Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. 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.

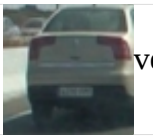
You're reading the pdf file. This is it!

Histogram of Oriented Gradients (HOG)

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

A. The code for this step is contained in the ***lesson_functions.py*** (first few lines 6~30).I started by reading in all the vehicle and non-vehicle images in function createSVC().

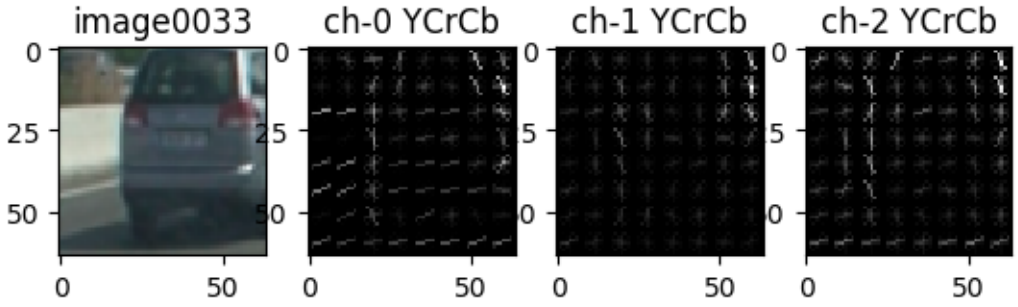
i. Here is an example of one of each of the vehicle and non-vehicle classes:

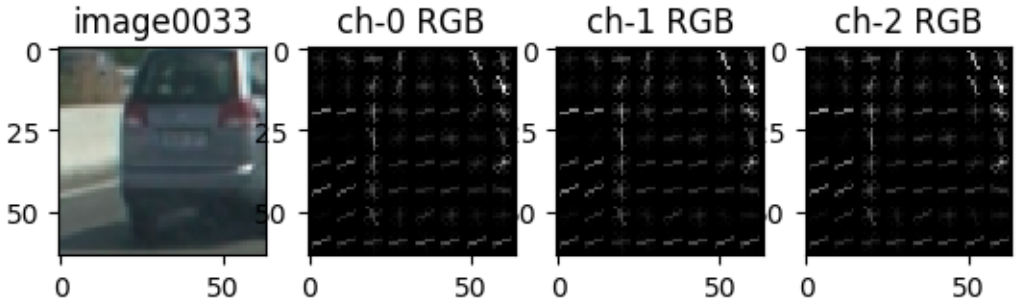
a)  non-vehicle  vehicles

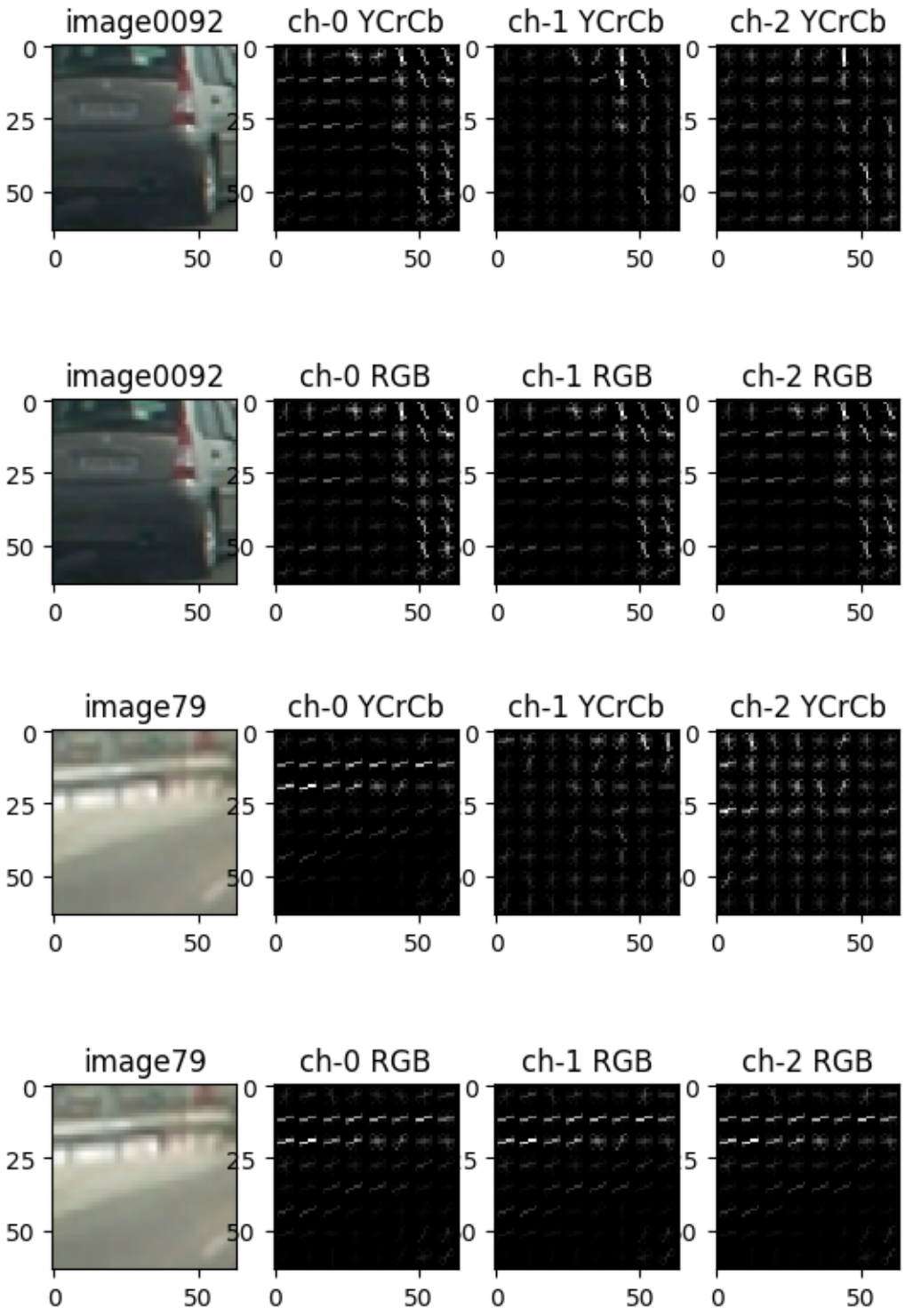
- In the code simple glob was used to pull in all the data, where:
  - # of cars example = 8,792 # of not cars are = 8,968
  - from that about 20% of the data were set aside for testing
  - After training Using: 9 orientations 8 pixels per cell and 2 cells per block
  - Test Accuracy of SVC = 0.9775, which is not TOO bad using RGB.

- I then, explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.
- Here is an example using theYcrCb and RGB color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

  The result is quite visually demonstrative. Among tried YcrCB clearly outperforms other measures. The 3 channels were distinctive when compared to RGB. This is not quite reflective from just error statistics, but, when using all available color channels, this should likely reflect less errors.
- The images below are some samples bulk generated by colorExplr.py. The outputs were partially saved into the directory ***output_images.***







Using: 9 orientations 8 pixels per cell and 2 cells per block with YcrCB reflects that assumption with Test Accuracy of SVC = 0.9879, which is better than 0.9775 from RGB.

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

I tried various combinations of parameters and looked at both the test accuracy and resulting plots, but settled on

- color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
- orient = 9 # HOG orientations
- pix_per_cell = 8 # HOG pixels per cell
- cell_per_block = 2 # HOG cells per block
- hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
- spatial_size = (16, 16) # Spatial binning dimensions
- hist_bins = 16   # Number of histogram bins
- spatial_feat = True # Spatial features on or off
- hist_feat = True # Histogram features on or off
- hog_feat = True # HOG features on or off.

It’s hard to improve upon:

Using: 9 orientations 8 pixels per cell and 2 cells per block

Feature vector length: 6108

16.08 Seconds to train SVC...

Test Accuracy of SVC = 0.9901

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 using function createSVC() in line 132 (all are in searchClassify.py)

- Intial features were extracted from test data images using extractFeatures() function but the work was really performed in single_img_features() function.
- scaled_X = X_scaler.transform(X) was used to normalize the features.
- The combinations were derived as in above (#2).
- To reduce the future computation, svcModel & X_scaler were pickled as below:
  - with open('./svcModel.pkl', 'wb') as fp: pickle.dump(svc,fp)
  - with open('./X_scaler.pkl', 'wb') as fw: pickle.dump(X_scaler, fw)

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?

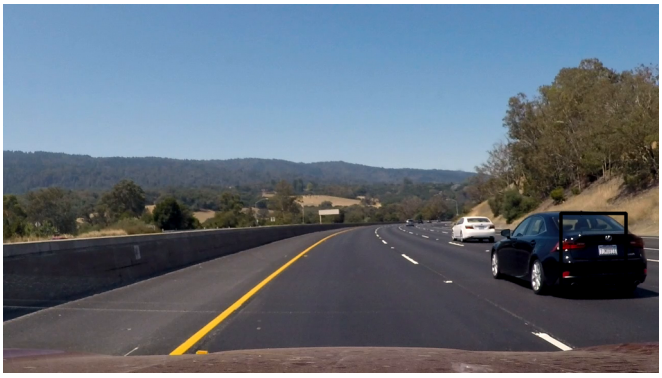
Inorder to tackle this part I need to:

- Apply grid & sliding windows
  - The grid needs to be setup in such a way that certain areas are masked out to reduce false positive - such as, sky, bonnet, over the barrier, & far right (cops may hide there but...).
  - The sliding windows size must reflect the perspective. Closer to the camera, the cars will be larger and require larger windows. And, father will require smaller windows.
  - I have chosen this by using 3 layered search Windows with 3 elements, which are in config.py file as a list of np.array(s) global variable, as shown below.
    The config.py also contains Original Global variables to control & others.
    i. ((xmin, xMax), (ymin, yMax))
    ii. number the coordinates which are relative to the image size, (i.e. between 0 and 1)
    iii. # search widnow size
    GVsearchWindows = [(np.array([[0.0,1.0], [0.5, 1.0]]), 64),
    (np.array([[0.0,1.0], [0.5, 1.0]]), 96),
    (np.array([[0.0,1.0], [0.5, 1.0]]), 128),]

- Perform Feature Extraction & Prediction
  - Features were extracted after looping through the above search windows and getting list of search windows called searchWinList. Not much changes were needed to the lecture example
  - The single_img_features function then takes image and passes back features with appropriate parameters placed in config.py as a scalar. The classifier get this features and makes prediction based via clf.prediction and returns the (hot) windows.
- Heatmap overlay on the above
  - The function add_head is applied on the hot_windows from above
- Eliminate false positive & duplicate detections
  - The function label is applied to above Heatmap
- Draw vehicle position boxes based on above
  - finnImg = draw_labeled_bboxes produces final image to be returned, but this is converted to 0~255 space prior to returning it.

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?

Ultimately I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:



Video Implementation

1. 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.)

Here's a link to my video result

2. 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.

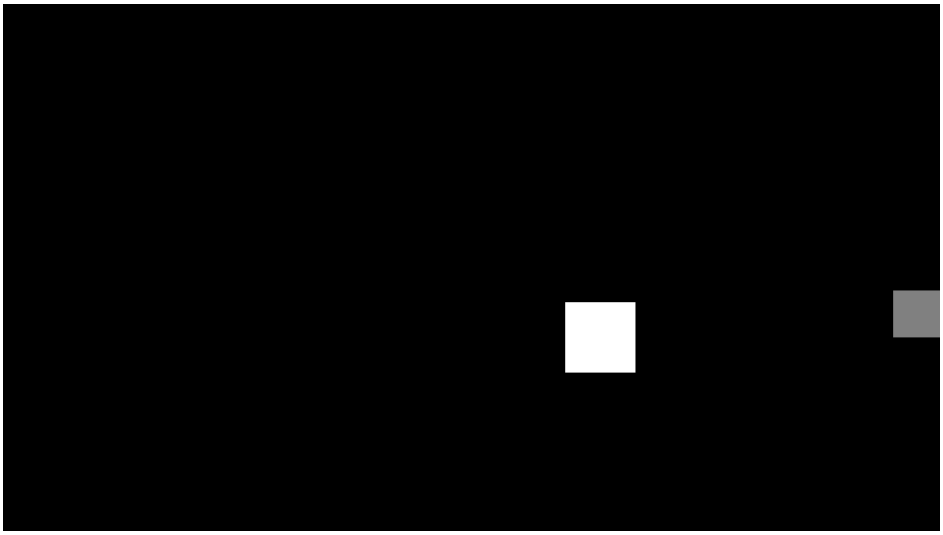
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used

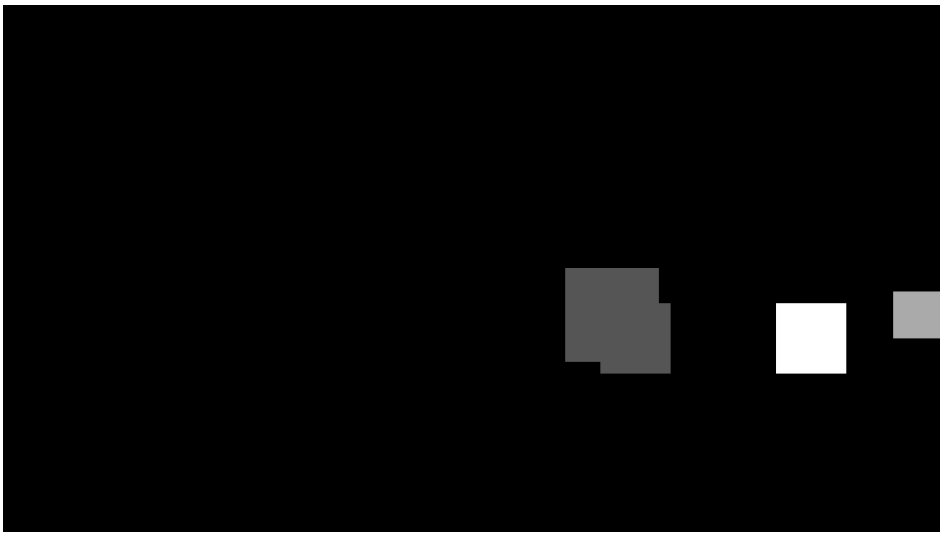
scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected. Here's an example result showing the heatmap from a series of frames of video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of video:

Here are some frames and their corresponding heatmaps:

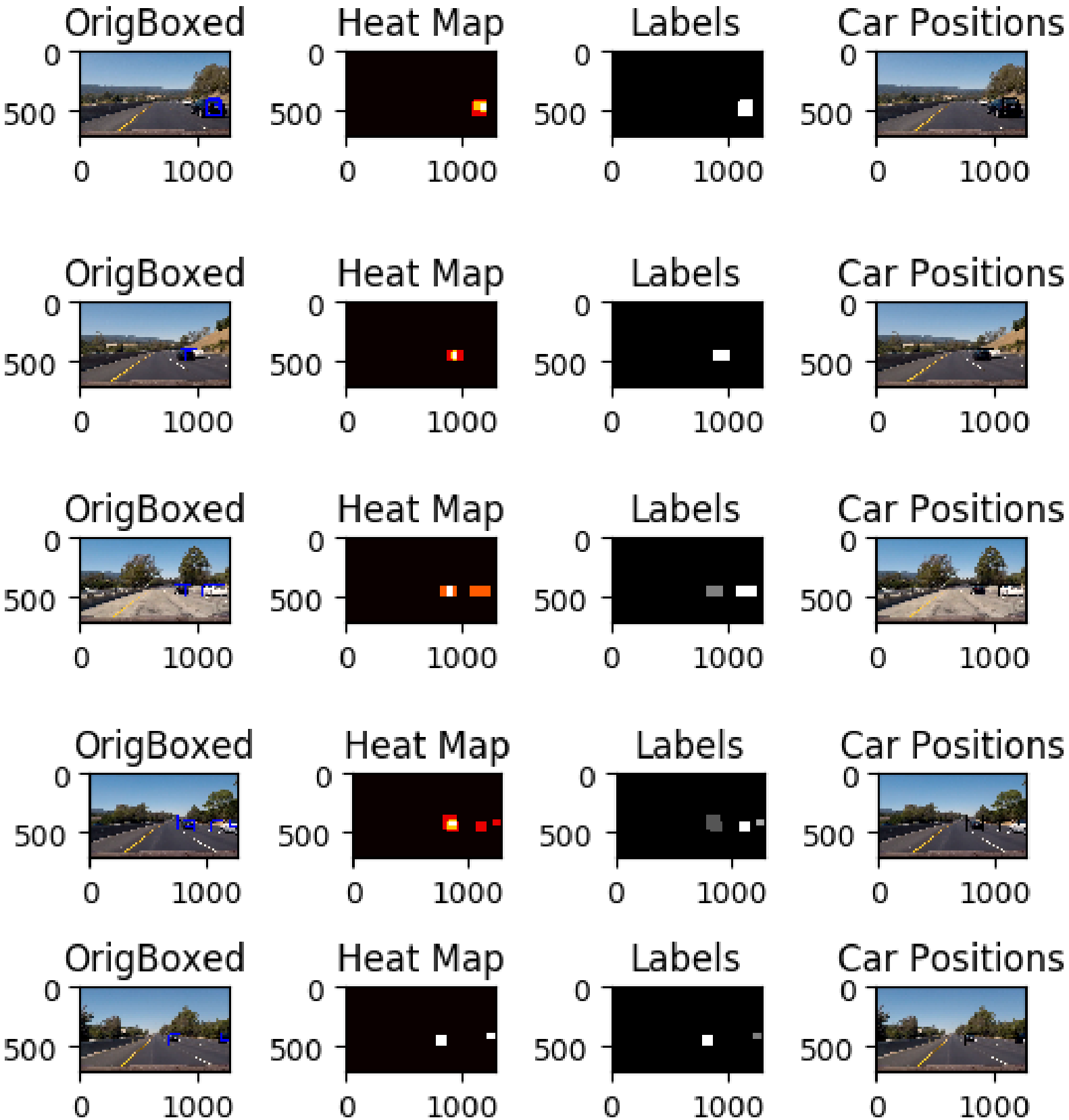
(see integrated images on next to the last section)

Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from few frames:





Here’s the final integrated Pics of all of Frames



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?

I seen better implementations where the heat thresholds were used and memory of videos were incorporated to make sure cars do not jump off or disappear from the frame. I certainly would like to add that to my mix, sometimes cars bound boxes jump off in and out.
Additionally, some also used watershed to make sure cars are near each other to distinguish as two cars rather then combine them together as a single bound box.
Thankfully, my SVC had high degree of reliability, so I was able to get away with a lot. Obviously, this could not be trusted in some weather conditions or poor visibility during the night.
SVC is nice but Random Forest & other techniques like CNN could be interesting to attempt as well, but would be costly in terms of computational speed, except when GPU could be utilized.

The individual pictures of the 6 frames generated are in outputs file: (Had a bit of problem generating documentation dual pic exactly)

- output_images/0_OrigBoxed.jpg
- output_images/0_Heat Map.jpg
- output_images/0_Labels.jpg
- output_images/0_Car Positions.jpg
- output_images/0_ALLout.png
- output_images/1_OrigBoxed.jpg
- output_images/1_Heat Map.jpg
- output_images/1_Labels.jpg
- ...
- output_images/4_ALLout.png
- output_images/5_OrigBoxed.jpg
- output_images/5_Heat Map.jpg
- output_images/5_Labels.jpg
- output_images/5_Car Positions.jpg
- output_images/5_ALLout.png