Meets Specifications

 Excellent job with the project! I'm impressed with how you iterated on the pipeline and leveraged all the main concepts of the Vehicle Detection lesson in your submission. 

Creating a pipeline that operates in real-time would be tough using an SVM/HOG approach, but there are deep learning methods that can help accomplish this — to see some examples you can check out [this student's YOLO project](https://github.com/JamesLuoau/Self-Driving-Car-Vehicle-Detection) or look into a [Keras SSD implementation](https://github.com/rykov8/ssd_keras" \t "_blank).

**Writeup / README**

**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.**

**Histogram of Oriented Gradients (HOG)**

**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.**

Nice job extracting the HOG features, and discussing how you arrived at your [HOG parameters](http://www.learnopencv.com/histogram-of-oriented-gradients/). 

To speed up the pipeline while maintaining good accuracy on the video, you can also try using more pixels per cell along with more orientations. For example...

color\_space = 'YUV'

orient = 11

pix\_per\_cell = 16

cell\_per\_block = 2

hog\_channel = "ALL"

**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.**

Good work training the linear SVC with the [extracted HOG features](http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/) and added color features. 

* To help the SVM generalize its predictions on the video, you could try **lowering** the ["C" parameter](http://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel) instead of raising it— a lower C value should help with any overfitting of the training data. (e.g., try .01 or .05)
* And to reduce the feature dimensionality and speed up the pipeline, you can consider removing the color histogram — many students are able to exclude color histogram features and still get good results.

**Sliding Window Search**

**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.**

Nice work implementing the [sliding window search](http://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/), and describing how you subsampled 6 scales of the whole image HOG extraction. 

You might be able to use fewer scales here — many students do well with just 1-3 scales. (e.g., I've had success with using 2 scales [1.2, 1.8] with 75% window overlap)

**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.)**

**Video Implementation**

**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.**

Terrific work processing the video and identifying the closest vehicles in the video!

Despite a few missed detections of the white sedan when it's farther away, it appears the classifier is doing a good job of detecting cars. 

 **Tip**: For future reference, if you ever need to save an image of a clip at a given time you can use save\_frame...

my\_clip = VideoFileClip("project\_video.mp4").resize(width=320)

my\_clip.save\_frame("frame.png", t='00:00:42') # saves frame at time t

And to create an animated gif from a clip you can use write\_gif...

my\_clip = VideoFileClip("out\_project\_video.mp4").resize(width=320)

my\_clip.subclip(28,30).write\_gif("sample.gif", fps=25)

[](https://udacity-reviews-uploads.s3.amazonaws.com/_attachments/15447/1506093908/sample.gif)

**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 positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.**

Nice job filtering the vehicle detections by thresholding the heatmaps, and drawing the bounding boxes around the detected blobs. I like how you utilized the deque data structure to store the heatmaps. 

**Discussion**

**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.**