Aidos Utegulov

Project 5 Vehicle Detection

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.

Histogram of Oriented Gradients (HOG)

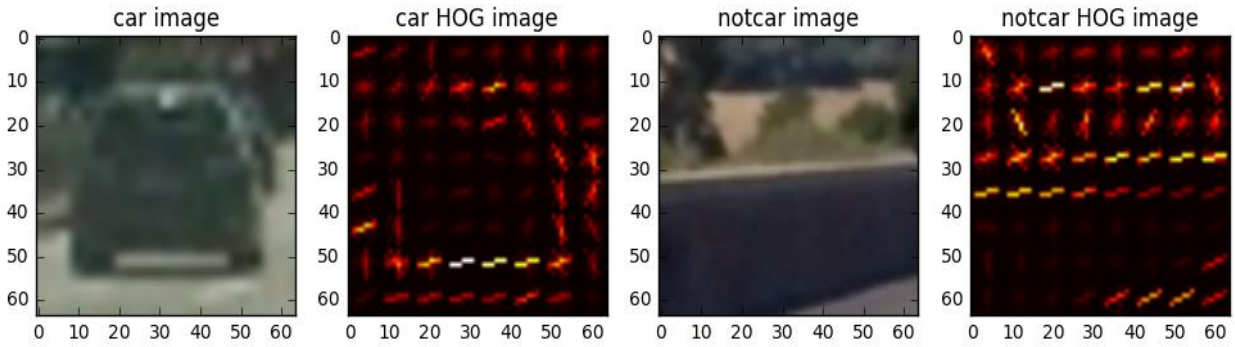
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first code cell of the Vehicle-Detection IPython notebook (referred to as "the notebook" hereafter).

I started by reading in a set of 2100 `vehicle` and `non-vehicle` images each (When I set the set size to 3000, my computer starts crashing). Training data were provided by Udacity – images from GTI vehicle database and Kitti vision benchmark suite. 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. The function get_hog_features() in the third code cell of the notebook (the one that contains the definitions of all functions studied in the lessons) performs the hog feature extraction using the skimage.feature.hog() function. The get_hog_features() function is also used by the functions extract_features() and single_img_features() (defined in the same third code cell of the notebook) to extract hog features along with histogram and spatial features for a single image.

Here is an example using the `RGB` color space and HOG parameters of `orientations=9`, `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. After changing the color_space to, consecutively, RGB, HSV, LUV, YUV, HLS, YCrCb, I found that YcrCb gives the best training accuracy – 99.6%. I did not touch the orient parameter, since, in their presentation of HOG, Dalal and Triggs show that setting orient to 9 gives best results, I set the spatial_size parameter to (32, 32), after trying (16, 16) and getting worse results. After that I set hist_bins to 32. I got varying accuracy from 95 to 97% when trying out the values of (16, 16) for spatial bins and 16 for hist_bins. Also, setting the hog_channel to 0 gives a slightly better result in the accuracy than 'ALL', for some reason. But mainly, I relied heavily on the recommendations given in the project walkthrough, the forums and the video from Dalal and Triggs when tuning the HOG parameters.

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 chose following parameters for identifying HOG features:

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimenstions
hist_bins = 32 # 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
```

I trained a linear SVC using 2100 samples of vehicle and not vehicle objects each. I then extracted features for both types of objects using extract_features() method (defined in code cell 3 of the notebook). I then stacked the features using np.vstack, scaled them with the help of StandardScaler().fit() method (code cell 5 in the notebook). I split the training and testing sets to 80% 20% respectively, and the ran the classifier. The accuracy of 99.5% was reached. I train the Linear Support Vector Classifier in code cell 5 of the notebook.

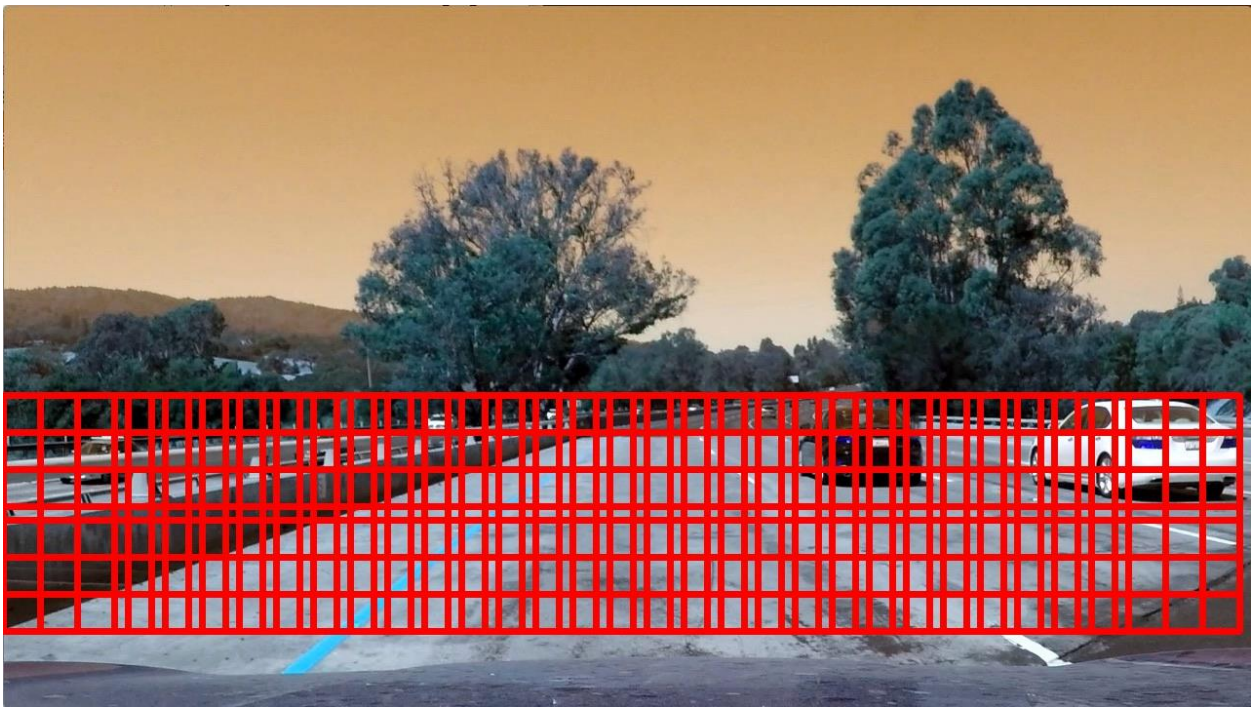
Sliding Window Search

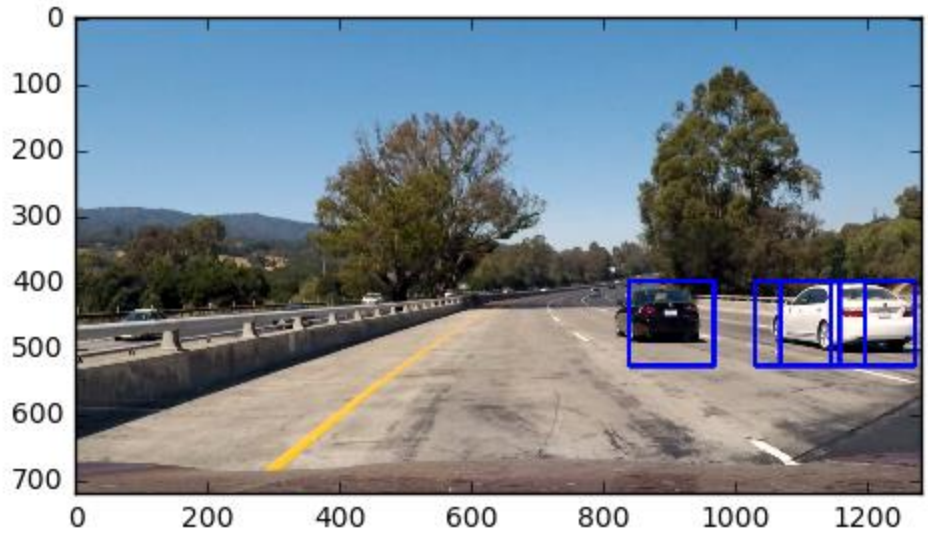
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The sliding window search is implemented with the help of two functions defined in the project intro lessons. Code cell 3 in the notebook contains the definitions of these functions. I then used these functions in code cell 6 of the notebook:

```
windows = slide_window(img, x_start_stop = [None, None], y_start_stop =
y_start_stop, xy_window = (96, 96),
                           xy_overlap = (overlap, overlap))
    #windows = windows1 + windows2
    hot_windows = search_windows(img, windows, svc, X_scaler, color_space
= color_space, spatial_size = spatial_size,
                                 hist_bins = hist_bins, orient = orient,
pix_per_cell = pix_per_cell, cell_per_block =
                                 cell_per_block, hog_channel = hog_channel,
spatial_feat = spatial_feat, hist_feat =
                                 hist_feat, hog_feat = hog_feat)
```

I set the area to search for windows like this: y_start_stop = [400, 656]. These are image height values to focus on during the search, since we don't really care about the horizon. The variable hot_windows in code cell 6 of the windows contains the output of search_windows() function, and I then draw the boxes on the test images using this variable and the draw_boxes function defined in code cell 3. Below are the images showing the result of calling slide_window() (cv2.imwrite() messed up the colors) and search_window() (I used matplotlib's savefig() function to save the output of search_window():



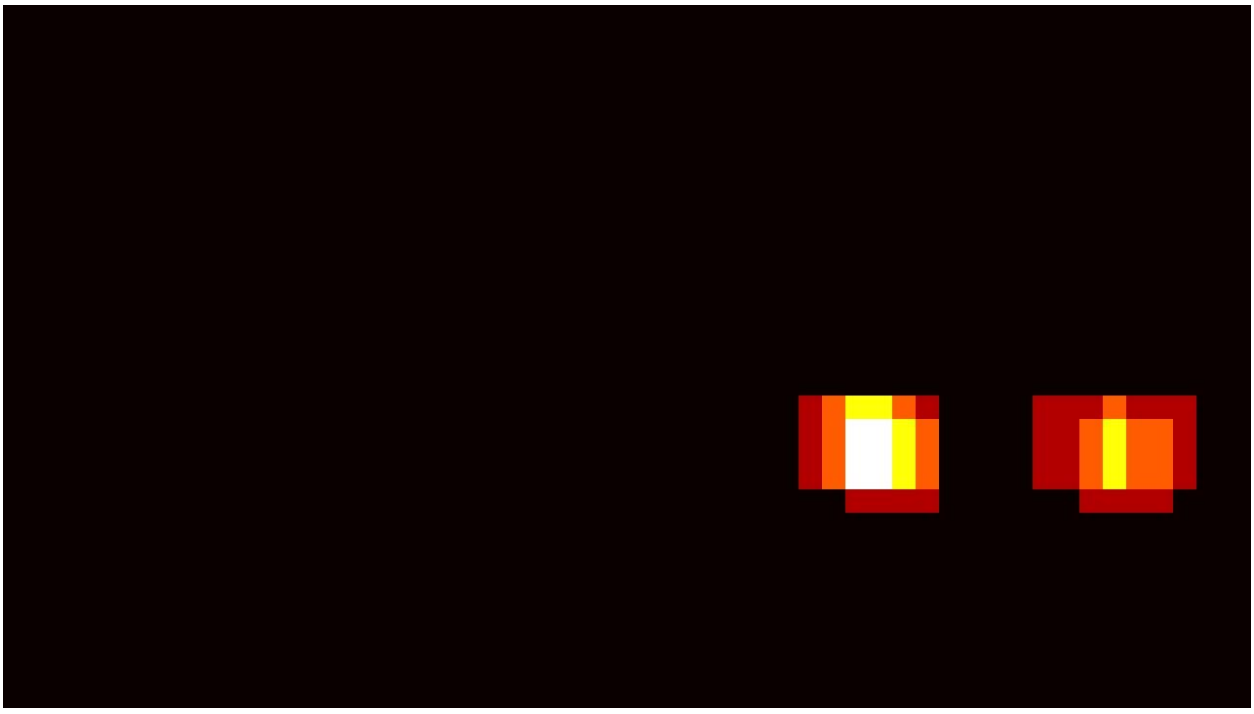


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

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. I then proceeded to normalize the feature vector using sklearn's StandardScaler() class, which provided a balanced set of training data to feed to the classifier. I split the dataset into training and test sets with the help sklearn.cross_validation.train_test_split and allocated 20% of the training data to the test set. I also tried handpicking the images and mixing the images from GTI database with images from Kitti.

I will first demonstrate the bounding boxes found by calling the function find_cars():

Then I apply heat to the area bounded by multiple boxes:



I then run the scipy.ndimage.measurements.label() function on the heatmap and draw the resulting labeled boxes using the draw_labeled_bboxes() function defined in code cell 10 of the notebook. As a result I get a single clean bounding box around each car:

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:

https://drive.google.com/open?id=0B3wm7W2ZfOg2aHhuRlFWX1lGbzg

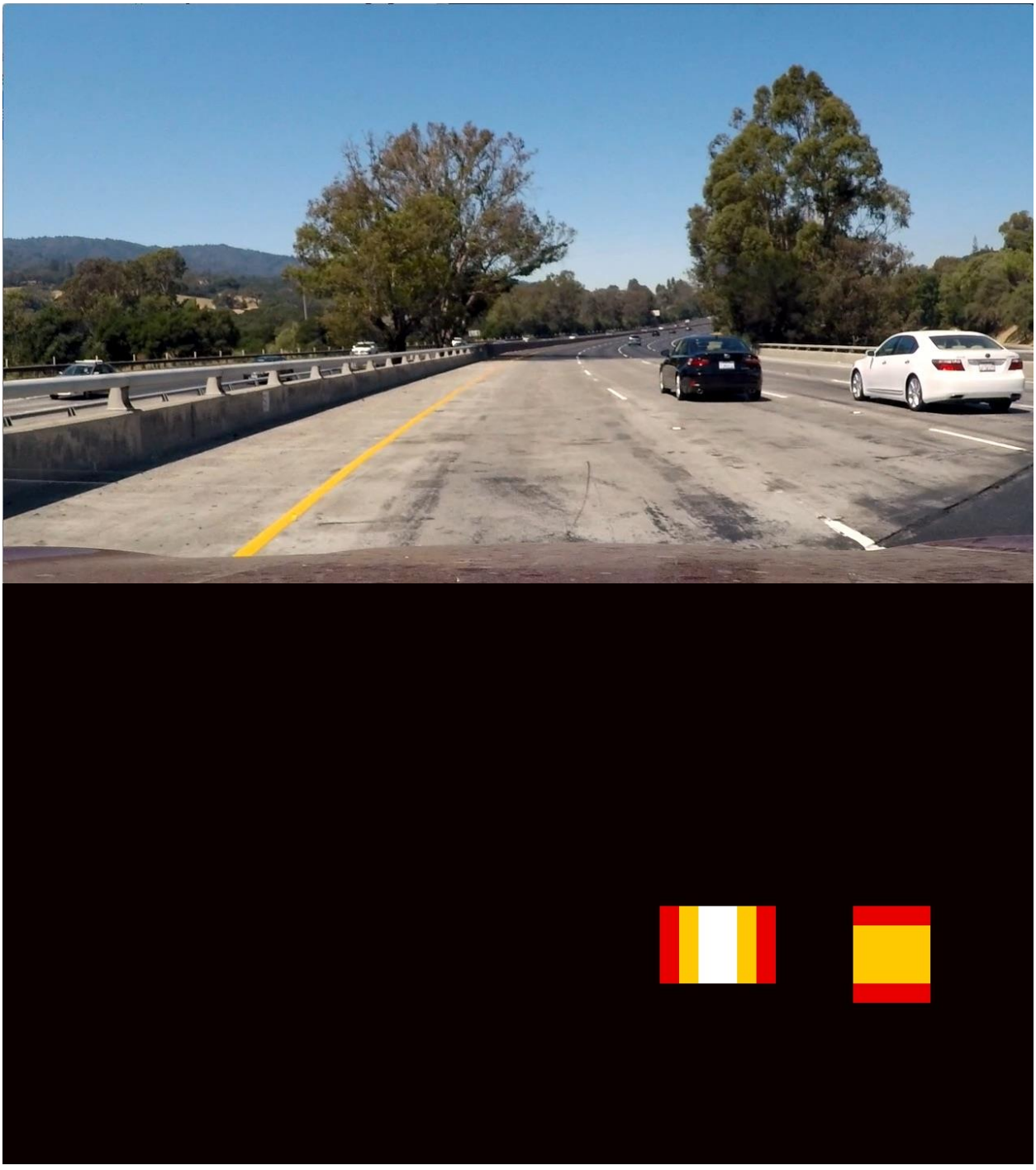
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

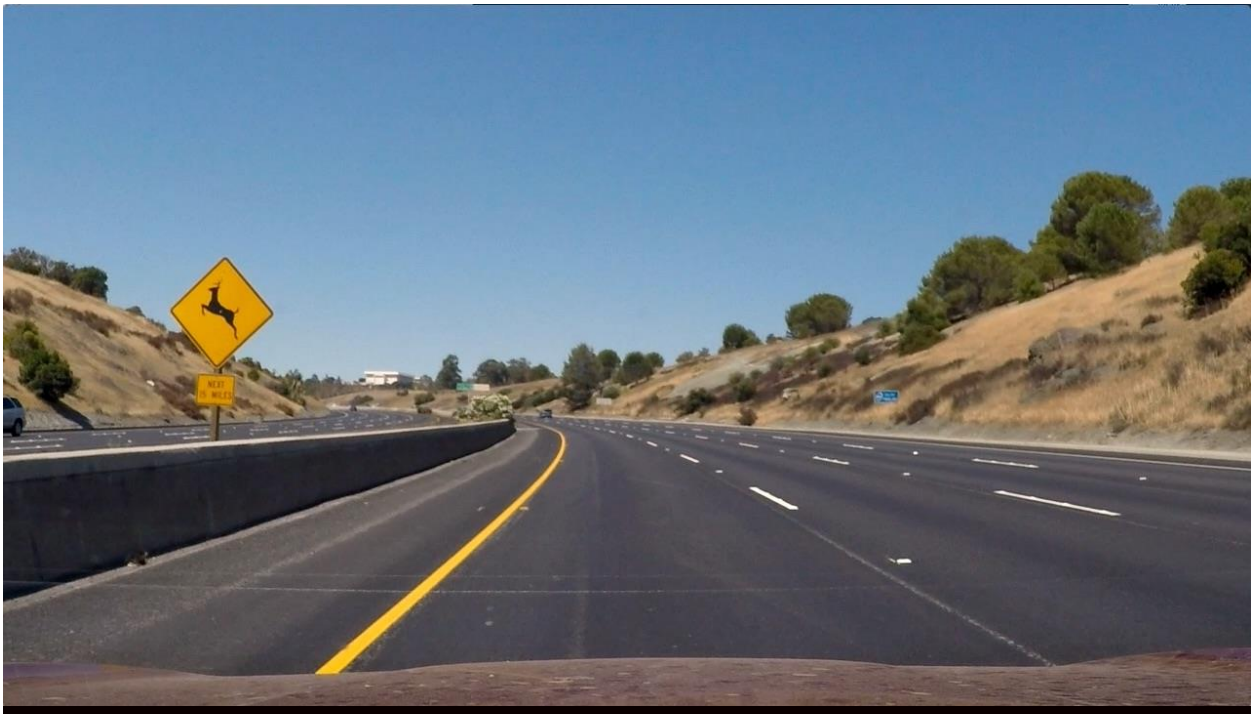
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected. Even then, after training the classifier on a sample of 2000 images of cars and notcars each, I had a lot of false positives from the left and the right of the road. I also got a series of false positives when the car passed a region

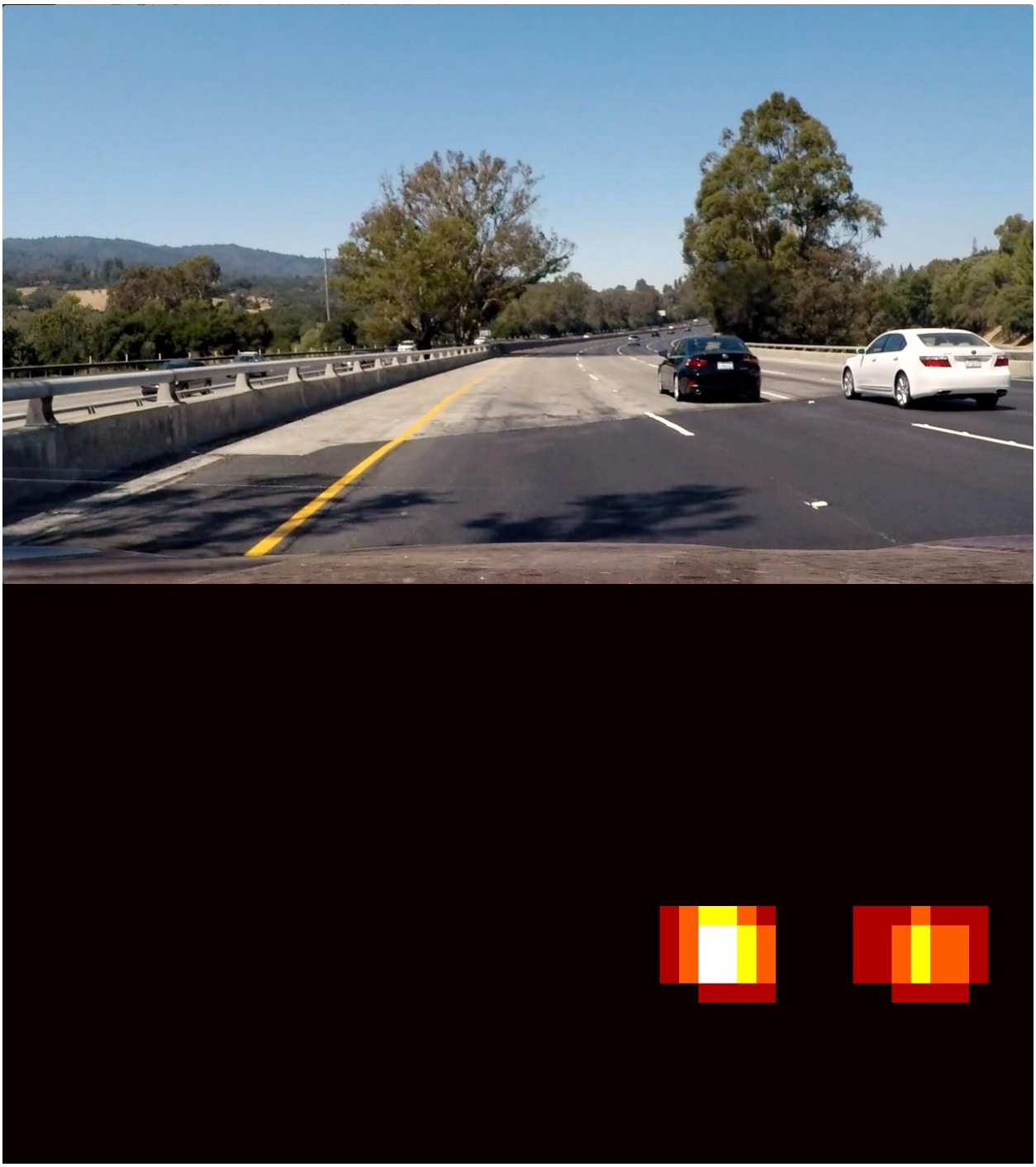
with a shade from a tree towards the end of the video. I then decided to manually handpick data for training, heavily focusing on the cars from the right and on road segments with shades from the tree. This helped a lot in eliminating the false positives from the shade by the tree. I still get a couple of false positives in the video, but the number decreased dramatically compared to the first attempt. The heatmap is applied in the process_image function defined in code cell 11 of the notebook, and I apply a threshold on heatmaps in the same function.

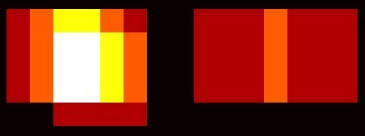
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

Here are six frames and their corresponding heatmaps:









Here is the output of `scipy.ndimage.measurements.label()` on 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?

The bounding boxes in the video still keep jumping even after trying to smooth them out from frame to frame. I kept 25 heatmaps and put a threshold of 15 in apply_threshold to keep around 60% of them. At some point in the video, the bounding box around the white car even disappears for a short period of time, and then comes back again. I suspect that I didn't feed enough training data to the classifier, especially from KITTI_extracted folder. Although I did keep almost all of the images from the GTI_Right folder, when the white car moves to a significant distance ahead of the camera from the right, the bounding box disappears for a second or two. Unfortunately, I could not train the Linear SVC on all images, because my computer kept crashing. I only took about 2100 samples each of vehicles and nonvehicles. I believe that the limited training dataset caused this issue.