Vehicle Detection Project

The files are:

detect_vehicles.ipynb

train_p

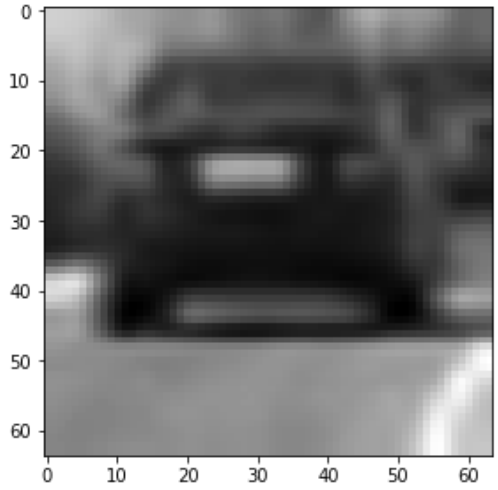
project_video_output.mp4

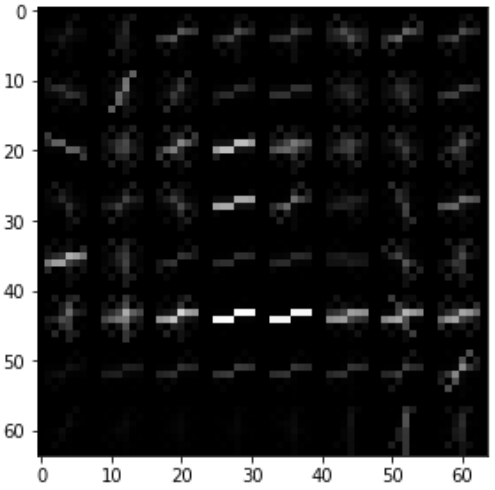
test_video_output.mp4

Histogram of Oriented Gradients

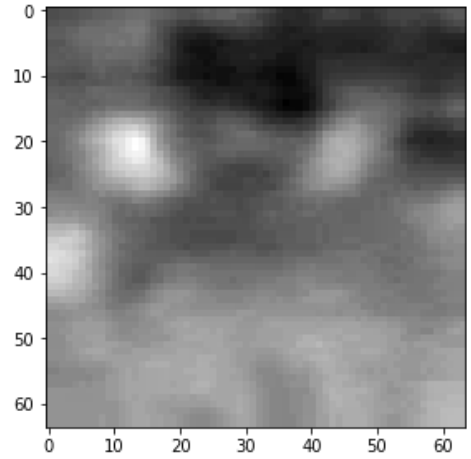
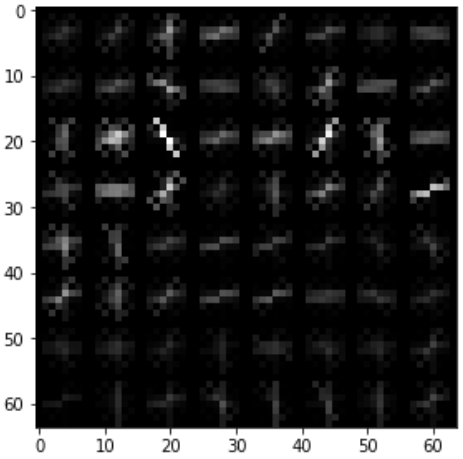
I ended up with parameters:
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9 # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 16   # Number of histogram bins

I choose to convert picture into YCrCb, these are three channels' hog pictures:
Channel 0

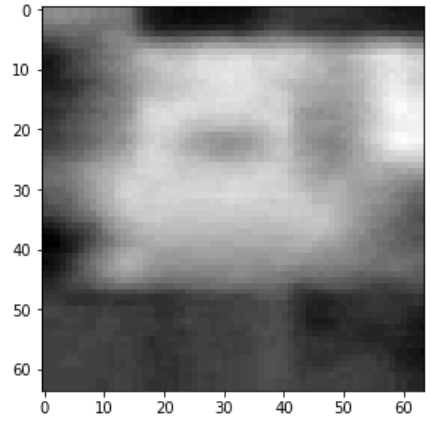
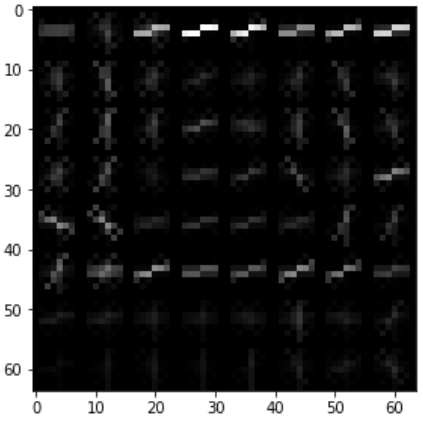




Channel 1

Channel 2

I use LinearSVM to train ~8k cars and ~8k no cars pictures.
Here is status I got:
nocars num = 8968
cars num = 8792
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 6108
44.8 Seconds to train SVC...
Test Accuracy of SVC = 0.9882

0.988 accuracy and with 20% of data as test set.
The train pipeline is in function **train_from_data()**.
In order to prevent re-train data several times, I wrote def load_predictor(), def save_predictor(svc, X_scaler) to load and save training predictor and X_scaler.

I tried to use only one channel's hog feature to train the data, but I got Test Accuracy of SVC = 0.9735, with 3 channels' features we got 0.9882.

Sliding Window Search

I tried generate sliding windows first and use the windows to generate features from subimage. As function **test_by_windows_search**() did in code.
But it takes a lot of time to re-compute the hog features.

So I used the smart way in course to avoid re-compute hog features, generate the hog feature once, as the function defined in **find_cars**()

I ended up with using three different scales to search vehicle in picture.
It's in function **pipeline**(image):
Scale = 1.2, 1.5, 1.7
And used the heat map to filter out with threshold <3
I used just scale 1.0, but it didn't work well, cannot detect out the vehicle in the test result pictures.
There are several test image examples, I detected by combining 3 scales with filter out heat map by 3.

Video Implementation

The function is defined in **test_video(),** which generates the bounding boxes frame by frame
The result is in project_video_output.mp4 and test_video_output.mp4.
It uses **pipeline()** function to generate result
I also tried to smooth the boxes we found in each frame. So I made a function **smooth_pipeline**() which will used last three frames' boxes to generate heat map and filter out by the threshold *3
It generates the videos project_video_output_smooth.mp4 (only by .subclip(15, 17)) and test_video_output_smooth.mp4.

The smoothed result bounding boxes looks more stable than non smooth ones. I think it gets a better result.

Discussion

I think for training accuracy could be improved above 99%, and the vehicles appear at very close or far from camera are harder to detect. Potentially we could use more different size of sliding windows to detect that. And training data should need more data like that.