Vehicle Detection

Vehicle detection project goals:

- 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.

To extract HOG features, I am using function *get_hog_features()* which uses *hog()* function to get histogram of oriented gradients.

Here is an example using the LUV color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

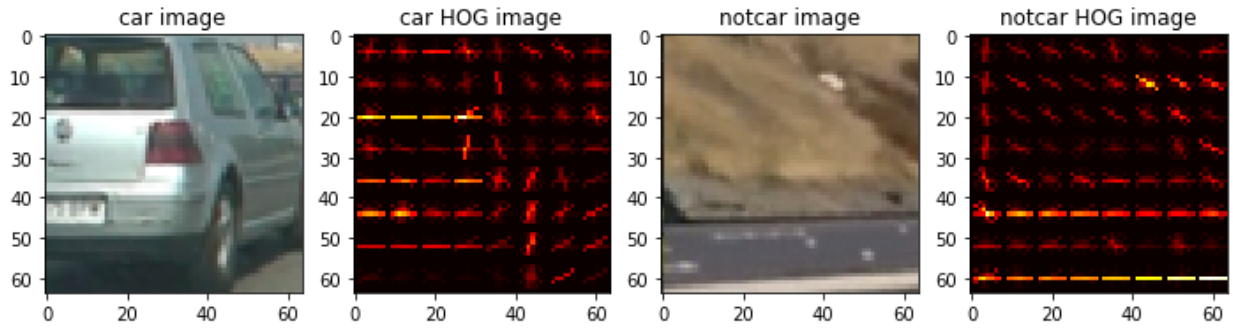

Figure 1. HOG features using L channel

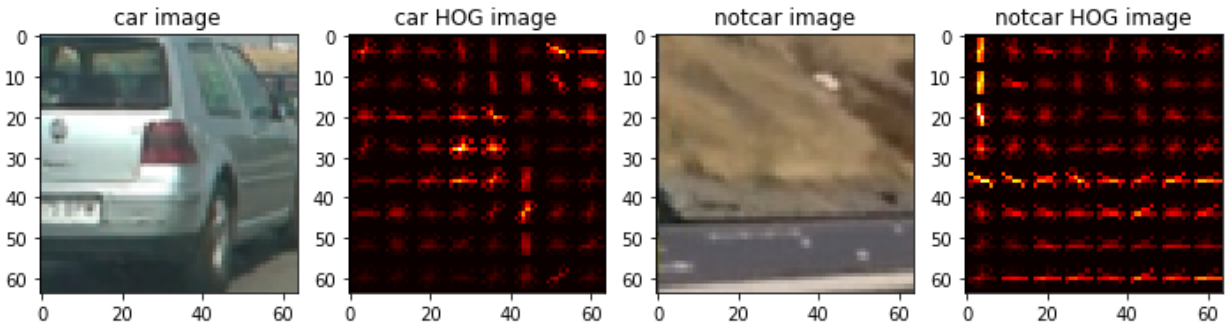

Figure 2. HOG features using U channel

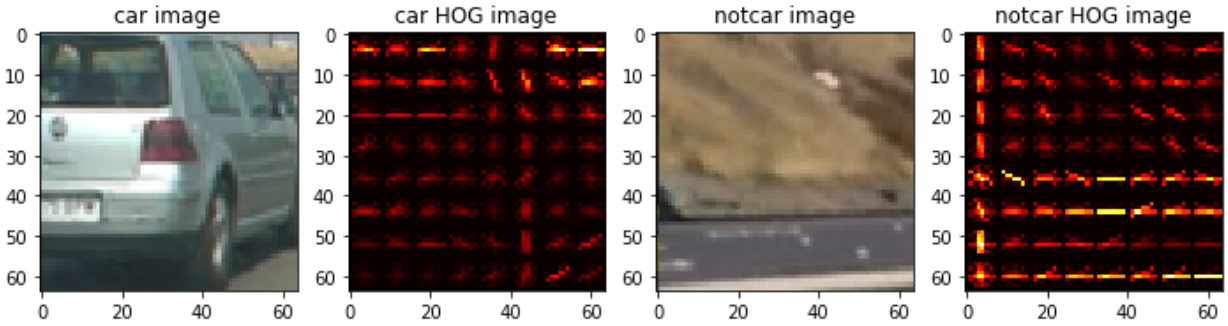

Figure 3. HOG features using V channel

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

From the above picture it can be seen that each channel picks up different features to focus on, so to train the classifier I used all 3 channels although it increases the feature vector length. To select the color space, I will monitor the training results and decide based on that as what might make sense to me might not be the same features the classifier focuses on.

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 trained a linear SVM in the "Training the classifier" block. I tried different parameters with 1000 random samples from car and not car images changing only one parameter at a time.

Initial parameters – color space RGB, orient 9, pix_per_cell = 8, cell_per_block = 2, hog_channel = 0, spatial_size = (16,16), hist_bins = 16 and all features activated.

Accuracy: 0.958

Changed colorspace to HSV >> Accuracy: 0.96

Changed colorspace to LUV >> Accuracy: 0.975 (2 580 features & 7 s computing features for 2000 images)

Changed colorspace to HLS >> Accuracy: 0.962

Changed colorspace to YUV >> Accuracy: 0.972

Changed colorspace to YCrCb >> Accuracy: 0.968

Kept LUV and changed hog_channel to ‘ALL’ >> Accuracy: 0.99 (6 108 features & 14.5 s computing features for 2000 images)

Kept ‘ALL’ and changed orientations to 6 >> Accuracy: 0.992 (4 344 features & 14 s computing features for 2000 images)

Kept 6 and changed spatial_size to 32 times 32 >> Accuracy: 0.975 (6 648 features & 13.2 s computing features for 2000 images)

Disregarded last change and changed histogram bins to 32 >> Accuracy: 0.98 (4 392 features & 14 s computing features for 2000 images)

Disregarded last change and switched off spatial feature >> Accuracy: 0.97 (3 576 features & 13.2 s computing features for 2000 images)

Disregarded last change and switched off histogram feature >> Accuracy: 0.968 (4 296 features & 12.3 s computing features for 2000 images)

Disregarded last change and switched off HOG features >> Accuracy: 0.928 (816 features & 2.9 s computing features for 2000 images)

Final training parameters that are also saved off to pickle file:

```
color_space = 'LUV' # RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 6
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL'
spatial_size = (16,16)
hist_bins = 16
spatial_feat = True
hist_feat = True
hog_feat = True
```

Figure 4. Final training parameters

Results after training on all the data, accuracy: 0.99

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?

I took the function “find_cars()” used in the course that takes the HOG feature of the whole picture at once to cut down processing time of each frame as there are 1 261 images in the whole video which means that one extra second processing time for each frame would result 21 minutes longer processing for the whole video.

To decide the scale and overlaps, I drew different possibilities to seek out the scale. In the following figure we can see the true detections according to the classifier.

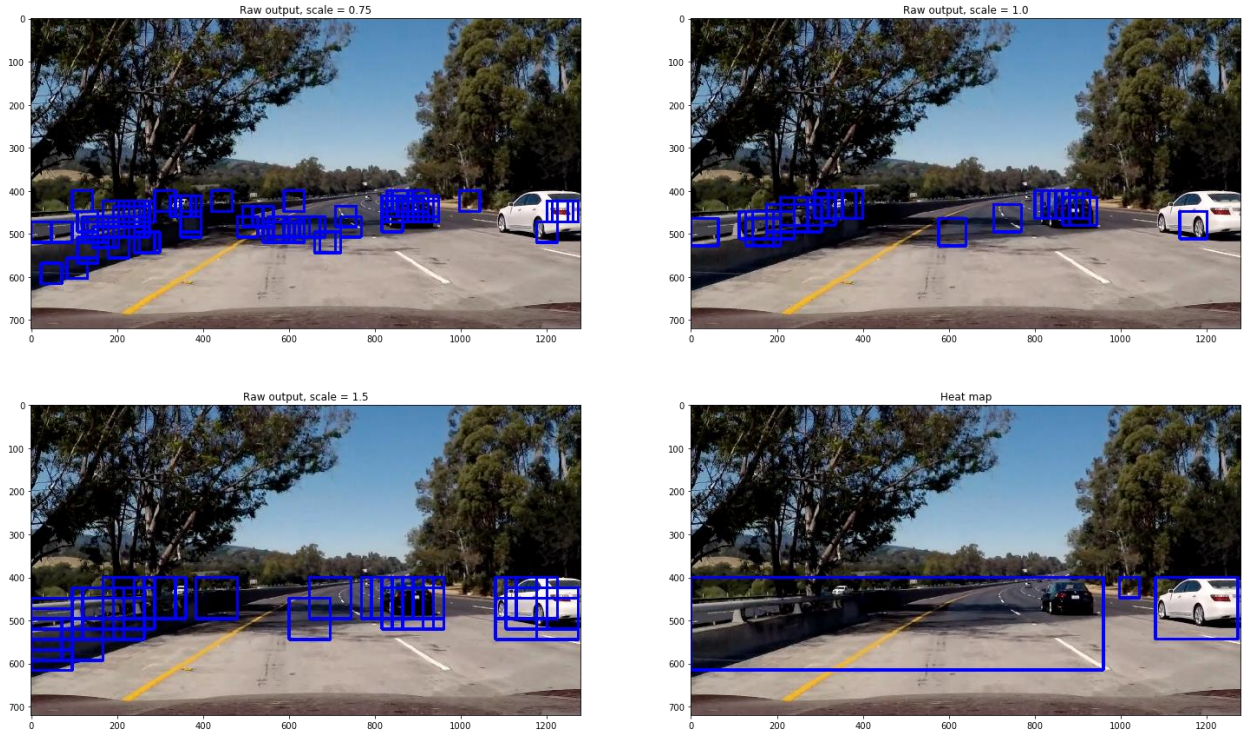

Figure 5. Classifier car detections

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?

As the Figure 5 seems a bit too boxed out, I figured to try to shift the orientations for the HOG features from 6 to 9 and retrain the classifier and see the results which can be seen in Figure 6.

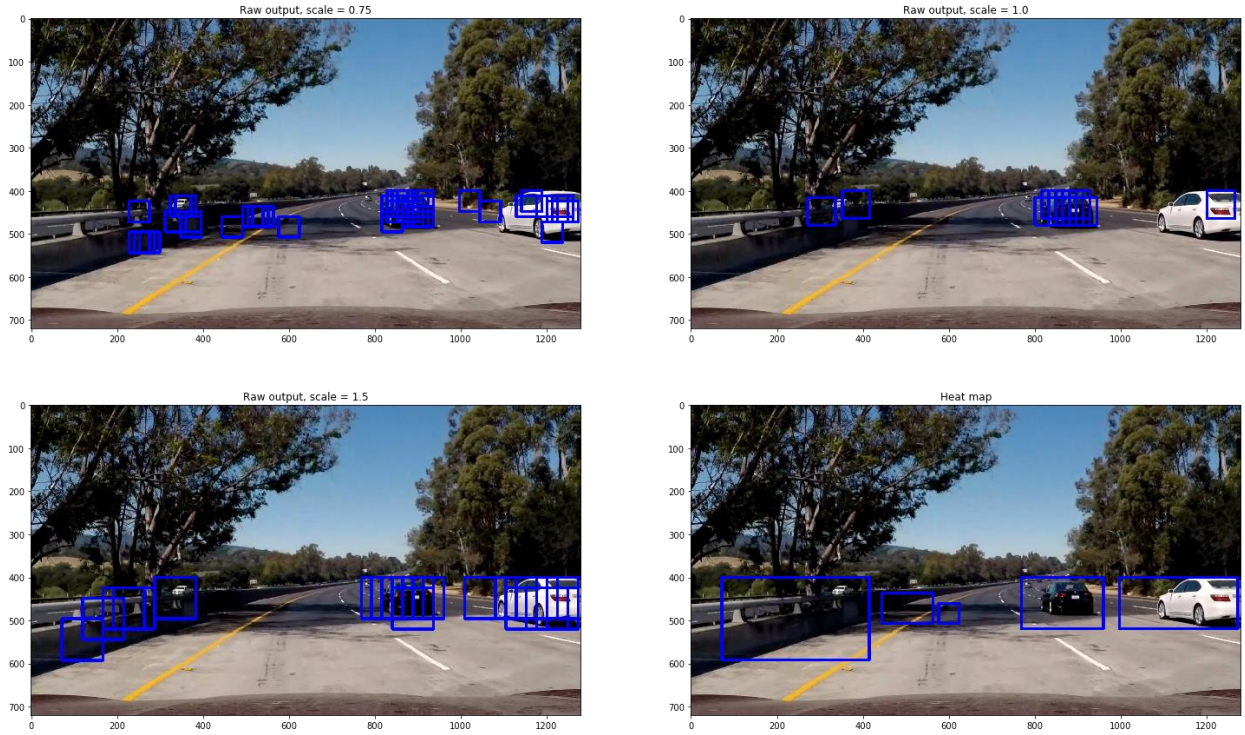

Figure 6. Classifier detections in case orient = 9

In the figure 6 we can see that the false positive detections have decreased significantly.

From the Figure 6 we can also see that the scale 0.75 produces a lot of false positives and even though it hits cars as well, the scale 1 and scale 1.5 seem to catch cars better and produce less false positives. Next, I replaced the 0.75 scale with scale 2 and used heat_map = 3 and result can be seen on the following picture.

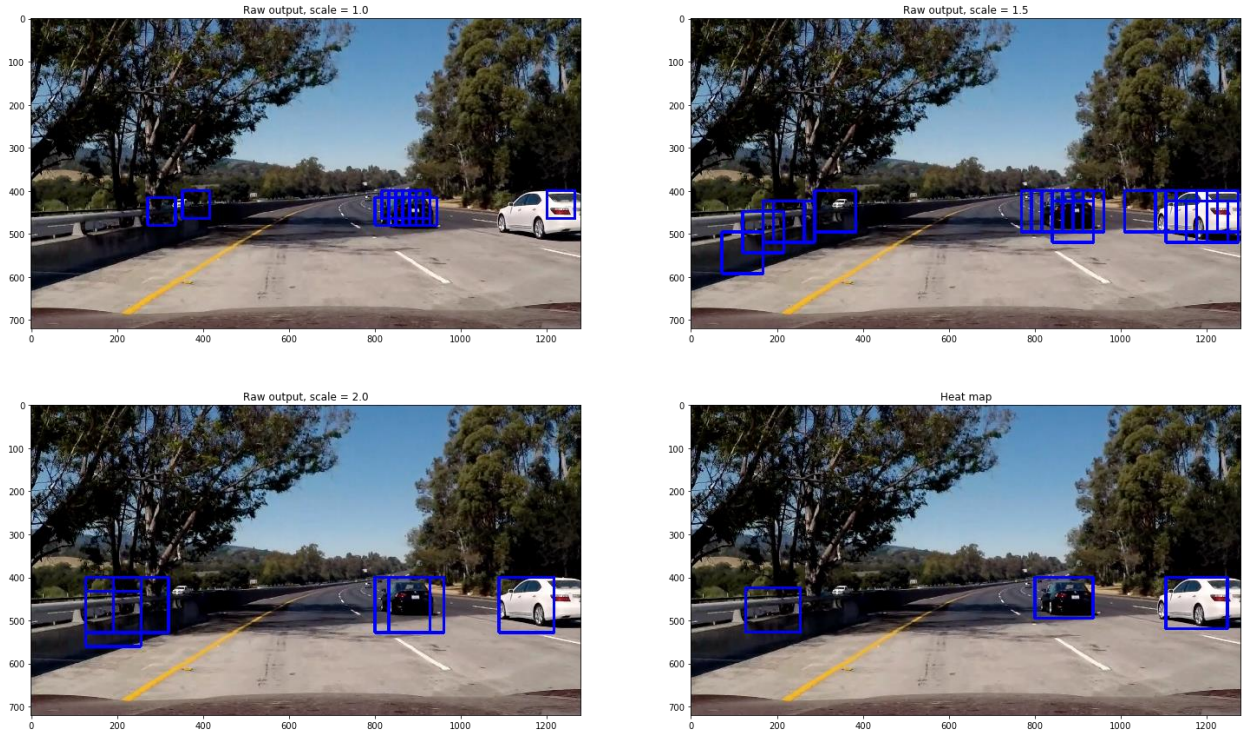

Figure 7. Detections with scales 1, 1.5 and 2 and final image with heatmap = 3

Video implementation

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 video is added to the project. The file name is “project_video_output.mp4”.

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.

As I saw from static images that there are quite a few false positives I decided to try to add bounding boxes to an array and take the last 30 elements from the array, but as I have 3 sized bounding boxes added for each frame it means I will create a heat map from last 10 frames and threshold it. This is implemented in the “process_img()” function under class “Detections” where I run the “find_cars()” function with 3 different scales and add the results to recent_findings array which I later use to take last 10 frame detected car frames and threshold it. This in theory should remove most of the false positives as the cars have relatively same speed as our car they wouldn’t move a lot from frame to frame but it is expected that false positives are more random so adding them up from 10 frames and thresholding them would eliminate most of the false positives. From Figure 7 I saw that threshold 3 might should remove most of the false positives, but still catch the cars quite well, so for image I will try to threshold the 10 last frames with threshold 10 times higher minus some change margin, so I think ~20 should be a good threshold.

Discussion

I noticed that the pipeline seems to work pretty well overall, but there are some false positives emerging when the tree comes on the left side and the shadows are recognized are cars. The same situation can be seen in the Figure 7 for example, as I took the most difficult frame to test on and see the results. Most of the other frames had few false positives which could be filtered out with the heatmap per single image. It would help to track the center of the detections and estimate the speed of the cars to predict where they would emerge on the next image and filter out the false positives even better. The pipeline might face some difficulties detecting scooter and other unusual vehicles right now as there weren't any other road legal vehicles in the training set.