VEHICLE DETECTION

Author: Jumana Mundichipparakkal

The goals / steps of this project are the following:

- 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)

My solution:

- Notebook: vehicle_detection_hog_param_exploration

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

- Python Module: feature_extraction.py in vehicle_detection module
- Output: output_images/hog

A. Get_hog_features():
I used this function to get the hog features. The code was more or less reused from the course material. Figure 1 and 2 shows the results of applying hog features on one of the car images and a noncar image.

Example Images:

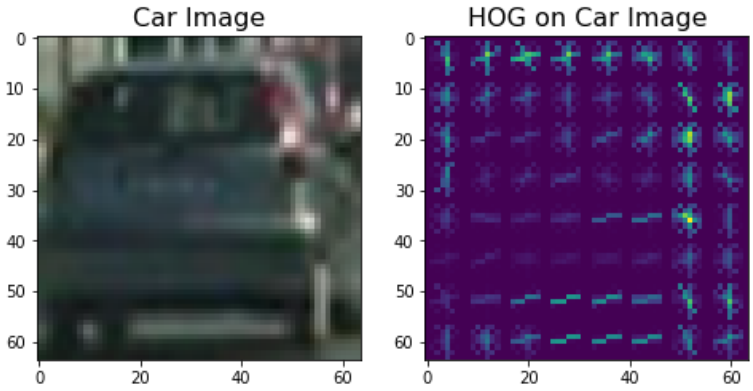

Figure 1 Hog features applied on an example car image

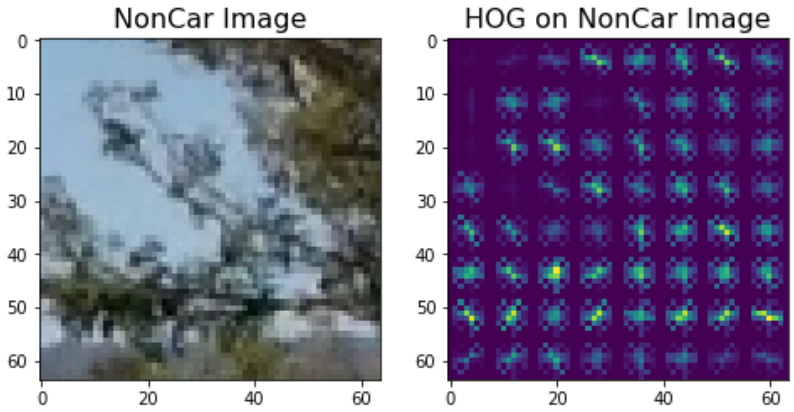

Figure 2 Hog features applied on an example noncar image

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

- Python Module: feature_extraction.py module in vehicle_detection package
  Classifier.py module in vehicle_detection package
- Output: output_images/hog

A. hog_param_exploration()

For choosing HOG features, I took an exhaustive search approach. I created a list of various values each parameter can take, and applied hog feature extraction and trained them using LinearSVC. I created a pandas dataframe out of all the results, and figured out the top 5 configurations for best accuracy. Among the top 5, I chose the configuration that took the minimum most training time as final solution.

This uses the **train_hog_features()** method in classifier to extract hog features and then run classifier on it for training.

Result:

hog_param_exploration.xlsx has the data for all configuration tests done.

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).

A. train_hog_features()

I used only HOG features for training classifier.

Hog feature extraction is performed for both car images and non-car images using the hog space parameters defined. These feature vectors are then combined and a label vector of 1 for car features and 0 or non-car features is created. All the feature and label vectors are the split into training and test sets for classification. As a nest step, I trained a linear SVM with the default classifier parameters using the training and test data. Accuracy of classification and training time is recorded for every iteration of hog parameters chosen.

Final Results:

I had done 287 combinations of hog parameters and was able to achieve an accuracy of 0.9868 as maximum. This took a training time of 6.20 seconds. I listed the best 5 accuracy configurations and found a configuration which gave an accuracy of 0.9862 and training time of 0.71 seconds.

I have added the final notebook execution as html file: vehicle_detection_hog_param_exploration.html

SLIDING WINDOW SEARCH

My Solution:

- Notebook: video_detection_sliding_window.ipynb
- Python Modules: sliding_window.py in vehicle_detection package
- Result notebook: video_detection_sliding_window.html

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?

A. detect_cars()

I modified the find_cars() method given in lecture to also support multiple channels in Hog. This function extracts Hog features with the chosen hog parameters from last section and performs the prediction on the image using svc model obtained by performing a linear svc classification on car/non-car images. The method is adapted to do Hog feature extraction only for entire image only once and then while doing sliding window search inside the image , full image features are subsampled per window. This approach basically saves a lot of time instead of doing feature extraction on every single window. After performing classifier prediction of all windows, it returns the list of rectangle objects per positive car-existence predicted window. Figure 3 shows the number of rectangles detected as locations of car; there are 8 detections in this image, when windows are taken between 400 and 550.

Figure 3 Applying sliding window search on an image with Hog feature based SVC classification

The result above has many false detections and searches only a single area defined by a definite ystart and ystop for windows. We need to perform sliding window on all positions that can likely have cars as well as find a better way to make identify real cars from all enlisted detection positions.

A. car_location_exploration()

Choosing the exact windows and locations of windows is a hard task as there can be infinite number of possibilities in terms of size of windows and positions. Initially, I did an exhaustive search approach using the detect_cars() method, where it creates multiple combinations of ystart, ystop and scale values that I chose. I chose the values based on likely locations of cars considering cars are mostly in the lower bottom of the image. This indeed is a very time taking process, having many permutations and combinations of parameters and performing the same approach on video took about one and half hours. An example image showing detections from sliding window is presented in Figure 4.

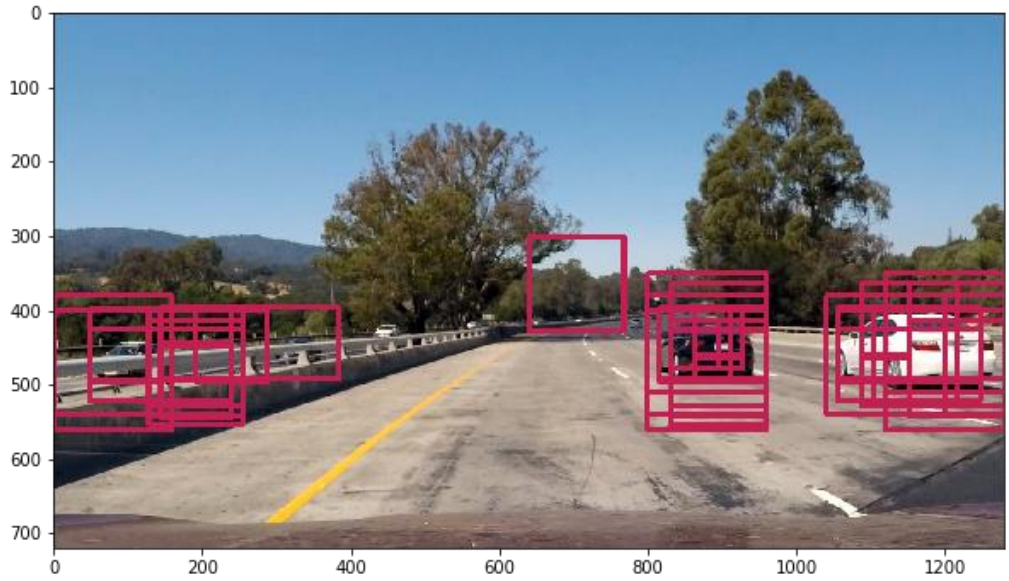

Figure 4 Detected car windows performing sliding window search using an exhaustive search approach.

B. add_heat()

For filtering out wrong detections and overlapping images, we first try to create a heatmap image of all the rectangles. Applying heatmap on Figure 4 gives Figure 5.

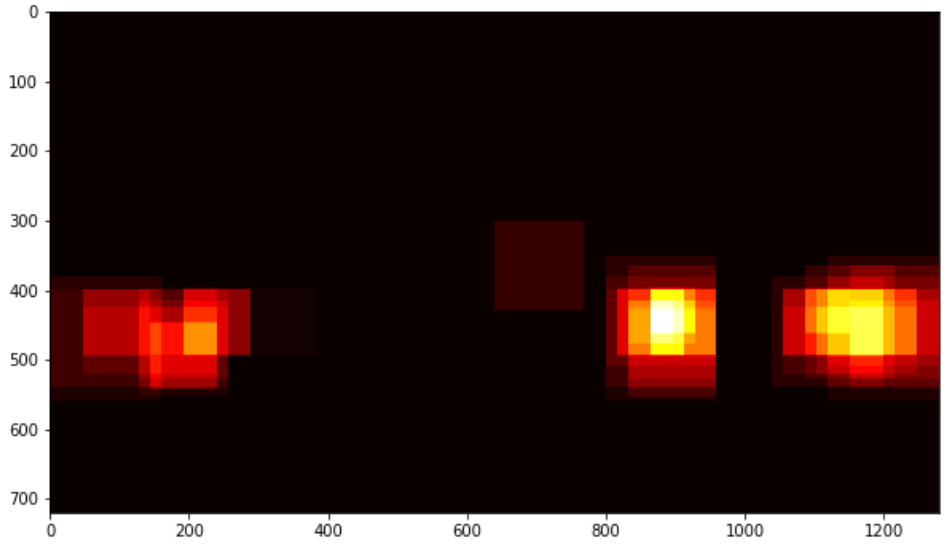

Figure 5 Heat map of windows detected in Figure4

C. apply_threshold()

As we can see from Figure5, there are a lot of duplicate detections as well as false detections. For eliminating false detection we apply threshold on heatmap image which basically counts of number of overlapping rectangles.

I used a threshold of 5 initially, but threshold has to be larger in this approach as I have drawn a lot of rectangles of various sizes. Then I use the scipy label function to create a single box out of overlapped detections. Figure 6 shows the labelled image after thresholding Figure 5.

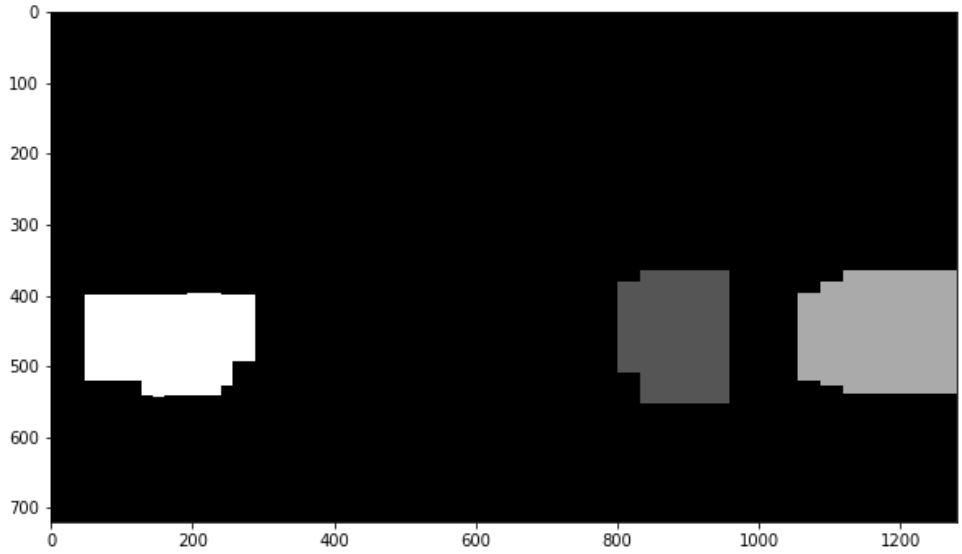

Figure 6 Thresholded and labelled heatmap image in Figure 5

As can be seen, we had a lot of false detections in the left most end contributing to a false detection.

D. draw_labeled_boxes()

This function is used to finally draw a rectangle around all detected pixel regions in the thresholded heatmap image.

Figure 7 Final detection of cars as per the set of algorithms used for the image Figure 3

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?

Optimization Steps:

1- car_location_exploration_fast()
Reduce the sliding window search time: Last approach had a significant time cost. To solve this, I decided to choose less number of windows around maximum probability of detection of car in the image as well as decide the scaling size according to the distance from the camera. This time, I created a list of ystart and ystop positions as well as scaling values. This approach brought down the project video processing to around 10 minutes from one and half hours from past approach.

2- Threshold value: Threshold value has to be reduced this time as number of rectangle detections have significantly reduced. I had to go through a bit of trail and error to eliminate all the false positives. I used this solution to experiment with images as well as project video to derive an appropriate threshold value. Threshold value was taken 1 this time and it seems to have worked fine for all test images.

Result: All result images are shown in the output html file of workbook.

video_detection_sliding_window.html

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The past frame shown in Figure 7 with this new approach gives no false detections. All the steps of pipeline performed on same image is shown in Figure 8.

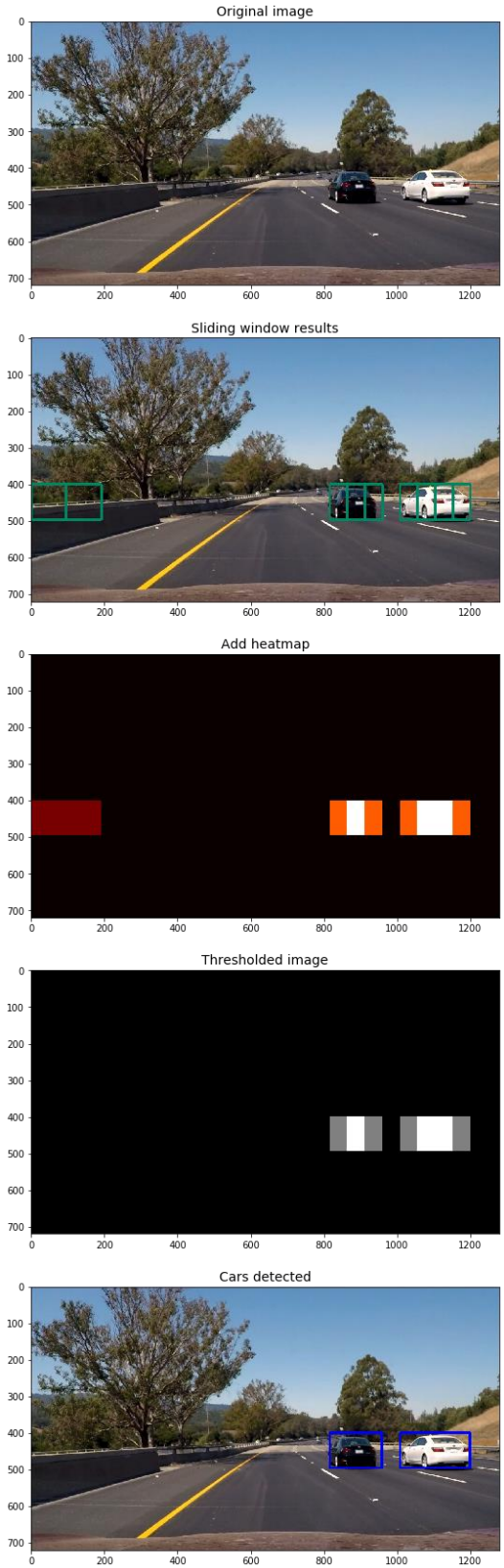

Figure 8 All steps in vehicle detection pipeline performed on a test image

VIDEO IMPLEMENTATION

My solution:

- **Output:** project_video_out
- **Notebook:** vehicle_detection_sliding_window-video

4. 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.)

Submitted as project_video_out.mp4

5. 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.

For video images, I encountered getting a lot of false detections in the left side of the road as well as missing frames. To solve this, I took the approach of remembering detections from past frames as we did for advanced lane finding. I chose to remember rectangles from past 20 frames and then add it all on heat map. This means threshold values can be raised higher, considering the number of rectangles accumulated over frames. For final solution, after working with video frames., I chose threshold value of 10 which gets added by half of the total rectangles stored from past frames.

Class used for remembering past frames: carPosition() with and update_detected_car_data()

Car detection_pipeline: detect_car_pipeline_video()

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?

Multiple issues with my current solution are:

1- I have false detections at times in the left hand side of the road. I was not completely sure if it was a concern as the lane in which the driving vehicle is moving can be the middle lane. I do need a more robust method for filtering out noise. Currently, the solution takes only the Hog features into account. I think detections can be more robust on adding color feature detections.

2- Threshold value is chosen based on trial and error. This was already a very painful process and I am not confident of it to work well with video tracks in widely varying light conditions. This case can also be solved by using various color spaces and feature extraction.

3- Noise detections, in areas with very tiny objects can be eliminated out by some smoothening or filtering for noise which can be done as a pre processing step.
4- Above all, more training images will help for the classifier to predict better as for any other learning problem. I think algorithm works well for images whose back side is seen, but not that great for side view. May be adding more side view images in training set should help in this case.

Currently, I am happy with the results for a first attempt. I wish to make more changes especially to add color features for classification.