Project 5 Resubmission:

Points revisited:

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.

You implemented the sliding window approach here with three different sizes of widows (scales). You carefully chose an appropriate y-axis range to narrow the search. Nice work.

However there are many frames in which the car is clearly visible and yet it is not detected (false negative detections). Two reasons for this can be that

- there is no appropriate window size which could capture the car, try to fine tune the window sizes or add new sizes (e.g. there is a relative big gap between scale 1.25 and 1.75, so you might want to consider adding scale 1.5, too)
- for certain window sizes the overlap ratio is not big enough, try to increase the overlap ratio which is 50% now (because `pix_per_cell=16` which means that there are 4 cells in a 64x64 image and `cells_per_step=2`, meaning that one step is 2x16=32 pixel).
    - By decreasing the value of `cells_per_step`
    - By decreasing the value of `pix_per_cell` (e.g. back to 8)

This was experimented. The initial reason for increasing the pix_per_cell was to cut down on processing time, evidently the quality of the end product was affected.

The pix per cell was decreased. This in turn allowed a lot more information to be captured. As one would expect, this also increased the amount of false positives. Nevertheless, the amount of true positives also increased proportionally. Using this increase in true positives, a “outlier” filter can be more easily implemented (see next point).

More window sizes were also included to capture cars of different sizes on the image.

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 sliding window search with SVC classifier has been used correctly to identify the vehicles. Great job.

However there are still false positive detections as well as many false negative detections in the video. Also, the bounding boxes of the detected vehicles are not stable, sometimes splitting. To overcome these issues you might consider applying the ideas given in the comments of the previous and next rubric points.

Also, I suggest checking if the color spaces were used consistently. Depending on the libraries you use, it is possible that the channel order (RGB or BGR) is different during the training and during the video processing. If this happens, the detection quality becomes rather low. If this is the case here, you can simply use `cv2.cvtColor(img,cv2.COLOR_RGB2BGR)` or `cv2.cvtColor(img,cv2.COLOR_BGR2RGB)` to ensure that the color order is the same.

This was verified. Though no error in this regard was detected, the color conversion function was added to make sure all image read are in the same color space:

```
def convert_color(img, conv='RGB2YCrCb'):
    if conv == 'RGB2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    if conv == 'BGR2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    if conv == 'RGB2LUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
```

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.

Here the idea is utilizing the fact that in the subsequent frames the cars are located at or near the same positions, while false positives are present only for 1-2 frames. There is no such optimization in the source code.

The simplest way to implement this is using multi-frame accumulated heatmap: just store the heatmap of the last **N** frames (**N** can be 5 or 8) and do the same thresholding and labelling on the sum (or average) of these heatmaps.

As a side effect this techniqe results much more stable bounding boxes as well.

To store the heatmaps I suggest using `collections.deque`, in this way you do not need to delete the oldest heatmap:

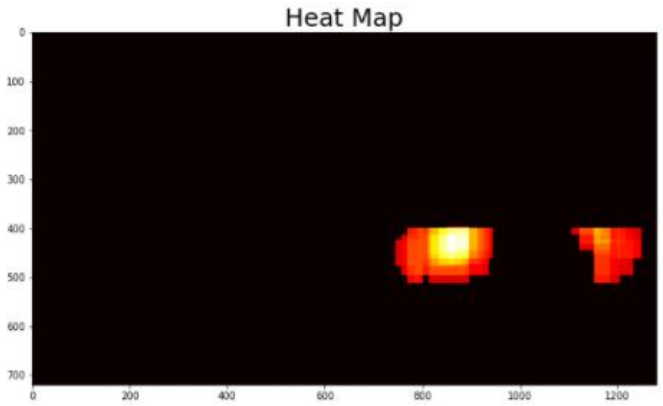
```
from collections import deque
history = deque(maxlen = 8)
...
history.append(current_heat_map)
...
```

This idea was implemented. A class was created to keep track of the history of the past 8 frames (as suggested). The sum of the 8 frames were used to create a heat map. Should the sum of nonzero area not exceed a certain value, it is then equated to zero. However, the information is still stored in the array, which means should the next 8 frame “agree” with the previously rejected frame, the sum of these will eventually exceed the threshold, thereby allowing a square to be drawn around it:

```
def heatmap_thresh(self, draw_img, bbox_list, threshold):
    heatmap = np.zeros_like(draw_img[:, :, 0]).astype(np.float)

    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1
    # "trim" edges
    heatmap[heatmap<1] = 0
    self.heat_images.append(heatmap)
    self.heatmap = np.sum(np.array(self.heat_images), axis=0)
    self.heatmap[self.heatmap <= threshold] = 0
```

In addition to this filter, which utilizes information of neighboring frames, a “trim filter” was also used to rid of weak area in the heatmap of any single frame (these would mostly be on the edges):



Notice how the black car has an oversized blue box, which was represented by the less hot area around the hot area. The threshold of the trim filter was left at 1, since the edges are relatively unstable and the overlap of these area sometimes help prevent false positive by boosting the overall heat from a true positive, allowing the first filter to more easily filter out the false positives (of course, this statement is only valid based on the assumption that false positives from a single frame rarely pile onto each other).

Video Revision:

A total of 30 videos were produce with different combinations of:

- Ystart, ystop, and scale
- Average filter threshold
- Single frame heat filter threshold
- Cells per step
- Color space for HOG features extractions

Video	**Comments**
project_video_processed	A preliminary trial with all the aforementioned functions implemented. Lots of false positives
Project_video_processed10	Raised the threshold for frame avg filter. Fewer false positives, but also accompanied by false negatives
Project_video_processed11	Moved around the ystart and ystop, as well as scales. Did not show much improvements
Project_video_processed12	“edge filter” was lowered on single frames, but frame avg filter threshold was raised
Project_video_processed13	Classifier retrained with C=0.0001
Project_video_processed14	Decreased frame avg threshold
Project_video_processed15	Decreased frame avg threshold
Project_video_processed16	Decreased frame avg threshold
Project_video_processed17	Decreased frame avg threshold