Vehicle Detection Project

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.

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.

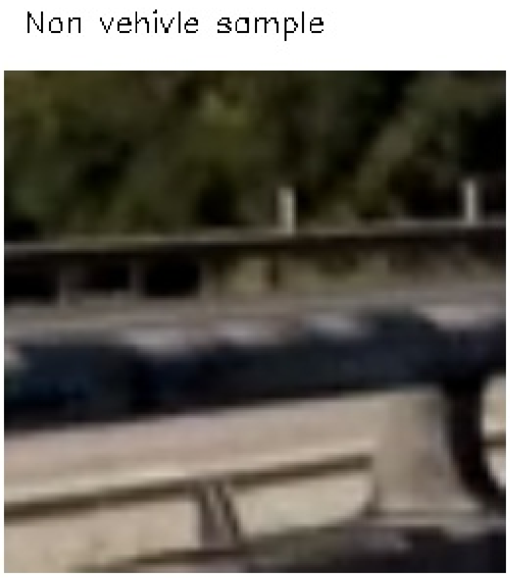
You're reading it!

Histogram of Oriented Gradients (HOG)

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

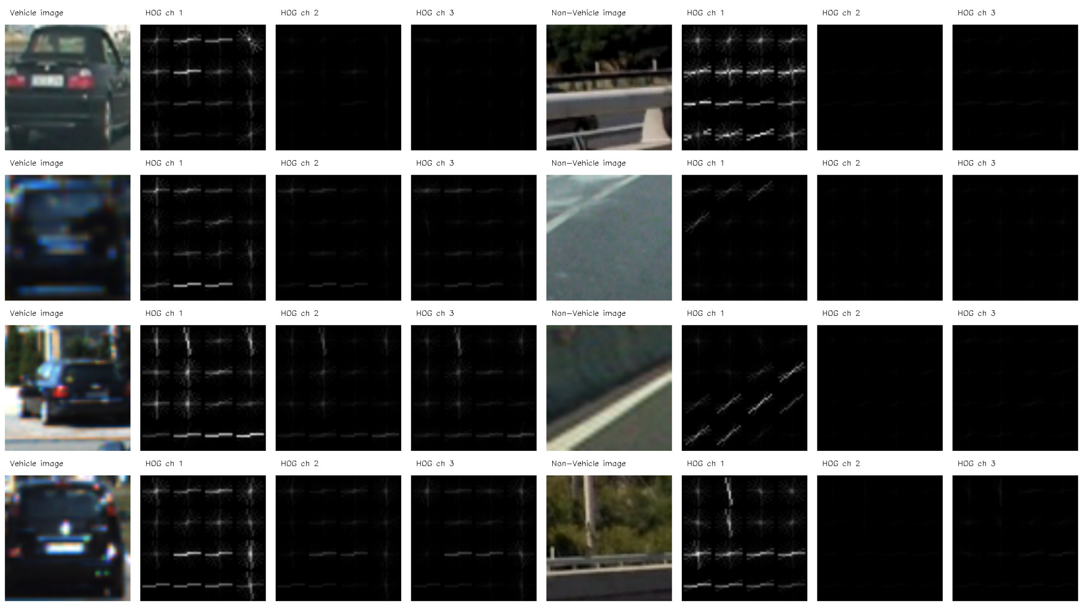
The code for this step is in utils/vehicle_functions.py file in extract_features_image function, lines 125 through 162. It's possible to use HOG of all channels or single channel of image (VehicleDetectorOptions.hog_channel class line 11). I use skimage.feature.hog in get_hog_features function lines 96 through 104 to compute HOG and other functions use this function.

I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:



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 the YUV color space and HOG parameters of orientations=11, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):



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

I tried various combinations of parameters and then trained the svm model for them as you can see in vehicle_detection.py file in hyper_train function, lines 73 through 93. Results can be seen in hyper-train-results.csv file and here you can see top 10 results ordered by test accuracy:

test_acc	color_space	hog_orient	hog_pix_per_cell	hog_cell_per_block	hog_channels	s
0.990811237	YUV	11	16	2	ALL	1
0.988973484	LUV	11	16	2	ALL	1

0.987660803	YUV	7	16	2	ALL	1
0.987135731	LUV	9	16	2	ALL	1
0.985823051	LUV	11	8	2	ALL	1
0.985823051	YUV	9	16	2	ALL	1
0.985560515	LUV	9	8	2	ALL	1
0.985297978	YUV	11	8	2	ALL	1
0.984772906	LUV	7	16	2	ALL	1
0.984772906	LUV	7	8	2	ALL	1

I chose the best one with 11 as HOG orientations, all channels for HOG and YUV as color space and 16 as pixel per cell for HOG.

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).

The code for this step is in utils/vehicle_detector.py file in train method of VehicleDetector class, lines 25 through 75. Selection of HOG features and color features can be done using VehicleDetectorOptions class in utils/vehicle_functions file. I did these steps in train method:

1. Find all train images in vehicle and non-vehicle directories (Lines 27-32)
2. Extract features for the train images (Lines 35-38)
3. Stack vehicle and non-vehicle features in one numpy array (Line 40)
4. Normalize the features using StandardScaler (Lines 43-48)
5. Build label array with ones and zeros for vehicle and non-vehicle features respectively (Line 50)
6. Shuffle features and labels and then split them into train and test features and labels (Lines 53-59)
7. Build an instance of LinearSVC model and fit it on the train features. I used default model parameters (Lines 65-68)
8. Test the model on test features (Line 73-74)

Sliding Window Search

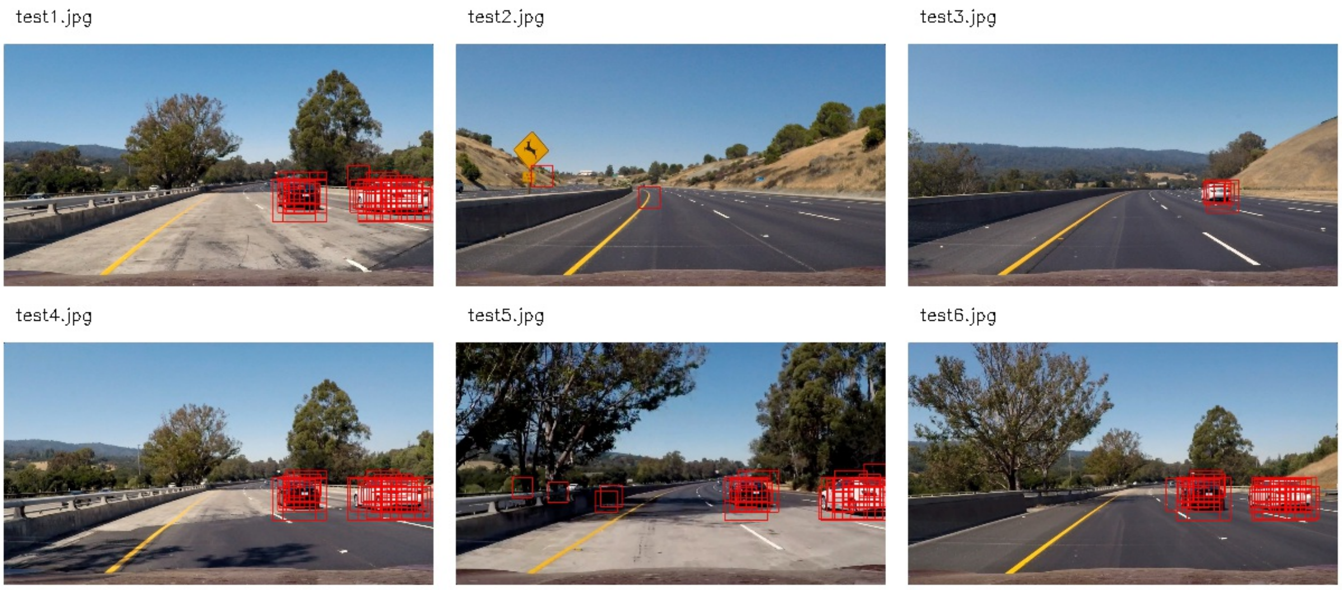
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for this step is in utils/vehicle_detector.py file in find method of VehicleDetector class, lines 91 through 137. I used down-scaled images in different scales (0.95, 1, 1.5, 2.0) and search with fix window size(64 pixels). The code for finding cars on a scaled image is in utils/vehicle_functions.py file in find_cars function, lines 179 through 262.

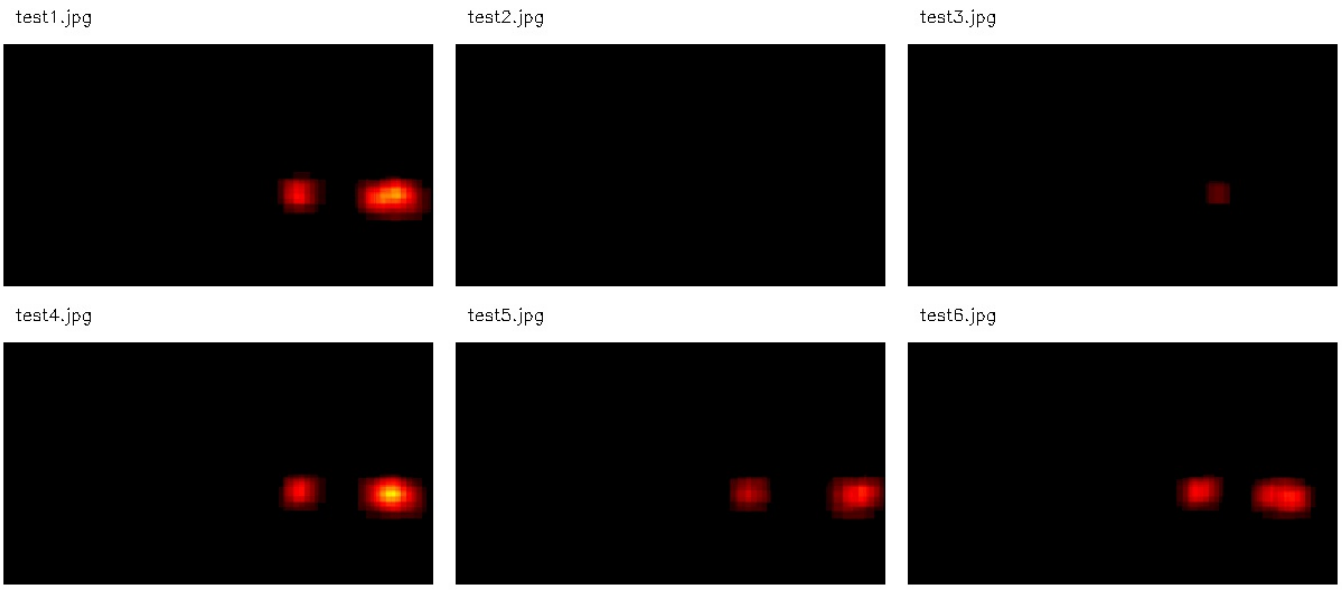
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on four down-scaled images using YUV color space, all channels HOG features plus histograms of color and spatial colors in the feature vector, which provided a nice result. Here are pipline example images:

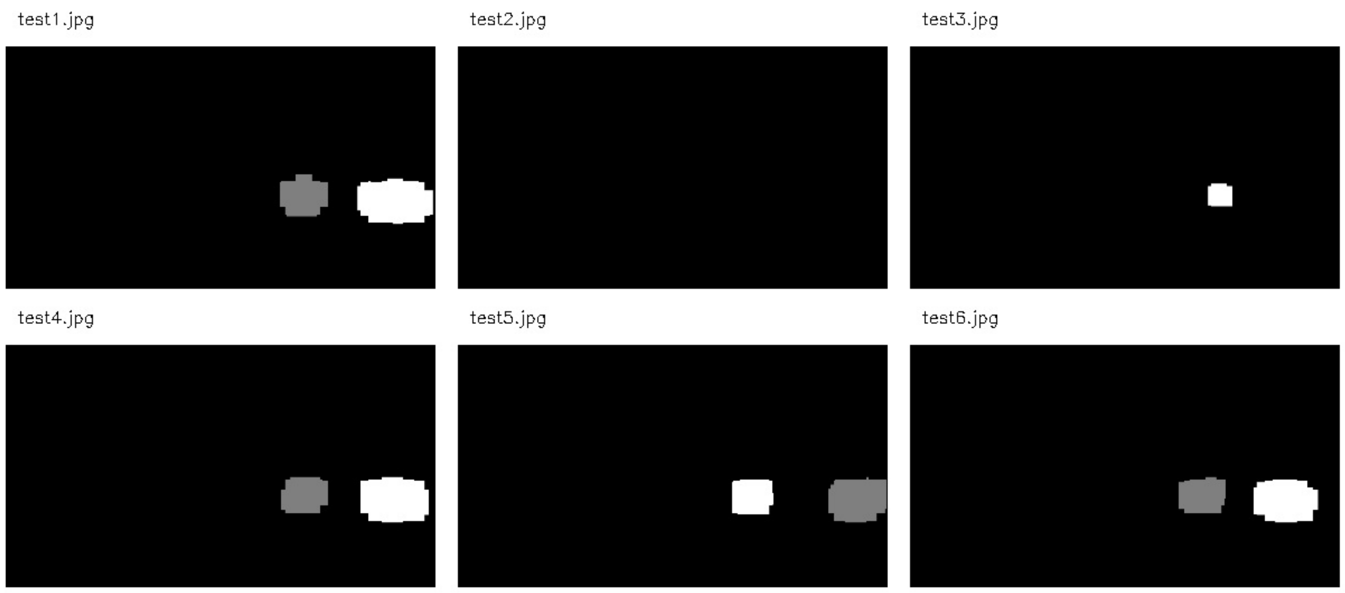
1. Find cars in multi-scale image (utils/vehicle_detector.py, VehicleDetector.find, lines 94-108)



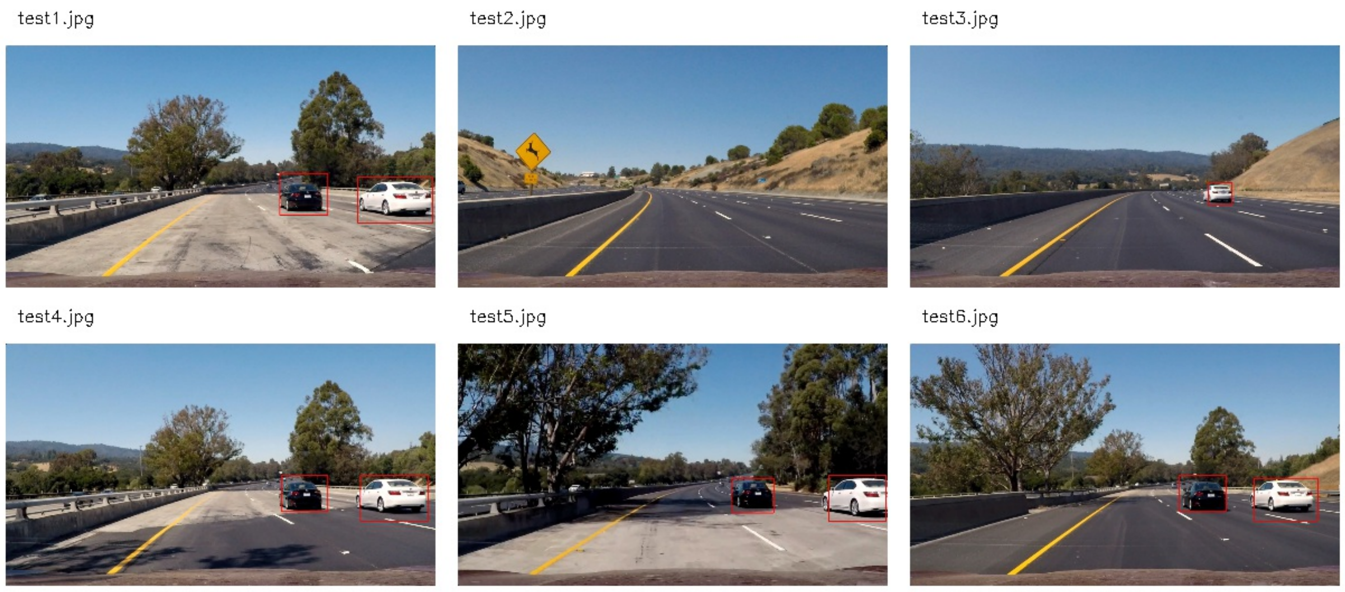
2. Build heatmap (utils/vehicle_detector.py, VehicleDetector.find, lines 110-120)



3. Find connected components using skimage.measure.label (utils/vehicle_detector.py, VehicleDetector.find, lines 120-127)



4. Find final bounding box (utils/vehicle_detector.py, VehicleDetector.find, lines 128-137)



Video Implementation

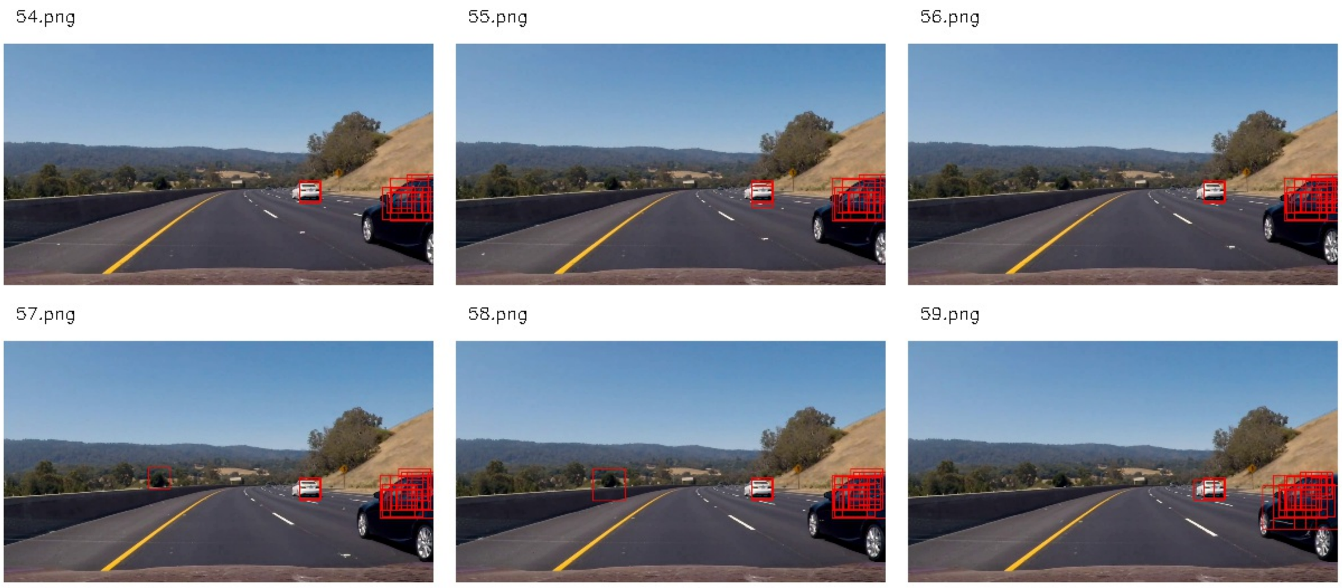
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

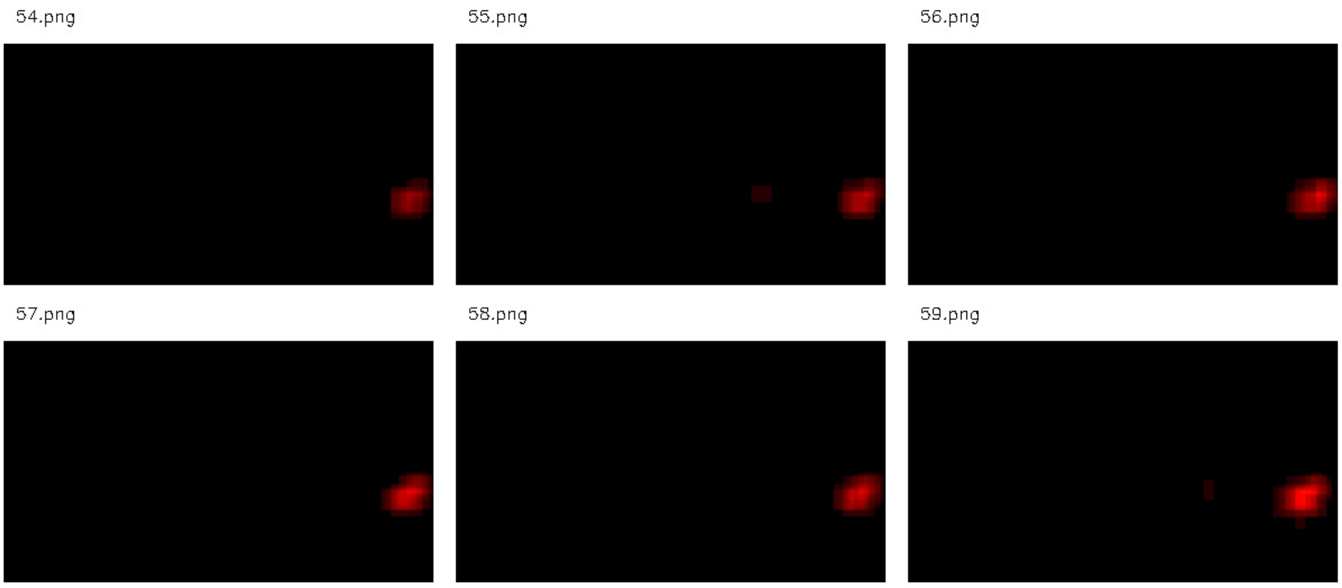
Here's a link to my video result

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.

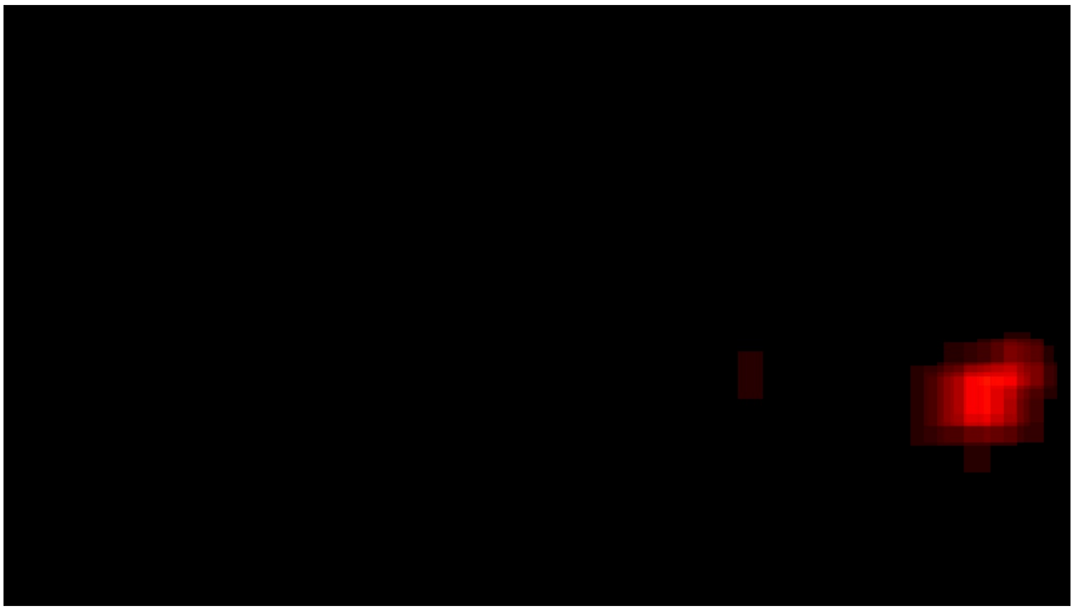
The code for this step is in utils/vehicle_tracker.py file in VehicleTracker class. I recorded the bounding boxes of positive detections in each frame of the video. then for a new frame I used previous frames' bounding boxes and build a heat map. I then use scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then found the number of bounding boxes that made each blob (max_nu_in_agg_label) and filter the blobs using thresholding on this number.

Here are six frames and their corresponding heatmaps:

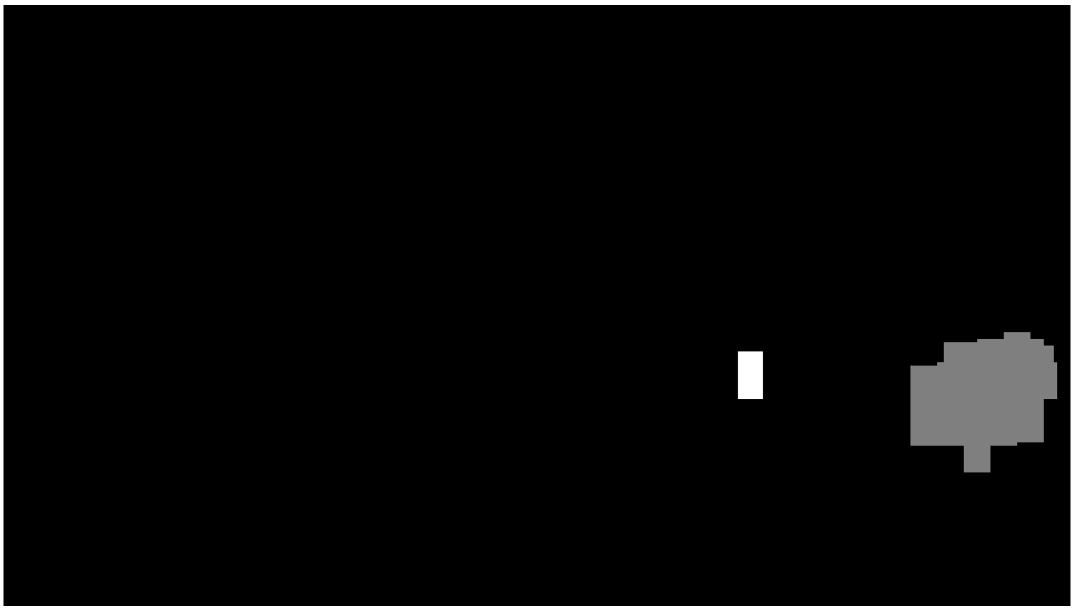




Here is the integrated heatmap from all six frames:



Here the resulting bounding boxes are drawn onto the last frame in the series:



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 used linear svm as classifier and spent lots of time on training the svm to learn the vehicle and non-vehicle classes properly but it did't go well enough. The problem of classifier I think was the lack of non-vehicle samples so I added some non-vehicle samples from videos of this project and the previous project and got good results.

I also found that YUV, YCrCb and LUV color spaces work better than RGB, HLS and HSV color spaces for this task.

The classifier I used and trained in this project still have problems identifying non-vehicle images and will fail on videos with different environment than this one.

I should work more on finding better features than those I used in this pipeline or even change the classifier to a convultional network based classifier. Furtheremore I should add more images to the dataset.