Vehicle Detection

This is the Udacity vehicle detection project. The goal of this project is to detect vehicles in an image and by extension in a video.

Method

The 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
- Apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- Normalize features and randomize a selection for training and testing.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the 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.

Histogram of Oriented Gradients (HOG)

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

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

Ans 1 & 2:

feature_extraction.py is used for extracting HOG and other features. The variables were selected after many trial and error. The image was converted to the YCrCb colorspace as it consistently gave better results over HSL, RGB, etc. HOG was generated for each color channel with 9 orientations and had 8 pixels per cell and 2 cells per block.

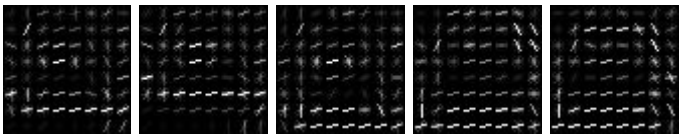
Here for a 64x64 image, it gives 1764 features per channel or an array of shape (7, 7, 2, 2, 9) before flattening (5292 features)

I take the color histogram of each color channel and concatenate them together. I divide the histogram into 32 bins. Adding the three bins, this returns a total of 96 features.

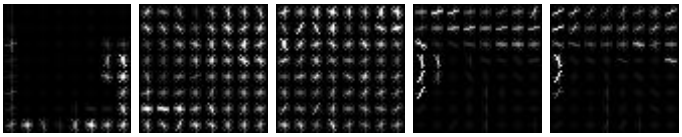
Afterwards, a spatial binning is performed after resizing the image to 32x32 which flattens the image array. So, for a 32x32 image with 3 channels, we get 3072 features.

There is a total of 8460 features per image.

The HOG images of car images are shown below:



The HOG images of non car images are shown below:



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).

A LinearSVC classifier was used to train the model as it was more accurate and faster than other SVC kernels as well as DecisionTreeClassifier. The classifier had an accuracy score of 0.994. Once all features were extracted, StandardScaler was used to scale the features down evenly with zero mean.

This training step is done ahead of time and the fitted classifier and scaler are saved to finalized_model.pkl so it can later be called without any need to retrain model for further predictions.

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_list method accepts a start (top-left x, y) and end (bottom-right x, y). number of pixels per step and a scale can also be passed through it. Window size for each sliding window is determined by the scale. A scale of one, means the sliding window matches the size of the feature input which is 64x64. A scale of two means each

sliding window will be 128x128. This scale is also returned with each sliding window output to compute HOG features later. Sliding windows are only chosen on the bottom half of the image/frame to avoid chances of getting a false positive. The sliding window also searches across the right half of the screen as the left side oncoming traffic is separated by a barrier. This was also done to speed up the computation

sliding window implementation

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?

Vehicle detection was done by iterating through each sliding window, calculating the features for each sliding window and passing it through the classifier. The sliding window boundaries are saved whenever a car is predicted. Each sliding window that was predicted to have a car was combined and a heatmap of all those rectangles is created. Only rectangles that overlap at least once are considered actual cars. This was done to ensure there are no false positives. One detection optimization I made was to calculate the image HOG for the entire area of interest up front, rather than calculating the HOG for each sliding window. This gave a significant speed improvement. Below are the detected vehicles in the test images.

Video Implementation

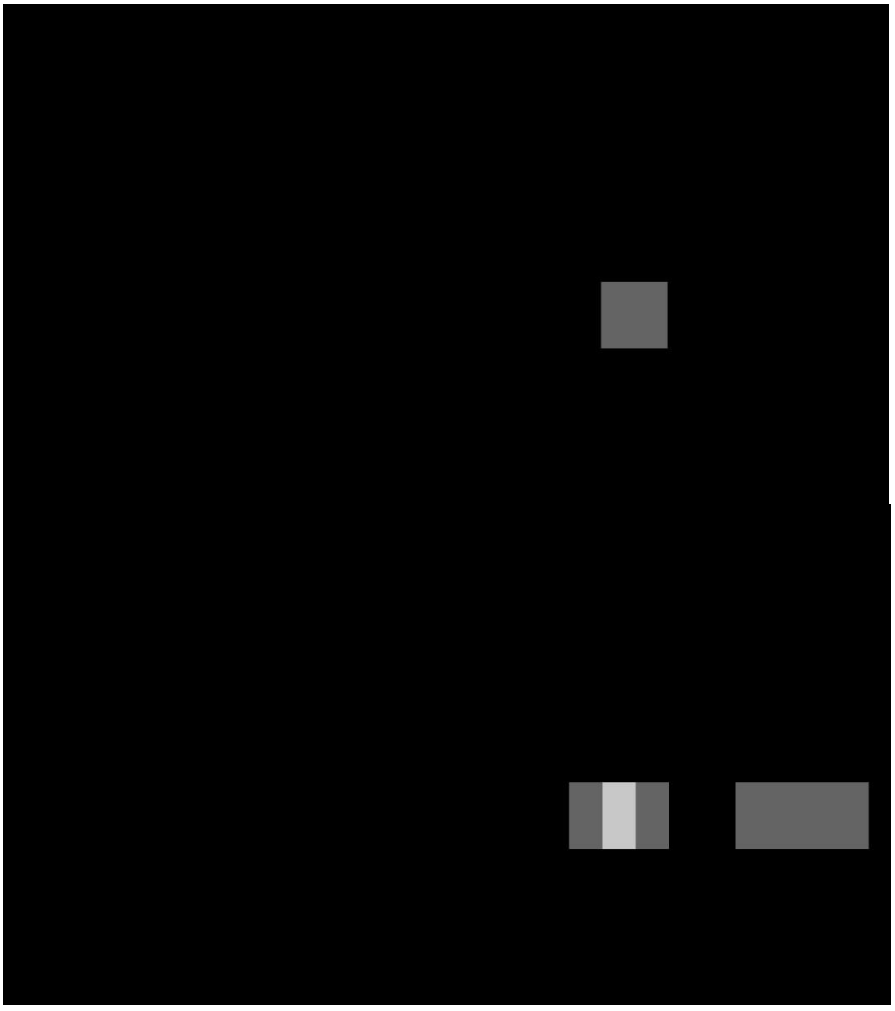
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable

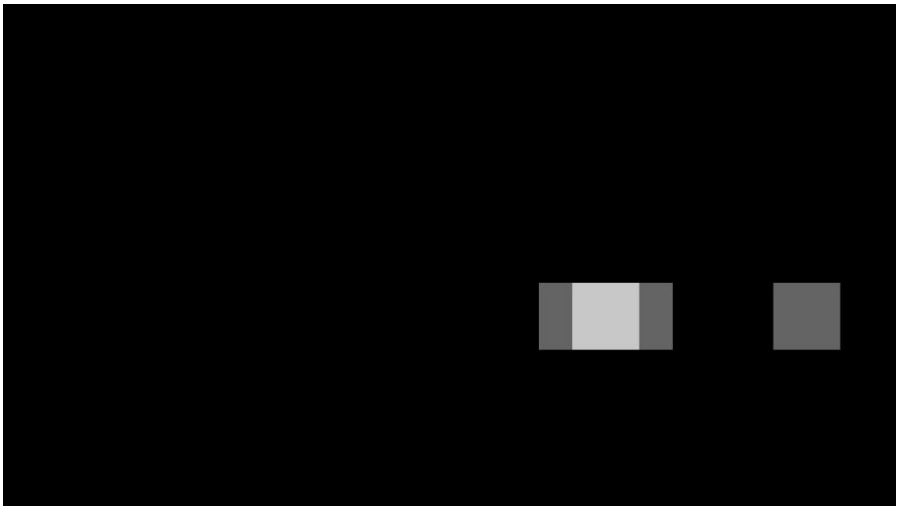
bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

This vehicle detection method is run for each image in the project_video.mp4 file. The heatmap for the last 4 frames are stored and the rectangular bounding boxes for the cars are detected from this history of heatmaps. This was done to ensure the detections don't shutter too much. The video output result has been added with the zip file

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.

Positions of positive detections were kept for each frame of the video. From the positive detections a heatmap was created and the heatmap was then thresholded to identify vehicle positions. scipy.ndimage.measurements.label() was used to identify individual blobs/points of interest in the heatmap. Each blob was then attributed as a vehicle and a rectangle encompassing it was constructed.
The following images show the sliding window with heat map applied:





Here is the final output of `scipy.ndimage.measurements.label():

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?

There were two issues that I faced in my video:

1. The detection boxes shuttered sometimes. I believe it can be fixed by taking more past frames into account.
2. Some false negatives were detected on the left side of the video. It can probably be fixed by changing the sliding window parameters