Vehicle Detection and Tracking

Author: Aneeq Mahmood

Email: Aneeq.sdc@gmail.com

The goal of this work has been to perform vehicle detection and tracking on a road using support from openCV and state vector machines (SVMs)

The individual rubric points are addressed below.

Files Submitted:

Following files are submitted for this work:

project5.ipynb : Jupyter notebook containing the pipeline

project5.ipynb: html file of the notebook

output_images/: A folder to contain output images

output.mp4: output video file

This write-up is also part of submitted work.

Note: The pipeline requires only image data sets from KTTI and GTI and does not use the dataset from Udacity. The vehicle and non-vehicle images folders, unpacked from their zip folders, should be put inside a folder called “image_bank”. Hence, to run this notebook

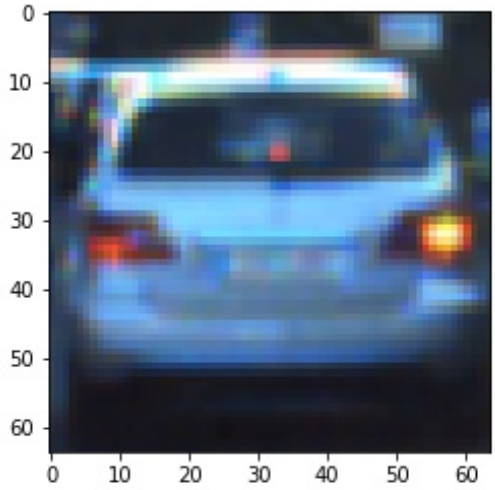
Thus,

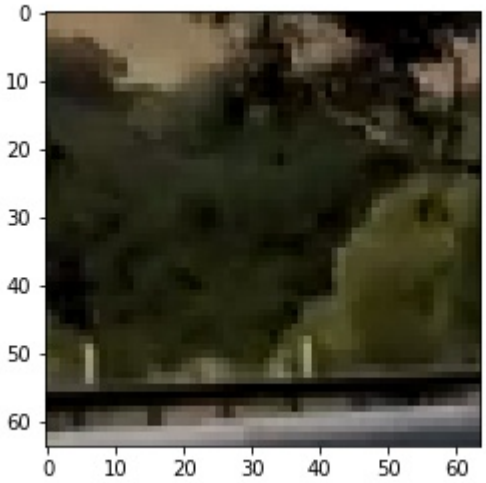
- Please make a folder called "image_bank" and get the vehicle images from GTI and KTTi from https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip
- Non-vehicle data is at https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip
- Please unzip the vehicles and non-vehicle files, and put the corresponding folder sunder image_bank

The individual Rubric points are addressed below

Image Collection:

To obtain the dataset for this work, the vehicle and non-vehicle images are obtained from the above mentioned links. It has been known that GTI set of vehicle images are in a time series i.e., these images contain cases where a single car has been shot in a sequence over a fixed time. To minimize this behavior, only every 5^{th} GTI image of a vehicle is used. This remarkably reduces the number of vehicle images as compared to non-vehicle images. To overcome this imbalance, the non-vehicle images are randomly removed so that 1760 images of each vehicles and non-vehicles type are present. Hence the complete dataset comprises 3520 images; each image is RGB and is of size 64 by 64 by 3. Sample images of each type are shown below. The code for this work is in Cell 1 and Cell 1a





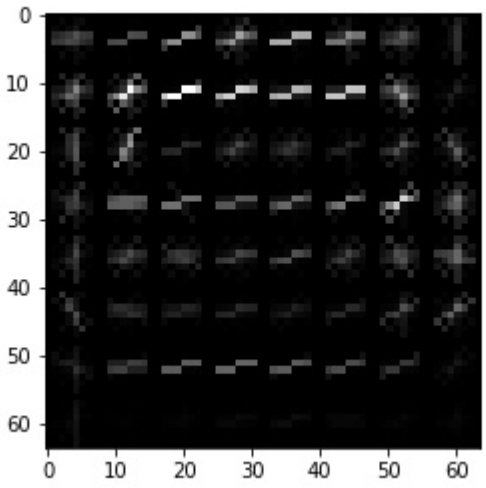
Feature Collection

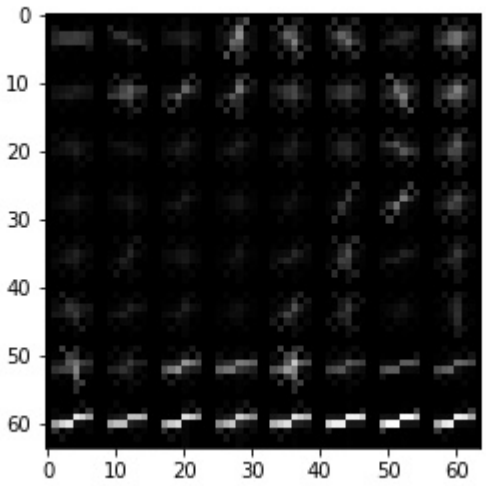
Three methods of feature collection are discussed during lectures namely spatial features (resized images), histogram features (concatenated histogram for colors in a color space), and histogram of oriented gradients (HOG). The code for obtaining these features is the same used in lectures and is present in cell 2. Cell 3 contain the parameters which have been used to extract features from image dataset. All three spatial, histogram, and hog features are included. For spatial features, the idea has been that 32 by 32 version of any image still contain useful information, and a sliding window of this size can detect far away cars in an image. A 32 by 32 image across 3 channels lead to 3072 features

Histogram features are used because color contain important features. YUV color space has been used in this work because of its bigger dynamic range. Moreover, white car in the test videos has been hard to detect with RGB and other color space, and YUV gave satisfactory results. Number of bins have also been tested with different values and 64 bins have given satisfactory results -in quizzes in the lectures. Hence, a total of 192 features are obtained from histograms.

HOG features are then also added for all channels. This has been advised in the discussion boards, as this may increase the total feature size but helps correct detection. The orientations considered for

HOG are 9 as it can help uncover different perspectives a car may offer when looked at. Pixels per cell is set to 8 by 8, and cells per block are set to 2, to obtain reasonable information from within the gradients. These values are also verified in the quiz setups designed to experiment with Hog settings and, they lead to reasonable results. A total of 1764 features per Hog channel are obtained. The hog version of test images for vehicle and non-vehicle, respectively are presented below.





At the end of the day, a total of 8556 features are collected for training from a single image. This will lead to a large processing time but will nevertheless lead to better car detection in the video, which is important for this projevt.

Classifier Training:

Once the feature set is obtained and scaled (Cell 4 and Cell 5), training of the SVM is done in Cell 6. For the SVM, kernel being used is ‘rtf’ with a C value of 10. These values have been obtained using *GridSearchCV* from *sklearn.model_selection* module. Linear and rtf kernels have been compared, and the accuracy on the training set has been very similar. For instance, with C = 10, the output has been

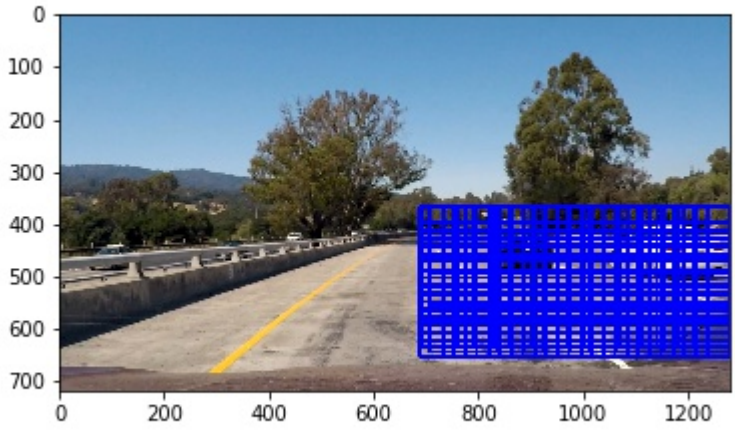
mean: 0.98260, std: 0.00265, params: {'C': 10, 'kernel': 'linear'},

mean: 0.98366, std: 0.00401, params: {'C': 10, 'kernel': 'rbf'}]

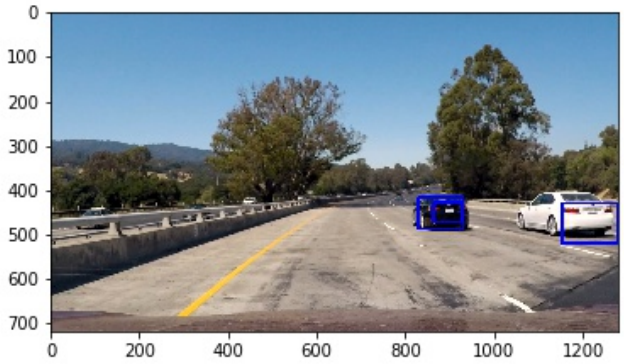
As shown above, rtf has slightly better mean accuracy but worse standard deviation. At the end, rtf kernel has been used which has resulted an accuracy of 99.5% on the test data. For rtf kernel, gamma value may also have been used, but seeing upon the accuracy on the test data, gamma value optimization has not been explored.

Sliding Window on Test Samples

Cell 7 , 8 and 9, shows how sample images present in the test_directory are uploaded and then sliding windows are drawn on them. The code used in cell 8 is based on the code in lectures. The slight modification is on the area being searched for the cars. The area is limited only to lower right quadrant of the image. This is consistent with test video where only cars form right emerge. This area selection will also minimize false positives coming from opposite direction. Moreover in this work, the windows are scanned from right to left, instead of left to right in the lectures. This will lead to early detection of cars appearing on the right. Following image show the area to be searched using windows of 64, 100, and 128 pixels.

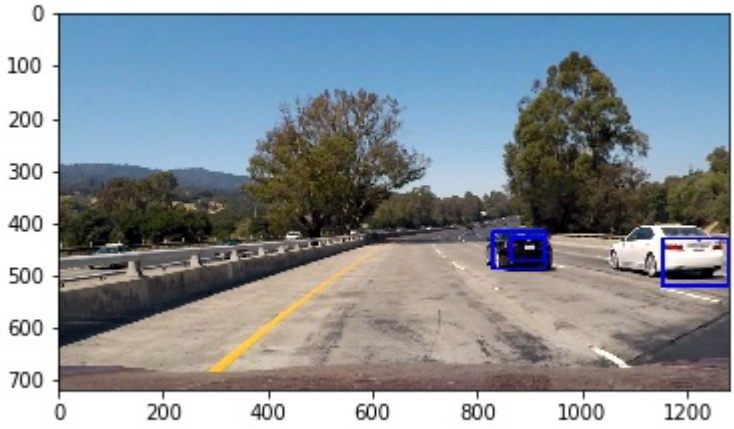


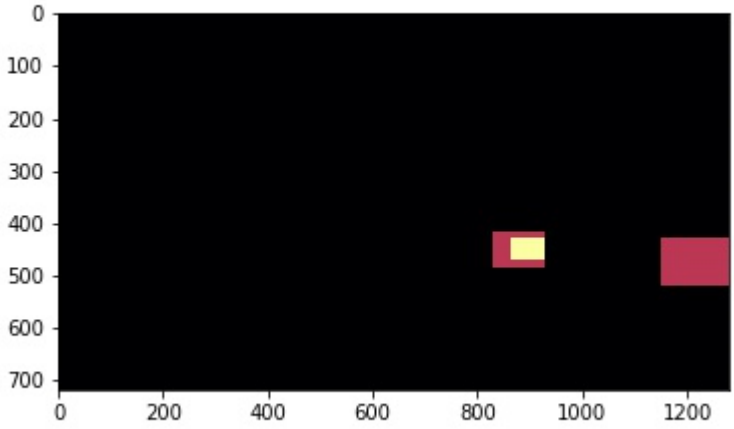
When such an image is searched for cars inside the classifier, the following output emerges (Cell 10)



Creating Heatmaps on Test Samples

Cell 11 and 12 contain the code for detecting and drawing heatmaps on the images obtained after classification. The code is based on the lectures, and a sample output is shown below.





Pipeline Settings and Video Generation

The eventual pipeline used for creating the video is in cell 14 and uses the trained classifier discussed above. For scanning for car using sliding window in every image, the following windows sizes (in pixels) are used.

win_size = [64, 100, 128 ,160, 200]

This covers a broad range and eventually leads to detection of cars which are close up or far away. The overlap of windows has been high (70%) to detect as many features related to a car in the image being scanned. To detect false positives in heat map, a threshold of 2 is used through hit and trial, which removes any wrongly detected area as cars. No additional filtering has been required because the area to be searched has been narrowed down already which reduces the false markers.

To smoothen the boxes around the detected cars, a deque or a ring buffer of size 25 is defined in cell 15. This size is chosen as it is the frame rate. This gives a running average once the buffer is full and minimizes the flickering of windows around vehicles. It is then used to correctly detect the cars. Moreover, from the combined mean heatmap, x and y coordinate of the detected boxes are passed through 3 sigma filter (cell 11) to try to fit the box around the car and to not use maximum and minimum values of coordinates as boxes.

Pipeline is verified for a sample image in cell 15 (result below) and is used to create the output.mp4 video in cell 16. The vehicles inside the frame are covered with a blue box

Discussion

There are number of ways this work can be improved.

1) The processing speed in pipeline for an image is around 10 seconds, which is really slow in real-time. A C/C++ based implementation may speed it up.
2) Another way to speed up is to perform Hog feature collection using subsampling as discussed in the lessons. This technique was not used in this work due to recurring errors (also discussed on the message boards), but can be used if implemented correctly.
3) The area to be scanned in the video is limited to lower right hand. In practice, this area should be over the complete horizon to cover right and left lanes (for example during overtaking). Moreover, for this video, it is fine not to scan the upper half of the images. However, if a huge vehicle like a truck appears in the periphery, it will cover the whole frame at one side. Hence, for a more robust vehicle detection, ideally the whole frame should be scanned.
4) At one point in the video, the black and white cars appear together as one vehicle. This is because the heatmaps from them are close and makes the label() function determine them as one. This can be corrected, for example, by using perspective from other cameras and not only using information from the main central camera.