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.

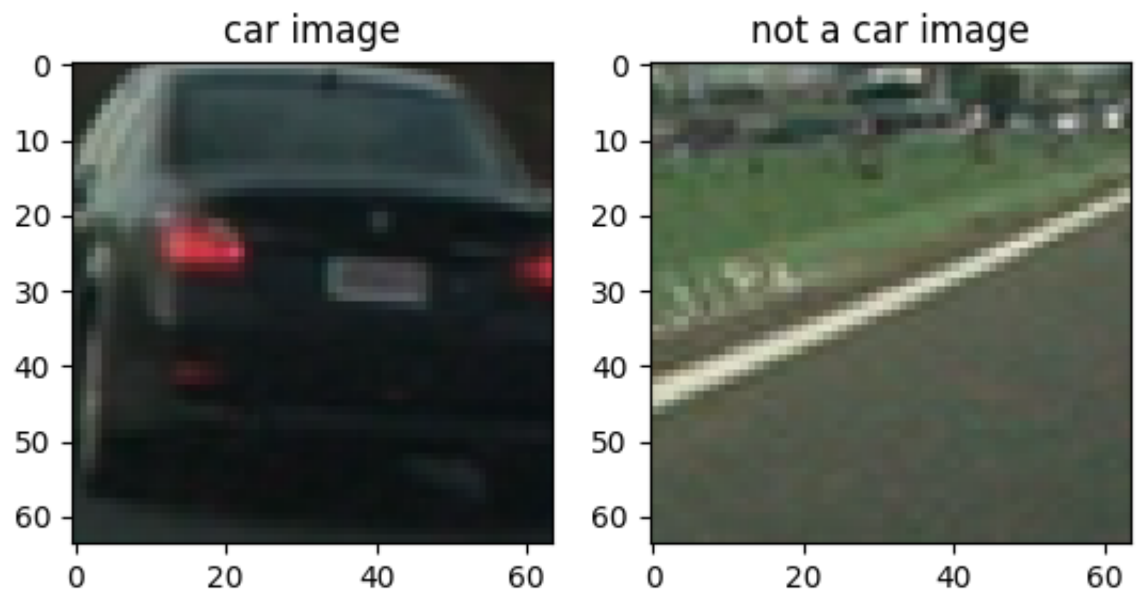
You're reading it!

Histogram of Oriented Gradients (HOG)

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

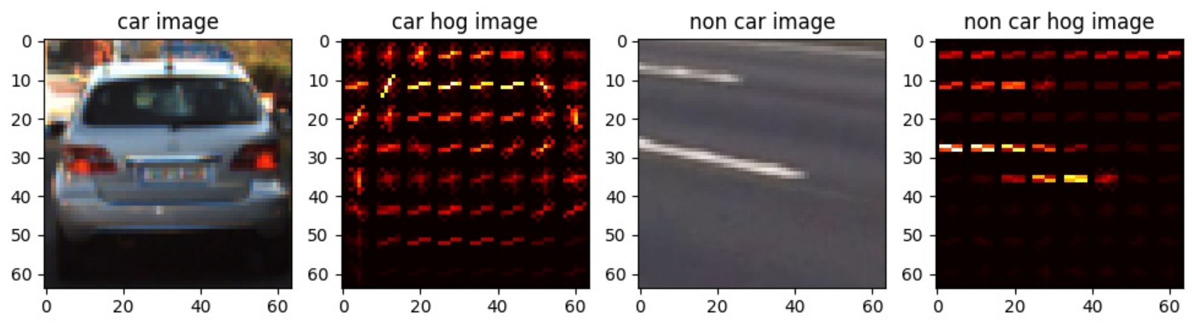
Please see utils.py file in the repository for a list of all the functions that I am using in this project, including the ones for feature extraction. HOG features are extracted by the get*hog*features function which is defined near line 21 in utils.py. I am using the sklearn.feature module function "hog" to compute the hog features for each sliding window in the image.

I started by reading in all the `vehicle` and `non-vehicle` images. Class TrainingData defined in training_data.py assembles the filenames of 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 (RGB, YCrCb, LUV, HLS and HSV) 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 `YCrCb` color space and HOG parameters of `orientations=6` , `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` . Note that the HOG features are only shown for the `Y` channel here.



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

I tried various combinations of parameters and decided to use the YCrCb color space as it seemed to have the one of the lowest correlations among the HOG features in the various channels. I used a combination of all the hog features (for each of the three channels in the color transformed image). I used number of orientation bins of 9, number of pixels per cell of 8 by 8 and number of cells per block as 2 by 2. Increasing the orientation bins may have improved the test performance but I was already getting an accuracy of 99% so I stopped at these numbers to keep the training/test computation times reasonably low.

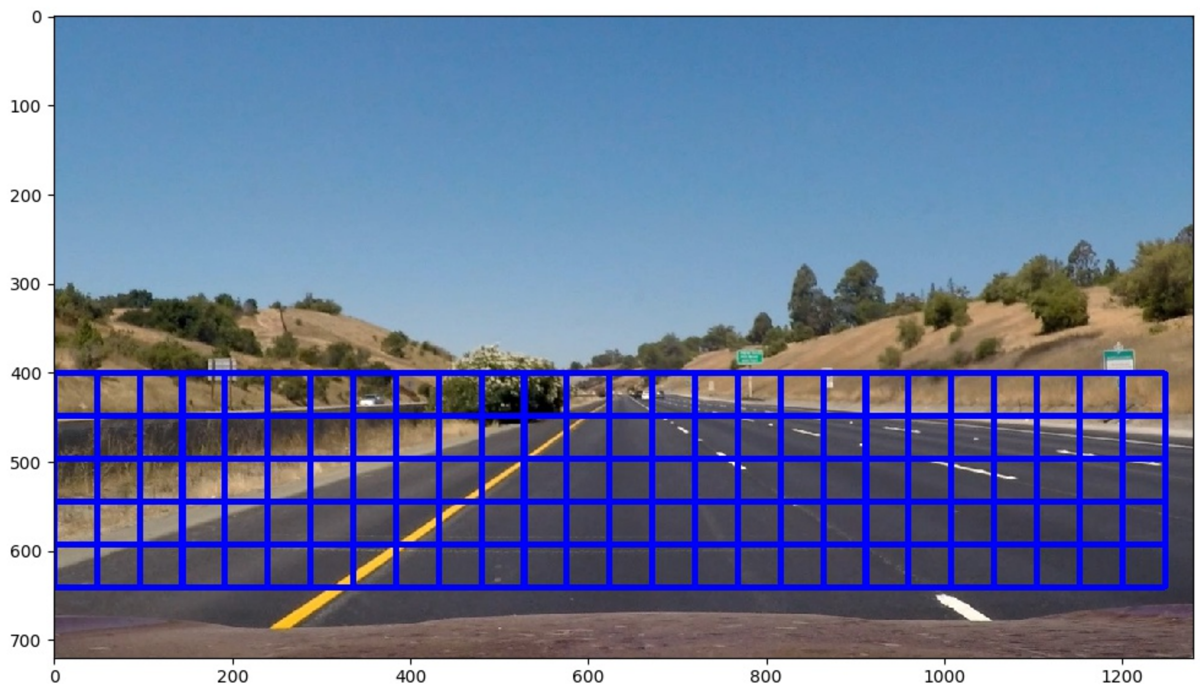
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM using hog features, color histogram features and spatial features (features generated by downsampling the image to 32 by 32 pixels). See `train.py` for the code that defines the train function and trains over the vehicle and non-vehicle classes. The `extract_features` functions from `utils.py` (which in turn uses the `single_img_features` function) is used to extract features from the training/test images. The feature extraction process first converts the image to the desired color space (`YCrCb` in this case). Then it computes the spatial features, histogram color features and hog features (for each channel) in succession and stacks them together into one long feature vector which is passed to the LinearSVM model for classification. For the chosen parameters, the feature vector is of length 4884 for each image that the classifier trains or tests on. I used the LinearSVC class from sklearn.svm package to train the SVM classifier on the extracted features from the training set. `10%` of the training set was randomly chosen to be used as test set for testing the training classifier. I used the `train_test_split` from sklearn.model_selection to randomly shuffle and split the training set.

Sliding Window Search

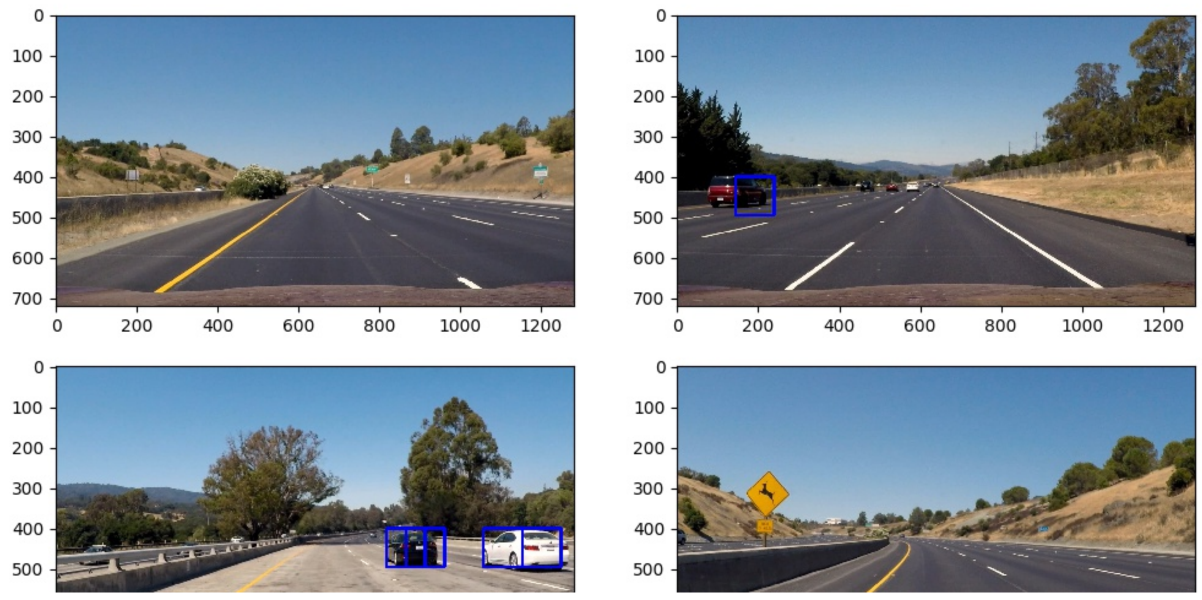
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

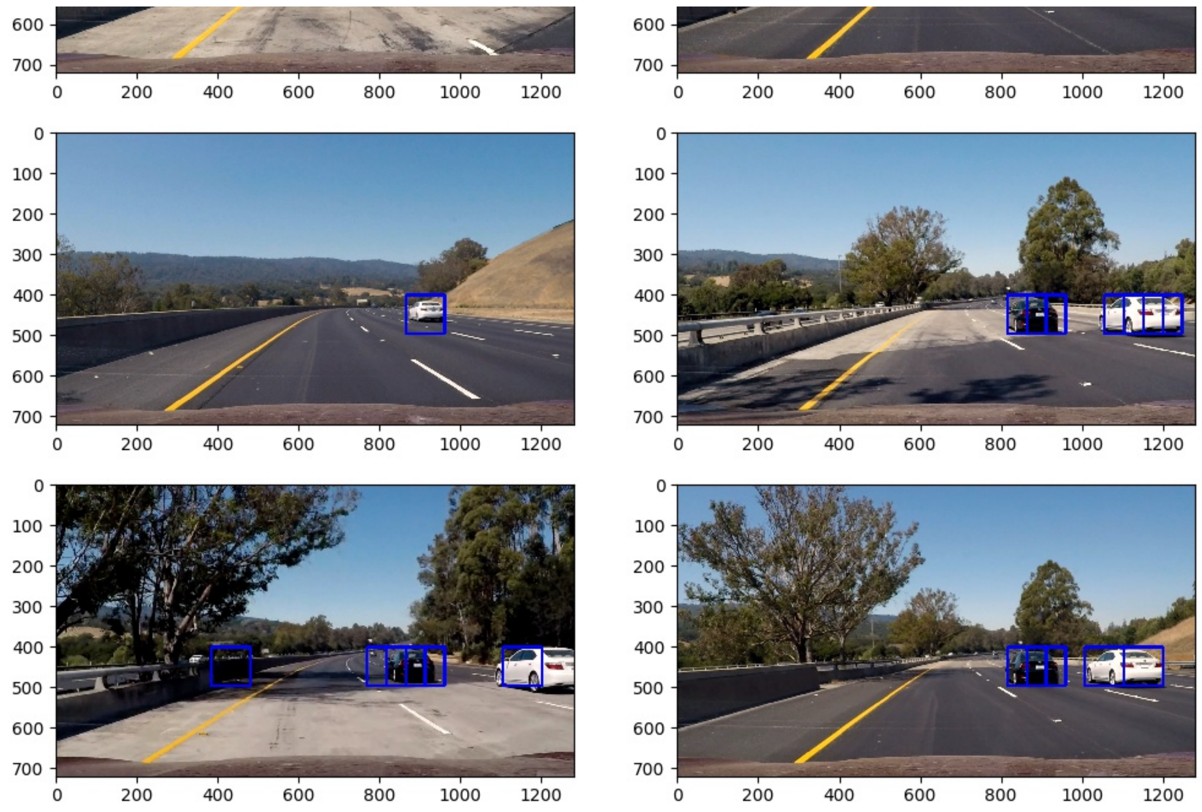
The sliding window search is implemented in the `find_cars` function defined in `utils.py`. I used a window size of 64 by 64 pixels limiting the search to a `y` indexes between 400 to 656 to avoid searching for cars outside the region of interest (which is the road area and not the sky/trees). Using values of `pix_per_cell` of 8 and `cell_per_block` of 2, the window overlap used was 48 pixels in each dimension, causing window overlap of `75%`. Such a high value of window overlap ensures a high detection rate as sliding windows traverse over the image, ensuring that we don't slide along a car without detecting 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 one scale 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.)

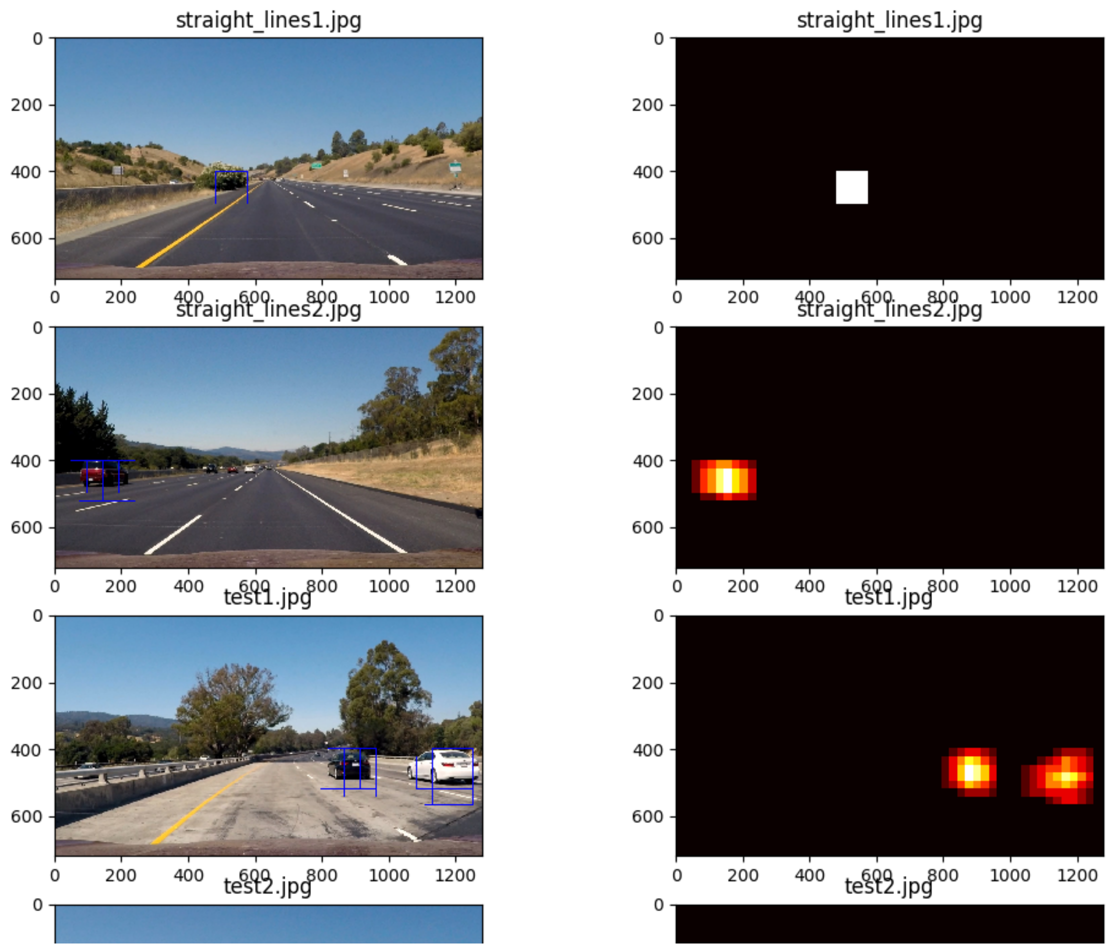
Here's a [link to my video result]()

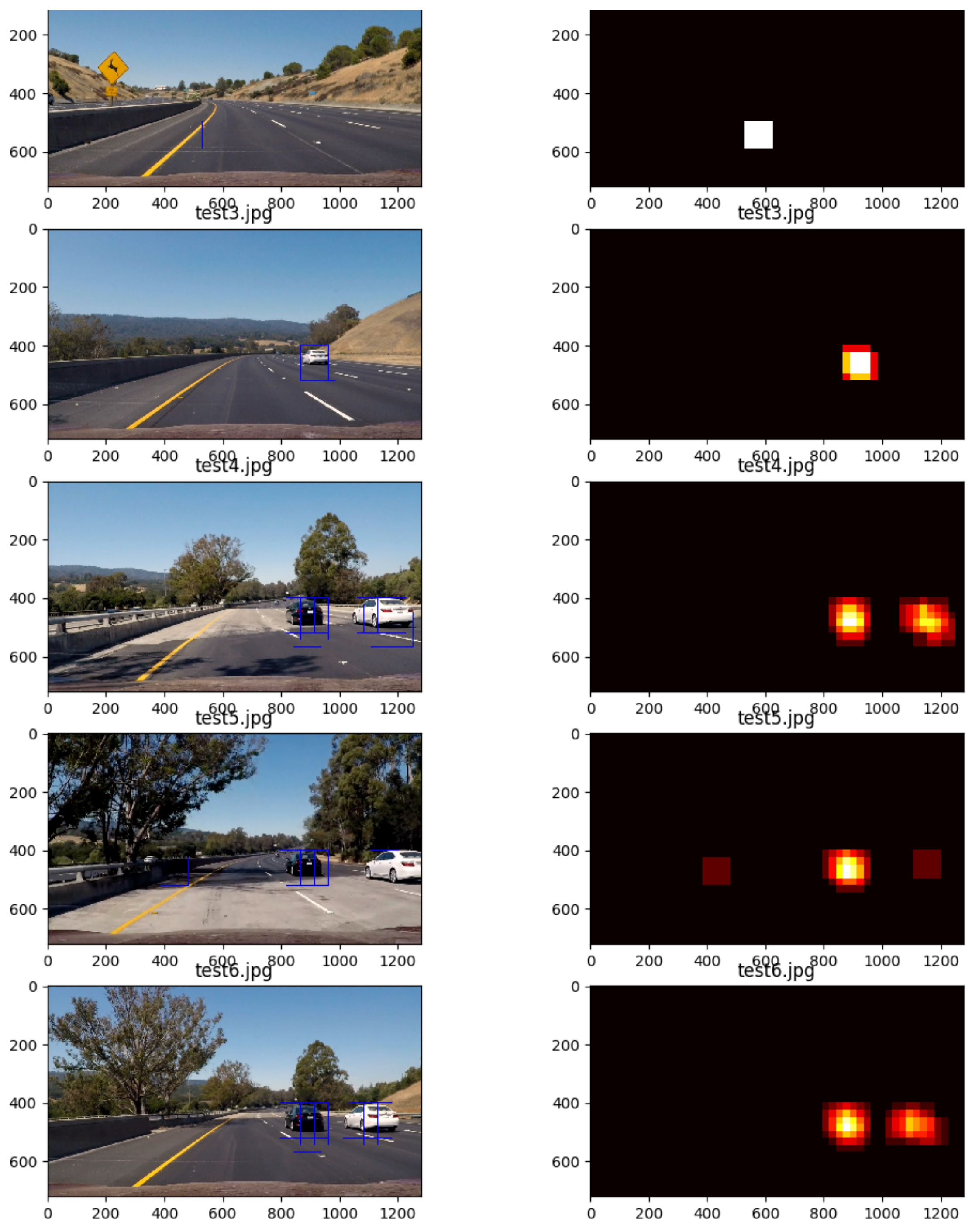
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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.

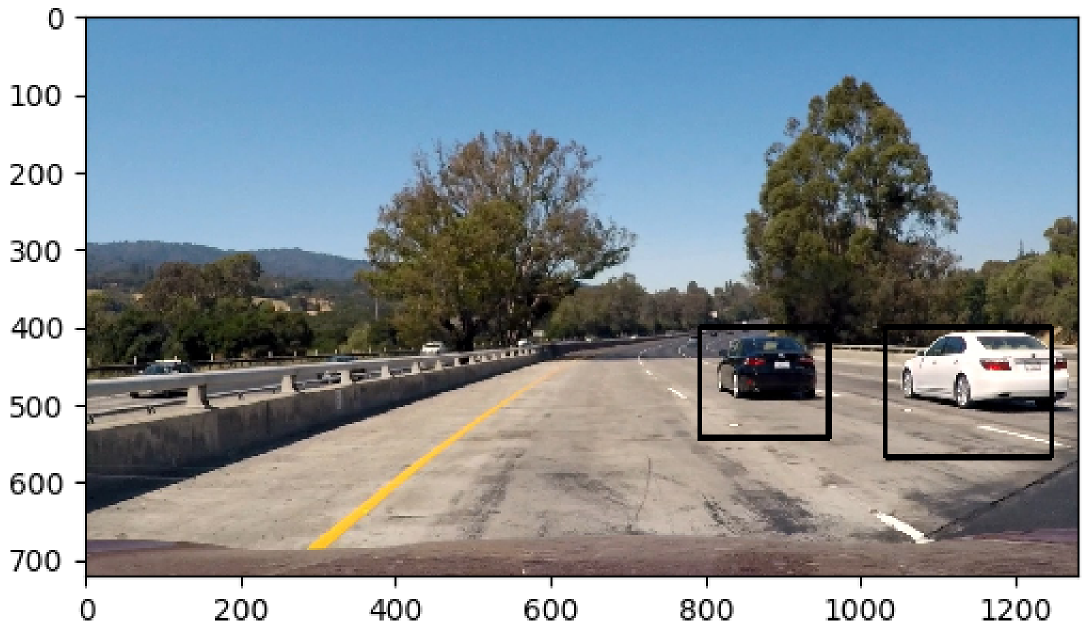
Here's an example result showing the heatmap from a series of 5 frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the detected cars:

Here are six frames and their corresponding heatmaps:





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


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

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 approach I took in the project was to start with the guidance provided in the lessons; namely, to start with extracting different kind of features from the image and comparing the feature space of different types of car and non car images and seeing if there are any noticeable similarities between features corresponding to the same class. Color histogram based features and downsampled spatial reslution based features were easy to start with but were not very robust because they easily result in false positives. Then I added HOG features to my pipeline and results improved considerably.

I tried to remove the false positives in the video stream by summing up the heat maps of 5 consecutive frames and thresholding the result to decide whether the current frame should display a bounding box around the `hot` values.

However, I still see a bunch of false positives in the video. I will try to improve on the project by playing with the thresholding and summing of consecutive heat map frames to see if I can improve on removing the false positives from the results.