Self Driving Car Engineer Nanodegree
Project 5: Vehicle Detection
Mano Paul

Vehicle Detection Project

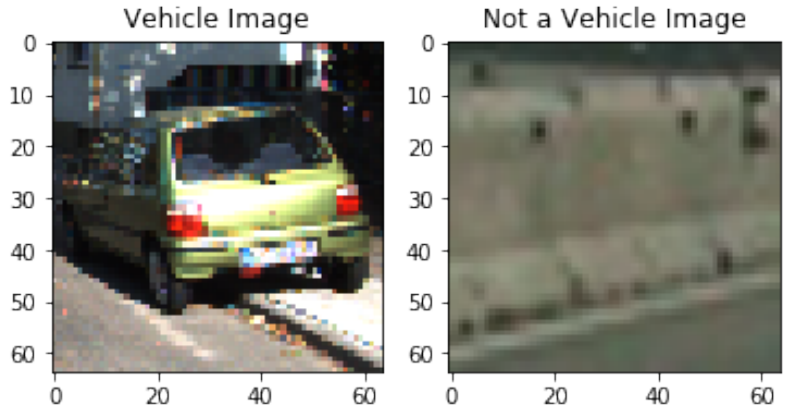
The goals / steps of this project are the following:

- 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.

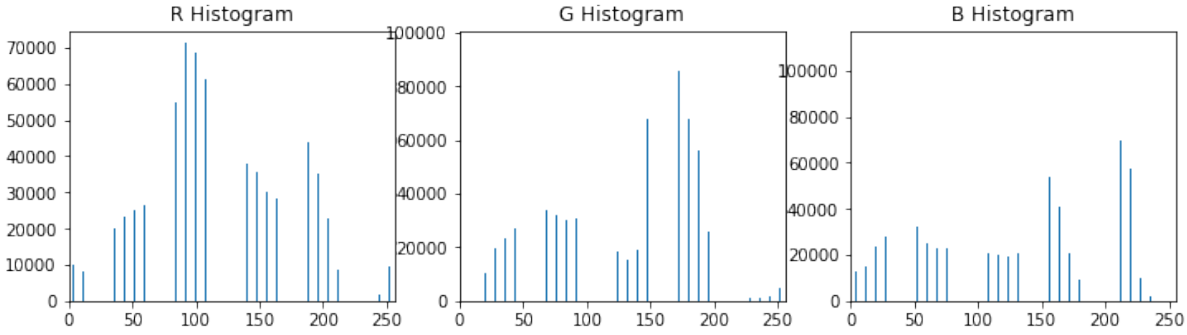
Histogram of Oriented Gradients (HOG)

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

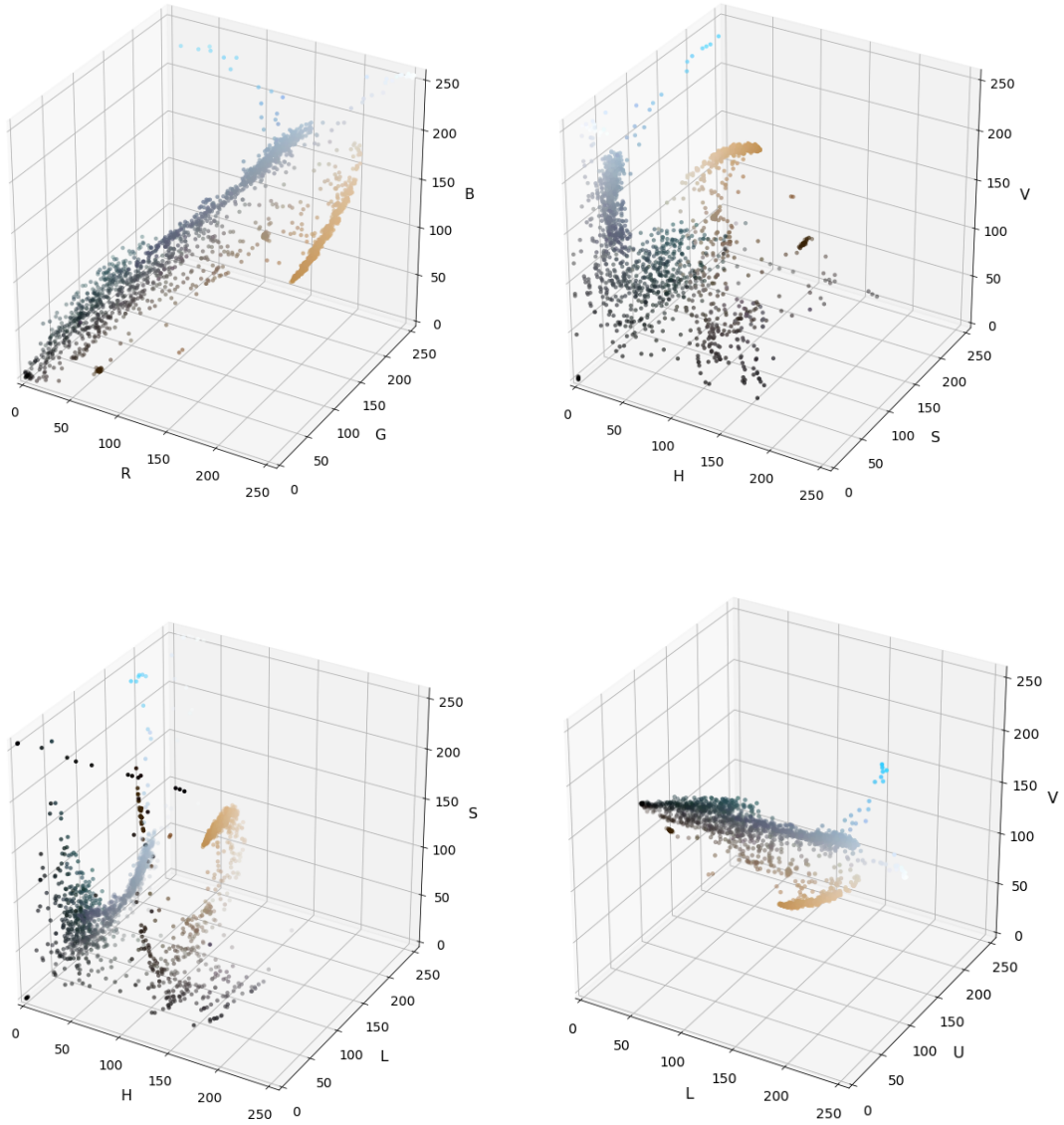
The ***get_hog_features()*** helper function in the IPython notebook is used to extract the Histogram of Oriented Gradients from the training images. This is defined as a helper function along with other helper functions (cell 1 to 6) that are used to determine the color space and histograms of the images.
The hyper-parameters that can be tuned are defined in cell 7.
The ***explore_dataset()*** function is used to explore the dataset of vehicle and not-vehicle images.
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:



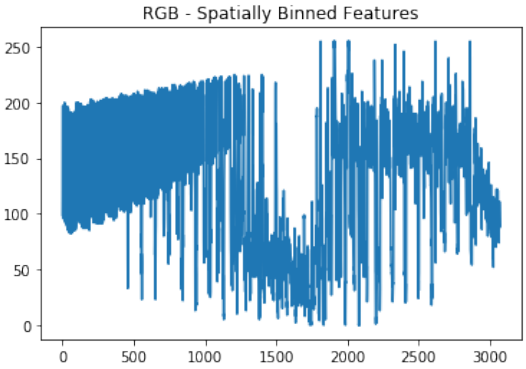
The R, G and B histogram features for test1.jpg image (see below) were plotted.

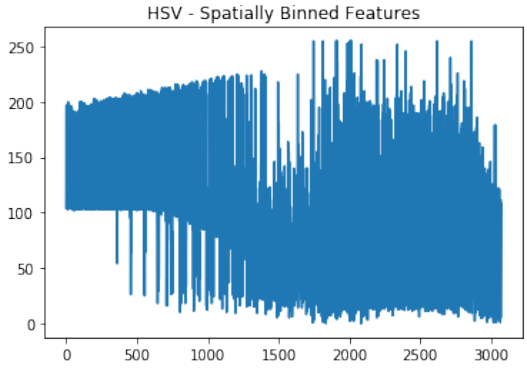


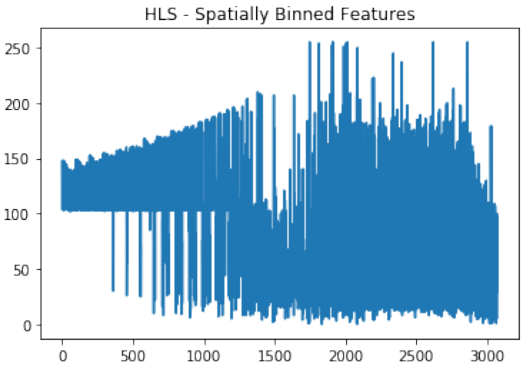
Since the R, G, B values directly or the raw pixel intensity (histogram) does not solve the problem of classifying objects of the same class (say all cars) that can be of different color. I explore color spaces (like RGB, HSV, LUV) which is the distribution of color values in a image. By plotting each pixel in some color space, we can learn more about the distribution of color values in an image.
It was noticed that it was hard to distinguish between the class of pixels that belonged to vehicles, from that of the background. It was more beneficial to to plot pixels from vehicle and non-vehicle images separately.
The RGB, HSV, HLS and LUV values of the test1.jpg image was plotted.

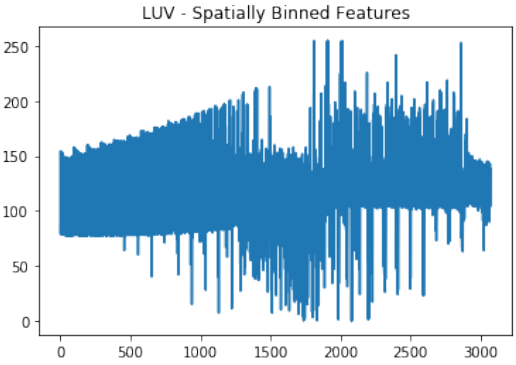


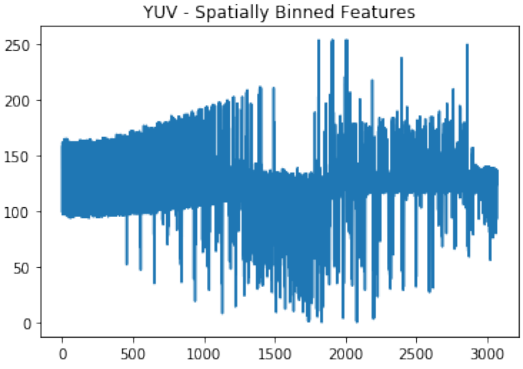
Since it is cumbersome to include three color channels of a full resolution image, you we can perform spatial binning (converting the image into a 1 D vector) by resizing the image into a 32x32 or 64x64 pix image. Even resizing an image into a 32x32 pix image still retains enough information to help in finding the desired object (say vehicle). The OpenCV's cv2.resize() function was leveraged for spatial binning.

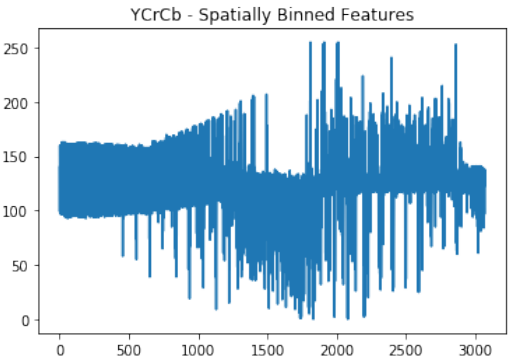

RGB - Spatially Binned Features


HSV - Spatially Binned Features


HLS - Spatially Binned Features

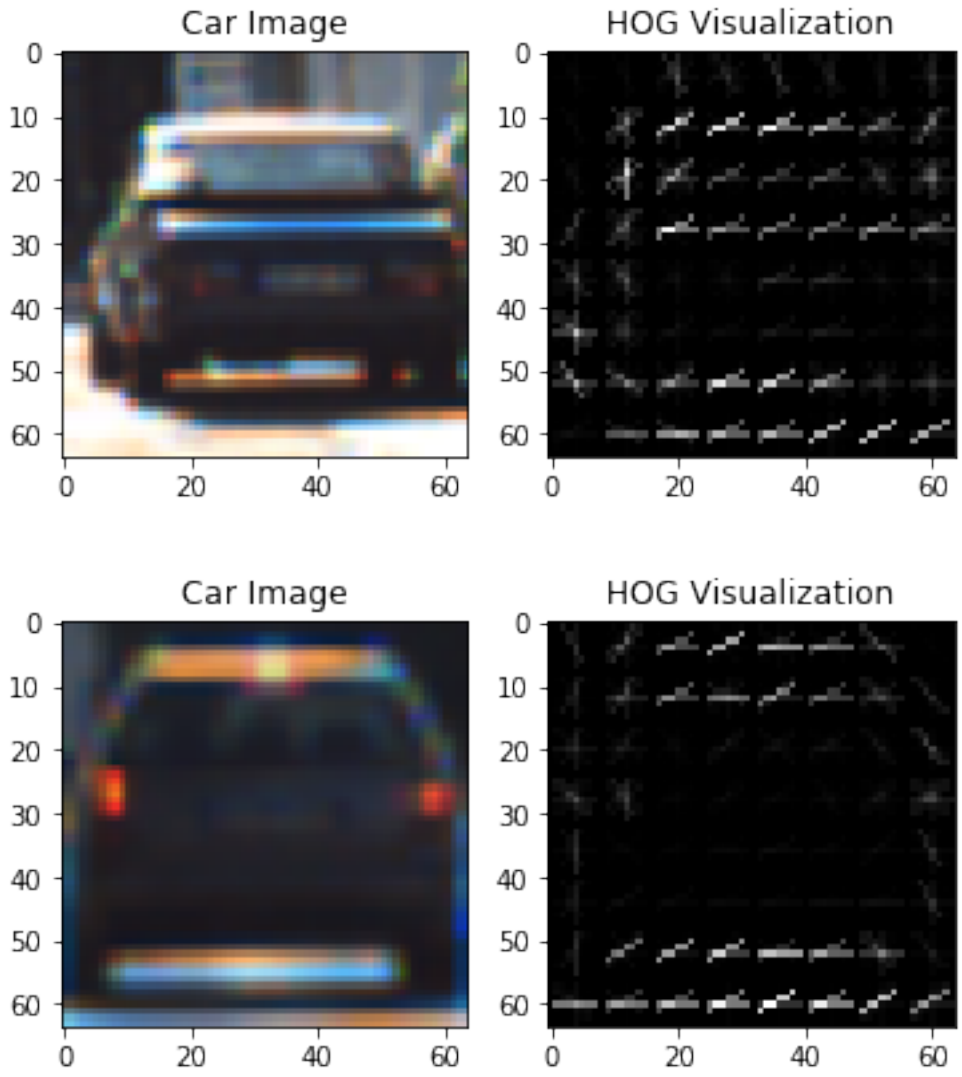

LUV - Spatially Binned Features

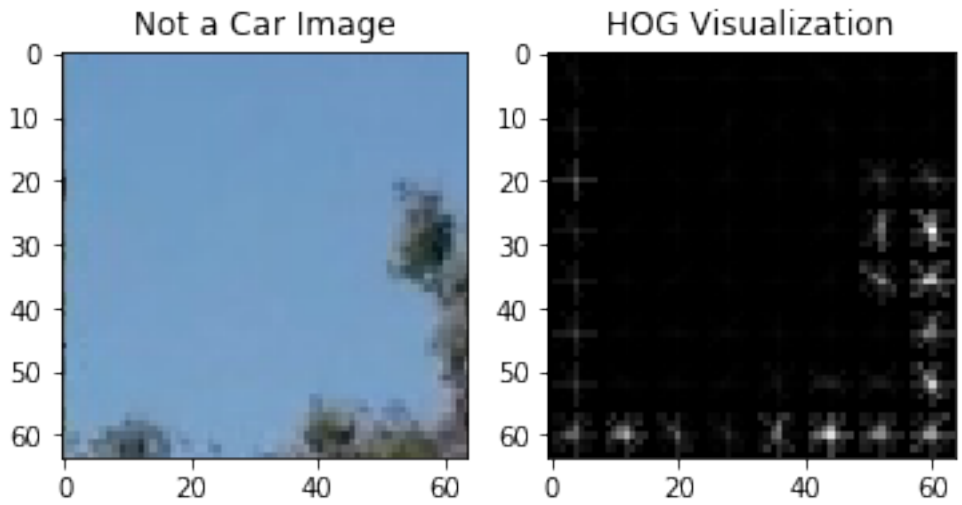

YUV - Spatially Binned Features

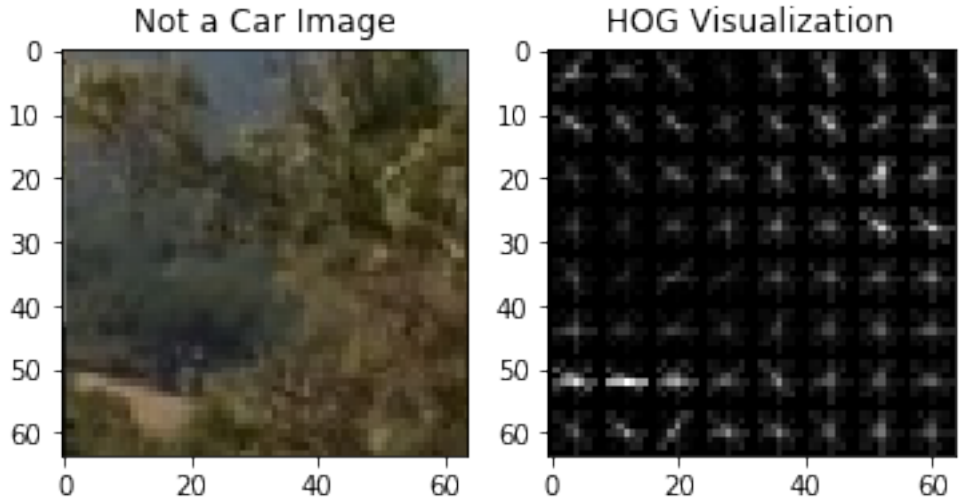

YCrCb - Spatially Binned Features

Since a histogram of color gives Color only characteristics, the Histogram of Oriented Gradients was plotted to get an understanding of the shape.

I plotted some 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 = 8, pixels_per_cell = (8,8)*** and ***cells_per_block = (2,2)***







HOG Parameters

I tried various combinations of parameters. These parameters I tweaked were:

color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 8 # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 1 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimensions
hist_bins = 32   # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = False # Histogram features on or off
hog_feat = True # HOG features on or off
C_value=1000 #1,10,100,1000

A combination of spatial binning features and HOG features showed the best chances of detecting vehicles from non-vehicles.

Classifier Training

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 a dataset of 8792 normalized images of vehicles and non-vehicles and a C value of 1000.

```
The explore dataset function returned a count of
8792 vehicles and
8968 non-vehicles
of size:  (64, 64, 3)
and data type: float32
Number of vehicle images used for training classifier:  8792
Number of non-vehicle images used for training classifier:  8792
```

The sample size of vehicle and non-vehicle images were made the same to reduce or remove bias of the classifier to classifier one object incorrectly as another.
Normalization was achieved using StandardScaler function from sklearn.preprocessing
The train_classifier() function extracted the HOG features from a randomized training dataset which was 80% of the total images. 20% was used as a test dataset.

The trained classifier had a test accuracy of 98.92%.

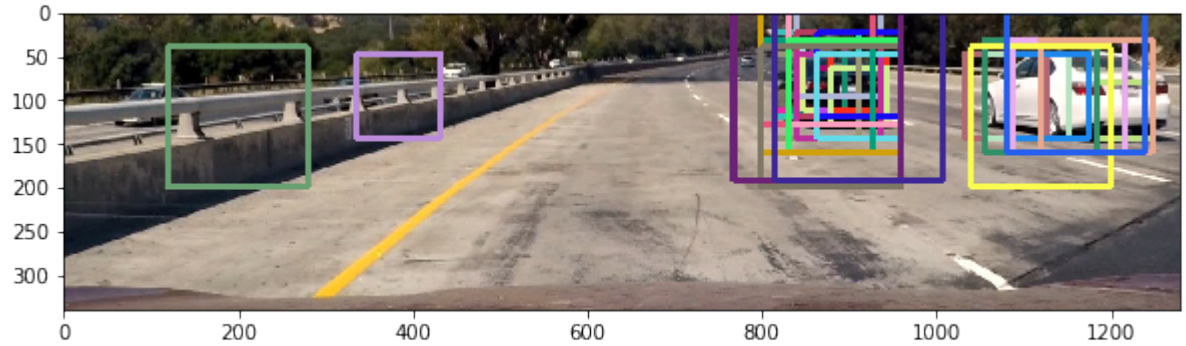
```
# Training the classifier
svc, X_scaler = train_classifier(vehicle_images, non_vehicle_images, C_val=
```

```
Starting to Train Classifier
Using: 8 orientations 8 pixels per cell and 1 cells per block
Feature vector length: 2304
1.29 Seconds to train SVC...
Test Accuracy of SVC =  0.9892
Finished Training Classifier
```

Sliding Window Search

I decided to search for vehicles only in the lower portion of the image and so in order to do that, I found divided the height of the image into half and used that as the starting point from which to search down. The searchable area was from that starting point to the entire height of the image minus 20 pixels for the hood of the vehicle in the image. An example of this shown below.

Upon setting the searchable area of the image, using a scale from 1 through 3, in increments of 0.5, bounding boxes of detected vehicles using the find_vehicles() function were plotted with random colors as shown below.



```
Number of bounding boxes:  35
```

The find_vehicles function extracts hog features once which is then sub-sampled to get all of its overlaying windows. Each window is defined by a scaling factor where a scale of 1 would result in a window that's 8 x 8 cells then the overlap of each window is in terms of the cell distance. This means that a cells_per_step = 2 would result in a search window overlap of 75%. It was possible to run this same function multiple times for different scale values to generate multiple-scaled search windows

Here are some example images.

Since the classifier used was a LinearSVM classifier, the C value was throttled along with different color spaces to optimize and yield the best result with the minimal number of false positives.

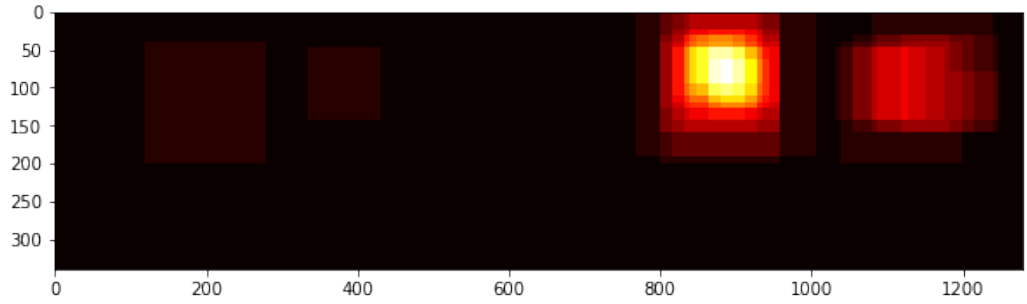
Video Implementation

Here's a link to my video result
https://youtu.be/twaoganSr34

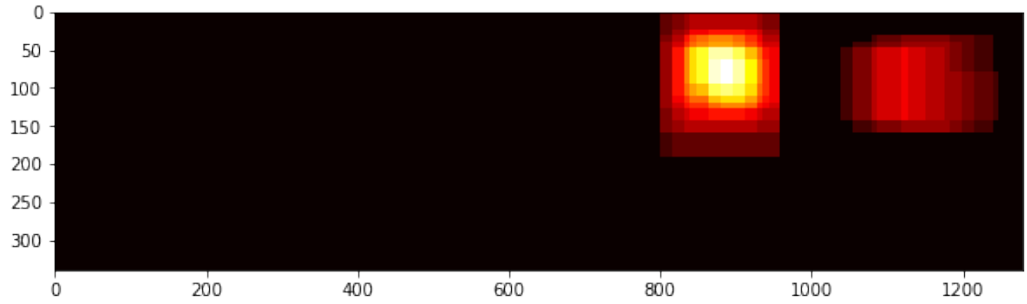
Filtering False Positives

I recorded the positions of the detected vehicles in each frame of the video. A heatmap image was generated to detect location of the vehicles and to reject any false positives. False positives were rejected by applying a threshold to the heatmap images as shown below.

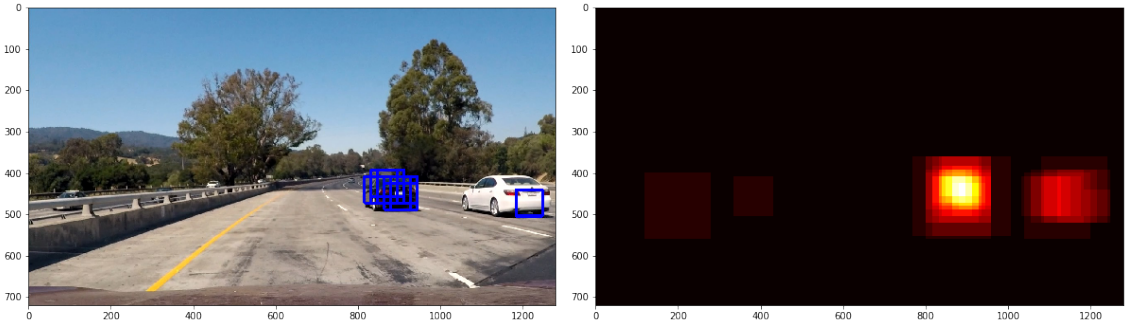
Example of a cropped heatmap Image

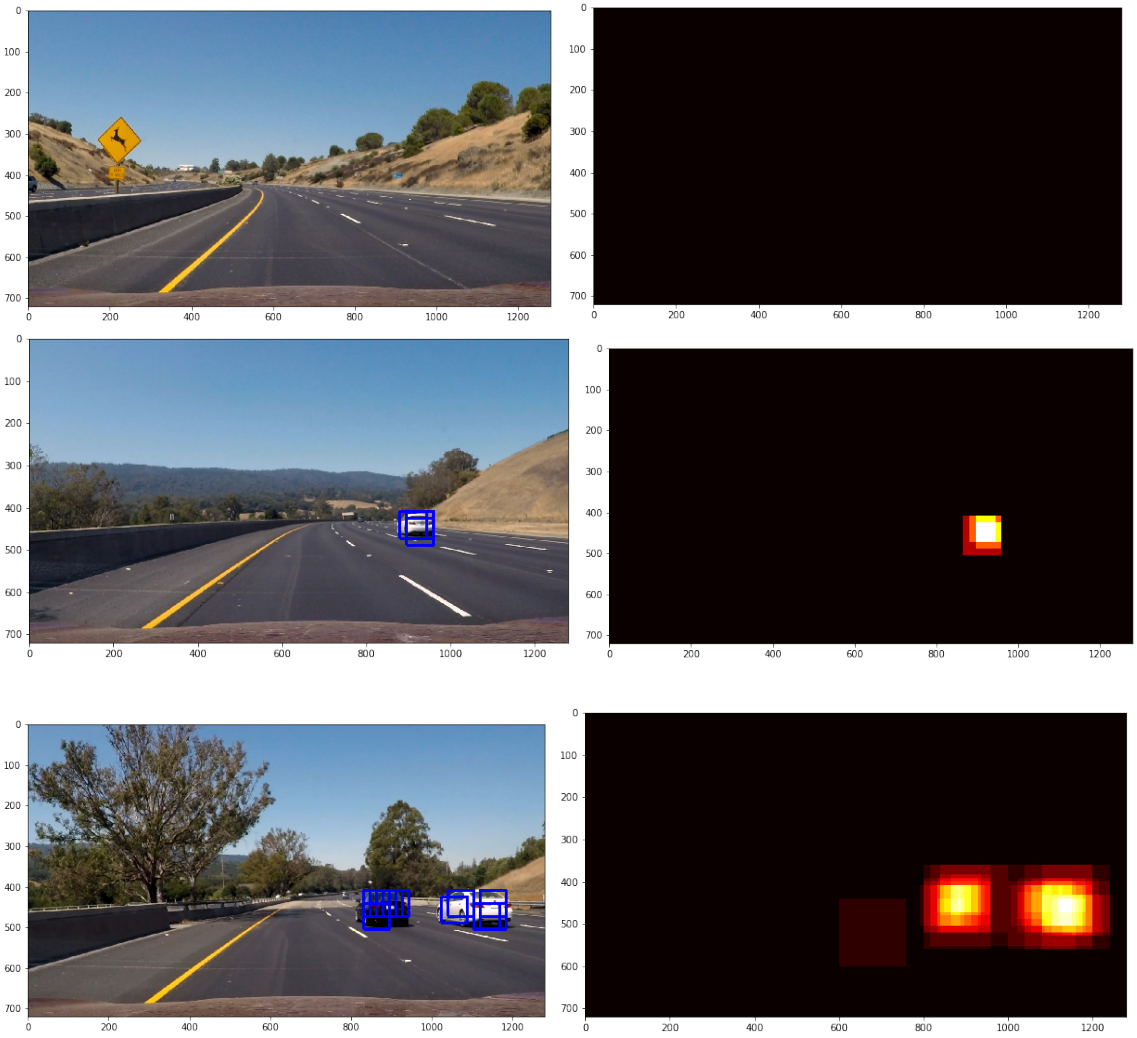


Example of a cropped thresholded heatmap image

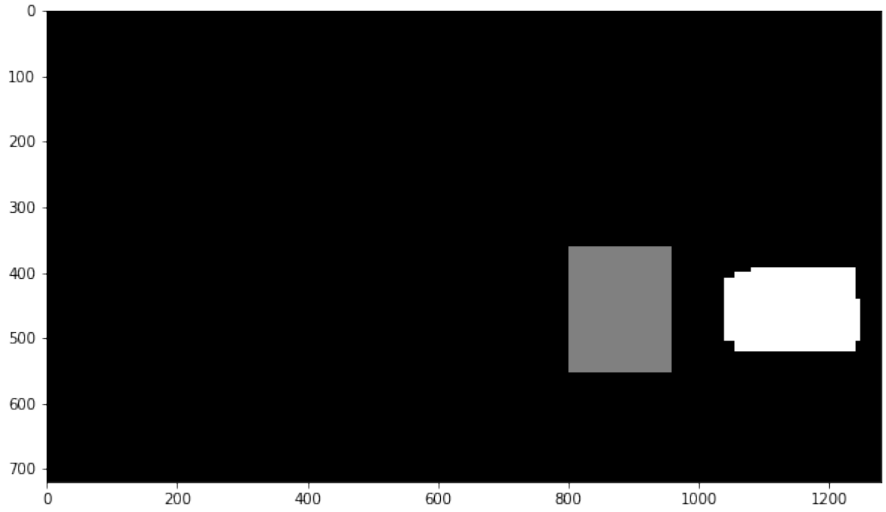


Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:





Once I had the thresholded heat-map, there were many ways I could have gone about trying to figure out how many cars were in each frame and which pixels belong to which cars, but one of the most straightforward solutions was to use the label() function from scipy.ndimage.measurements, which is what I did as shown in the image below.



Finally the resulting bounding boxes are drawn onto the image in the last frame.

Discussion

I faced difficulties when combining features from spatial binning, color histograms and HOG and so resorted to only using spatial and HOG features. Also since the test and training data in the classifier was randomly selected and shuffled, the classifier resulted in varying test accuracy with each run. The pipeline still missed out detection of the vehicles in a few situations when the road matched the color of the vehicle or detected vehicles in shadows when there was none.