Vehicle Detection and Tracking

Writeup / README

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.	The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.	This project submission includes the following files. • The jupyter notebook with code(CarND-Vehicle-Detection-Copy9.ipynb) • The html and pdf of the notebook (CarND-Vehicle-Detection-Copy9 .html, CarND-Vehicle-Detection-Copy9.pdf)

Histogram of Oriented Gradients (HOG)

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.	Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.	The hog features are extracted using the function get_hog_features. Mainly this function is used straight from the lesson and exercises. Cell 79 has the final code of the function. The final hog parameters are settled on the relative accuracy of the parameters. I have shown a combination of parameters tested in the notebook. Please refer the cells 82, 83, 104-113 in the notebook. In the cells 104-113, changed the parameters and the boxes found are shown in each of the test images. As we can clearly observe, the following parameters worked better. orient = 31 # HOG orientations

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

		pix_per_cell = 5 # HOG pixels per cell
		
		cell_per_block = 2 # HOG cells per block
		
		Though I have tested, many other combinations, I'm reporting just these sample combinations

Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).	The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.	As can be seen in the cells 104-113, SVM is used as classifier. One observation is the the accuracy of the model is better when we have more orientations than pixels per cell or cell per block.

Sliding Window Search

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

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 sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.	The main functions are again mainly borrowed from the lecture and exercises.
		
		The following functions are implemented;
		
		get_hog_features()
		
		bin_spatial()
		
		color_hist()
		
		extract features()
		
		slide_window()
		
		draw_boxes()
		
		single_image_features()
		
		search_windows()
		
		convert_color()
		
		These functions can be found in cells 79-81.

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		using the find_cars() function and later enhanced it as find_cars_new(). Here I added a few enhancements such as; * parallelizing hog feature extraction * profiling each step; In addition the find_cars_new() itself is used as an instance in the function parallelize_find_cars() Different scales are used and each of them are tested for performance and based on the results, finally decided to use just 3 scales. Scales tested: scales1 = [0.95] scales2 = [1.15, 1.35, 1.55] scales3 = [0.75, 0.95, 1.15, 1.35, 1.55] scales4 = [1.15, 1.25, 1.35, 1.45, 1.55]

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

		scales5 = [0.75, 0.8, 0.85, 0.9, 0.95, 1.05]
		
		scales6 = [0.75, 0.85, 0.95, 1.05, 1.15, 1.25, 1.35, 1.45, 1.55]
		
		scales7 = [0.75, 0.8, 0.85, 0.9, 0.95, 1.05, 1.10, 1.15, 1.2, 1.25, 1.3, 1.35, 1.40, 1.45, 1.5, 1.55, 1.60]
		
		Finally decided to use scales2

Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?	Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)	To improve the reliability of the classifier, I have used numerous methods. 1. The threshold used is 3, as im testing for 3 different scales. During my testing pretty much all positive locations had a found. 2. Used head maps with the threshold of 3. 3. To eliminate the false positives, I added code to check if a vehicle was there in the previous frame(s) and add the position to that car already found. Vicinity check is done by checking the centroid of the new box is within 48 pixels or the old box in the

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		previous frame. 4. To mitigate the error of having the boxes in unknown or stuck at the end of previous vehicle detection, I determined the current active cars and only update them. 5. In addition a size check was done for boxes less then 48x36 pixels were ignored. Most of this code is implemented using the functions: add_heat() apply_threshold() draw_labeled_boxes() in cell 146. For tracking the cars and their position a new Car class is created. This is in cell 145. A couple of more functions for centroid and distance.

Video Implementation

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
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 sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.	The video is embedded in the html file and also in the project_videos_output folder of the zip file. Please refer the the videos with 9 as the last digit in the filenames. The videos are named with suffixes of with_vicinity and without_vicinity With vicinity means all the bounding boxes are shown for all cars in yellow, with the current detection shown in blue color.
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.	A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false	The same points I mentioned above to improve the reliability, addresses this point. To improve the reliability of the classifier, I have used numerous methods. 1. The threshold used is 3, as im testing for 3 different scales. During my testing pretty

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
	positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.	much all positive locations had a found. 2. Used head maps with the threshold of 3. 3. To eliminate the false positives, I added code to check if a vehicle was there in the previous frame(s) and add the position to that car already found. Vicinity check is done by checking the centroid of the new box is within 48 pixels or the old box in the previous frame. 4. To mitigate the error of having the boxes in unknown or stuck at the end of previous vehicle detection, I determined the current active cars and only update them. 5. In addition a size check was done for boxes less then 48x36 pixels were ignored. Most of this code is implemented using the functions: add_heat() apply_threshold()

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		draw_labeled_boxes() in cell 146. For tracking the cars and their position a new Car class is created. This is in cell 145. A couple of more functions for centroid and distance.

Discussion

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

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?	Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.	I have thoroughly enjoyed working on each of the projects for this course. But I have spent a little more time to bring in the real time aspect in to this project. But I have to admit, while seeing the hope, im not really successful.
		
		The idea was to achieve, processing 24/30 frames per second.
		
		As you can see in the cells 117-123 I have done a lot of profiling.
		
		In cell 121, we can observe
		
		`Length of task list:  36`
		
		`Number of processes used: 4`
		
		`The times for each task are: [0.180446, 0.1489, 0.156686, 0.166166, 0.153977, 0.177666, 0.140012, 0.089308, 0.099289, 0.123507, 0.115486, 0.127686, 0.111594, 0.091043, 0.078914,`

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		`0.101854, 0.094594, 0.097585, 0.089808, 0.065616, 0.098844, 0.105233, 0.063709, 0.084402, 0.069043, 0.08867, 0.067207, 0.08404, 0.093701, 0.10529, 0.064909, 0.063618, 0.086028, 0.059732, 0.070263, 0.043568] with:` `Minimum: 0.043568 Maximum: 0.180446 Average: 0.1016 seconds` `Time taken for processing each image 4.5231` Also, in cell 123, we can observe, `Length of task list:  3` `Number of processes used: 3` `Process-4634, Step 1: Divide with 255, processing time: 0.02824 seconds` `Process-4634, Step 2: Resize if scale is not 1: 0.016095  seconds` `Process-4634, Step 3: Get HOG channels: 2.5e-05 seconds` `Process-4634, Step 4: Define blocks and steps as above: 7e-06 seconds` `Process-4634, Step 5: Compute individual channel HOG features for the entire image: 0.411193 seconds` `Process-4634, Step 6: Misc initializations: 1e-06 seconds`

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

		`Process-4634, Step 7: for loop: 0.025537 seconds`
		
		`Process-4634All steps time: 0.48109799999999 seconds`
		
		`Process-4634, Find cars processing time: 0.481233 seconds`
		
		From this the conclusion is that most of the processing time in the entire sequence is taking for extracting the hog features.
		
		Python multi processing is used to spawn multiple processes to handle each patch / sub window. The least I could see in some cases we around 0.04 seconds with a sub window size of 170 pixels.
		
		But in some cases during the testing, I could also observe around 0.01 seconds. This mean I can do about 40-100 frames per second, theoretically
		
		But then the reality kicked in. Looks like python multi processing seems to have a

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		lot of overhead. As we increse the number of processes, the overhead is too much it kills the benefit. The best performance I achieved so far was about 1.6 – 1.7 seconds per image. I’ll certainly play around with python multi processing and see if I can achieve the performance. In addition i’ll also try playing around with C++ to see how much performance gain i’ll get. Another interesting thought as I write the report is that, once a car is found and a bounding box is decided, I can have a dedicated thread around that box with 25-50 pixel area for the already found car, and another dedicated thread or process to sweep the whole image or parts of the image continuously.