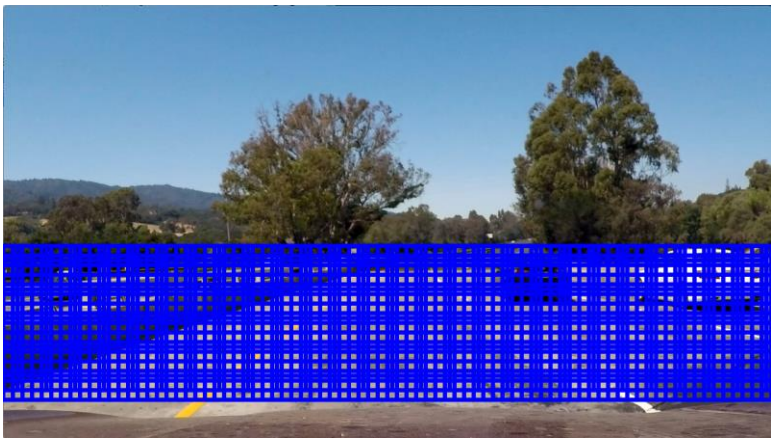
To reduce computing effect, I define another function *find_cars().* The function is used to get final image with hot windows on it. The function compute HOG only once in every input image. And each window just extract subarray from the HOG feature array.

By calling functions above several times with different window sizes, multiple scales of windows can be used to search on one image.

Here are some important parameters.

Window sizes, in pixels.	128, 96, 64
Overlapping	75% on both x and y directions
Range of searching area on y direction	(400,656)

Here are some examples of images during sliding windows. More images during the process can be found in folder '*test_images_output'*.



All sliding windows

All windows with positive detections

Heat map and thresholding

Based on hot windows, I add 1 heat to each hot window area on a blank image. Then a threshold is used to remove false positives. After that, a heat map, along with labels on each heat area, can be obtained. Finally, I draw boxes that can cover each heat area, which are considered to be positions of detected vehicles.

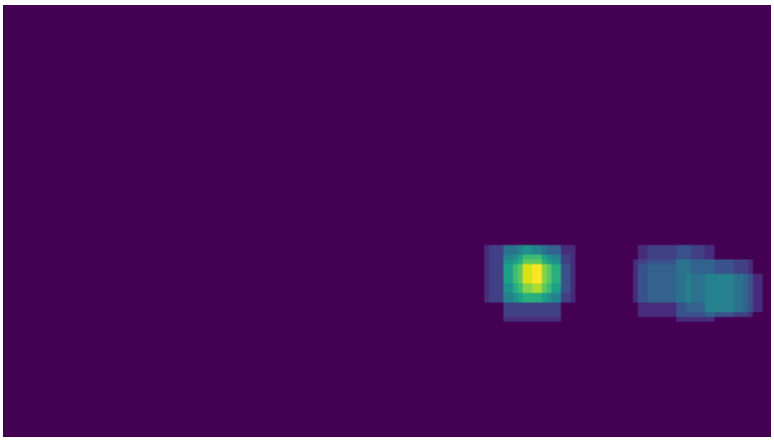
When dealing with videos, to filter out transient false positives, all heats are stored during the last 5 frame, and a threshold is used to the sum heat.

What is more, to filter out false positives with high heat values in small area, the minimum size of output boxes is limited

Here are the thresholds. All thresholds are proved to be working well on test images and the given videos.

Threshold on single image	1
Threshold on videos	7
Minimum size of boxes	30*30

Here are a example of heat map. More images during the process can be found in folder '*test_images_output'*.



Heat map of the same image as above

Result

The pipeline is used on test images. Here is an example of results. More images during the process can be found in folder '*test_images_output'*.

Final image of the same image as above

The pipeline is also used on 'test_video.mp4' and 'project_video.mp4'. Results can be found in folder 'test_videos_output'.

Conclusion and some discussion

The pipeline works well with given test images and videos.

However, there are still some issues.

Firstly, the pipeline is too time-consuming that it runs about 25 minutes on processing 'project_video.mp4' which is only 50 seconds. Although it is certainly to be helpful that

reduce features size such as reducing numbers of HOG originations and numbers of channels or reduce window numbers, the pipeline will show some misdetections during testing if I further reducing features. Actually, my first try on the same video with longer feature vectors took about 2 hours. My computer is of pretty high performance, so I do not think my pipeline is practicable when dealing with video stream in real-time. I will keep optimizing my pipeline in the following study.

Secondly, as I mentioned above, I already filter out those boxes that are too small that they are certainly false positives in the case of given videos. But in other scenarios, other vehicles may appear to be really small, and are easily to be considered as false positives. Those small false positives could a shadow with very dark color or shinny guardrails, which are similar to cars. However, there should be some right way to improve my pipeline and get more robust. I will continue focusing on my pipeline.

Appendix

There are 2 folders and three files in the ZIP archive.

The first folder ‘output_images’ consist of 4 subfolders, and stored all output during processing those given test images. Subfolder ‘all_windows’ includes all images with all sliding windows. Subfolder ‘hot_windows’ includes all images with all windows with positive detections. Subfolder ‘heat-maps’ includes visualized heat maps of all images. Subfolder ‘final_images’ includes all final results with boxing vehicles’ positions of all images.

The second folder ‘test_videos_output’ includes results of implementing my pipeline on both ‘test_video.mp4’ and ‘project_video.mp4’.

‘CarND-Vehicle-Detection-master-final.ipynb’ file is the code.

‘Report.html’ file is a record of the running result of the code.

‘writeup’ file is the current introduction.