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 Linear SVM Classifier.
- Optionally, you can also apply a color transform and append 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 heatmap of recurring detections frame by frame to reject outliers and follow the detected vehicles
- Estimate a bounding box for vehicles detected.

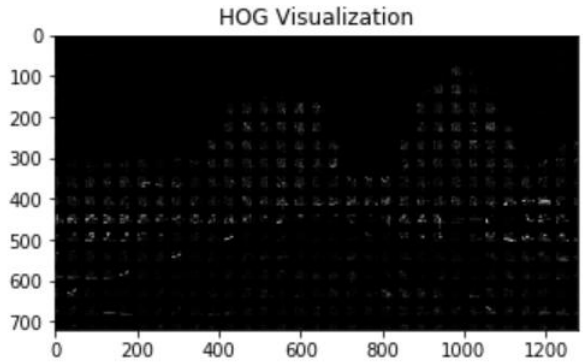
Writeup

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1- **Provide a Writeup that includes all the rubric points and how you addressed each one.** Here are the Writeup for the project you are reading it.

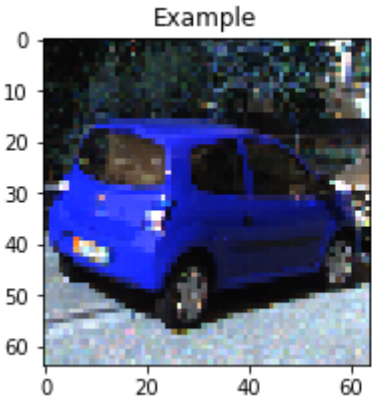
Histogram of Oriented Gradients (HOG)

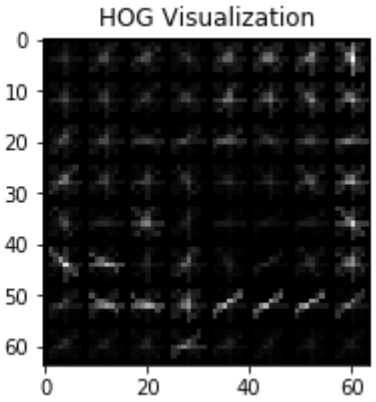
1- **Explain how (and identify where in your code) you extracted HOG features from the training images.**
The code for this step is contained in Section (2) which is named by **Features Extraction** especially Section (2c) which is named by **Oriented Gradient Histogram Features.** However this function takes the image as an input, so it performs the extraction only for the HOG. Here is an example image:

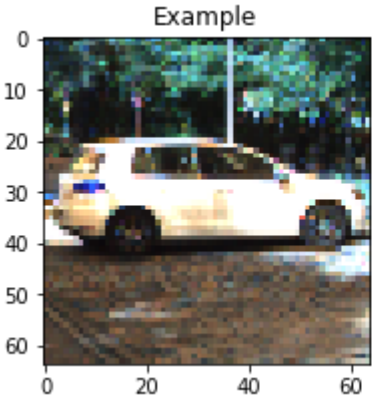


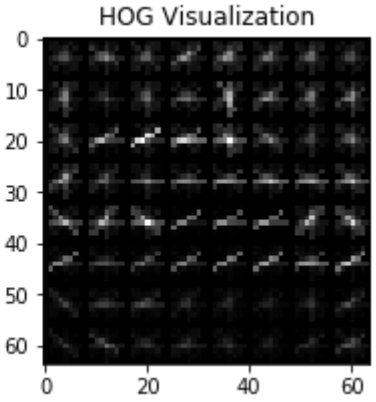
I then explored different color spaces and “skimage.hog()” parameters like: **Orientation, pixerls_per_cell, and cells_per_block**. I grabbed random images from each of the two classes and displayed them to get a feel for what the “skimage.hog()” output looks like.

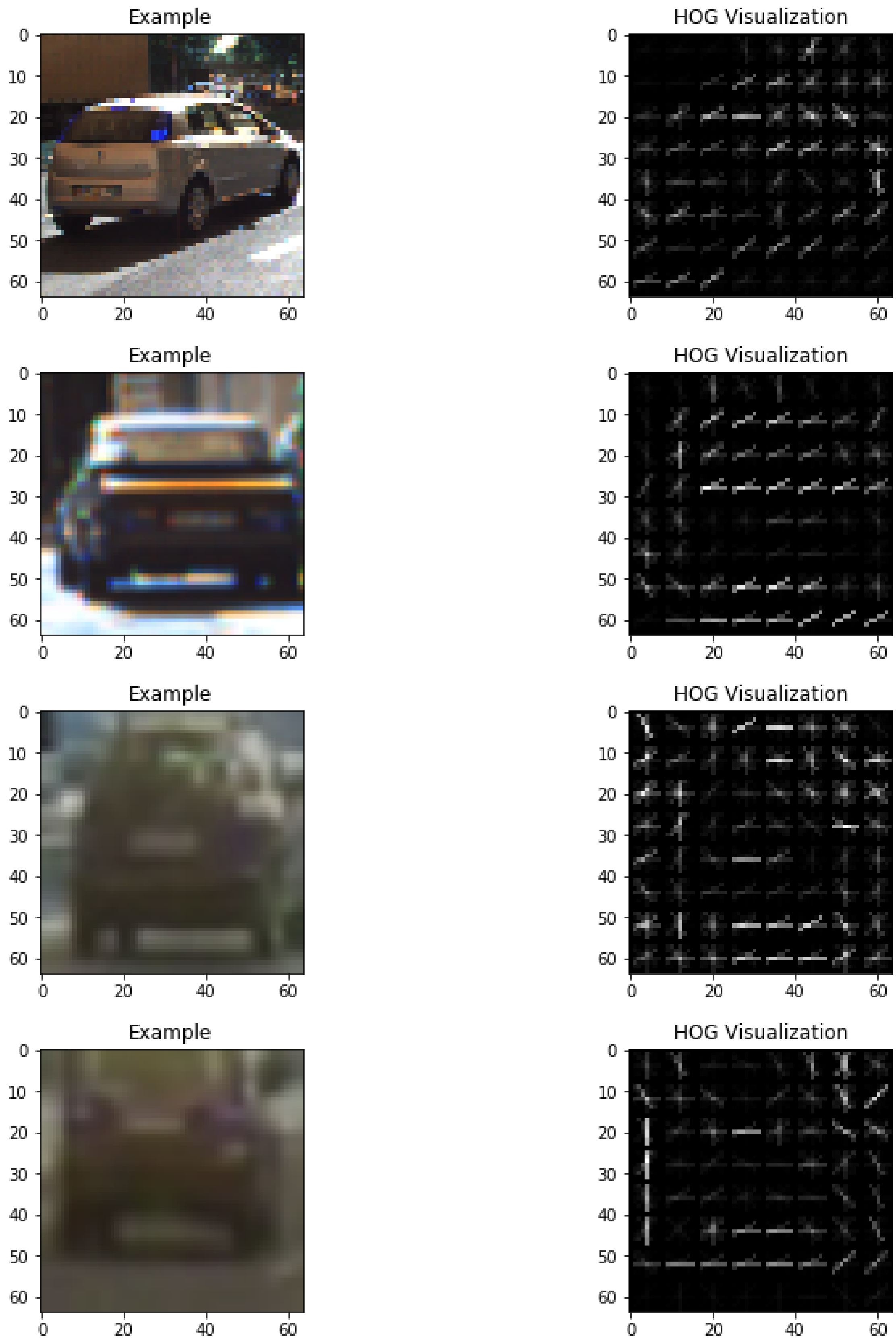
Here are examples using the “YCrCb” color space and HOG parameters of:
Orientations = **8**, pixels_per_cells = **8**, cells_per_block = **2**:

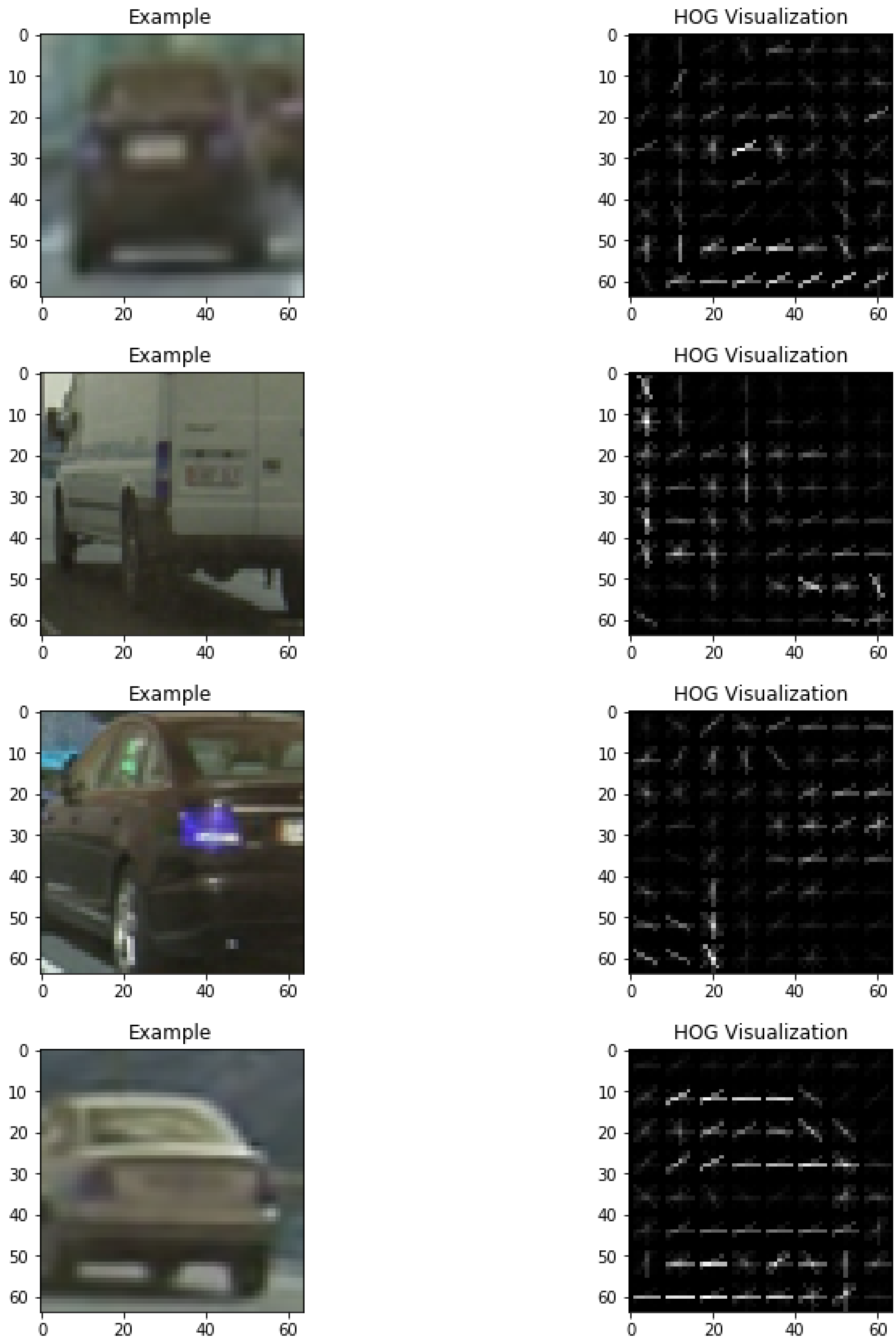


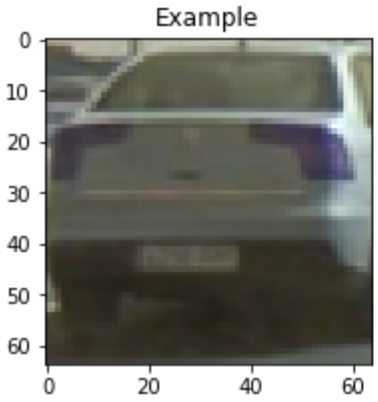


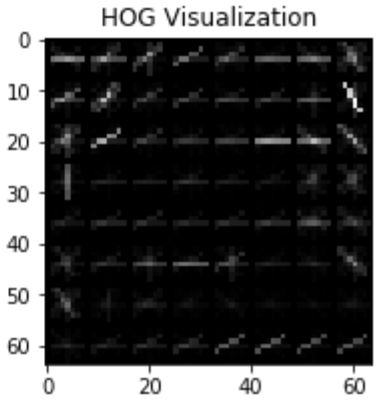


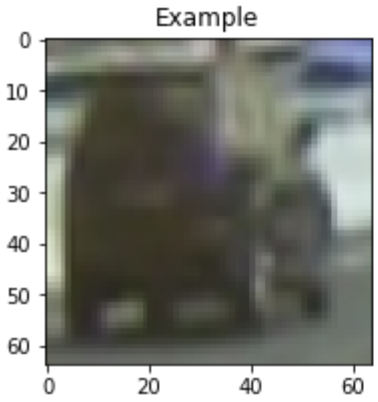


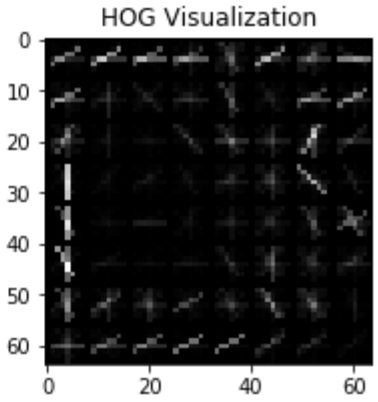

Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization
0
10
20
30
40
50
60
0
20
40
60

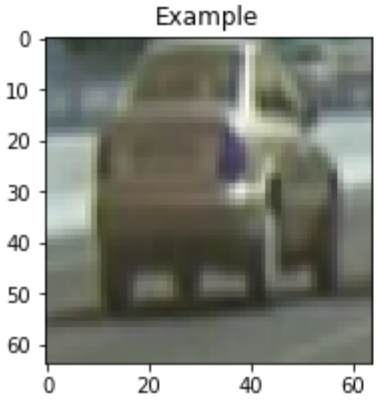

Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization
0
10
20
30
40
50
60
0
20
40
60

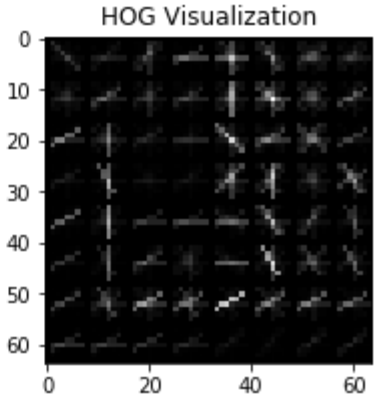

Example

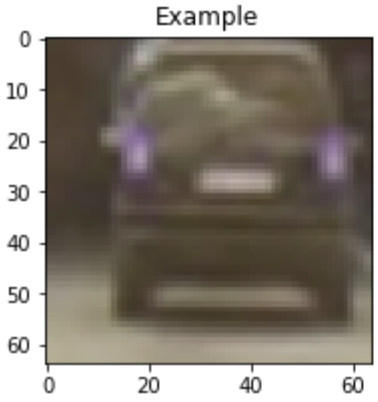

HOG Visualization

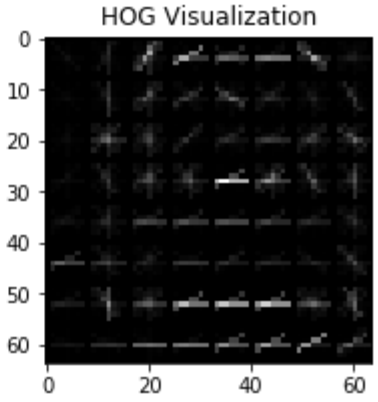

Example

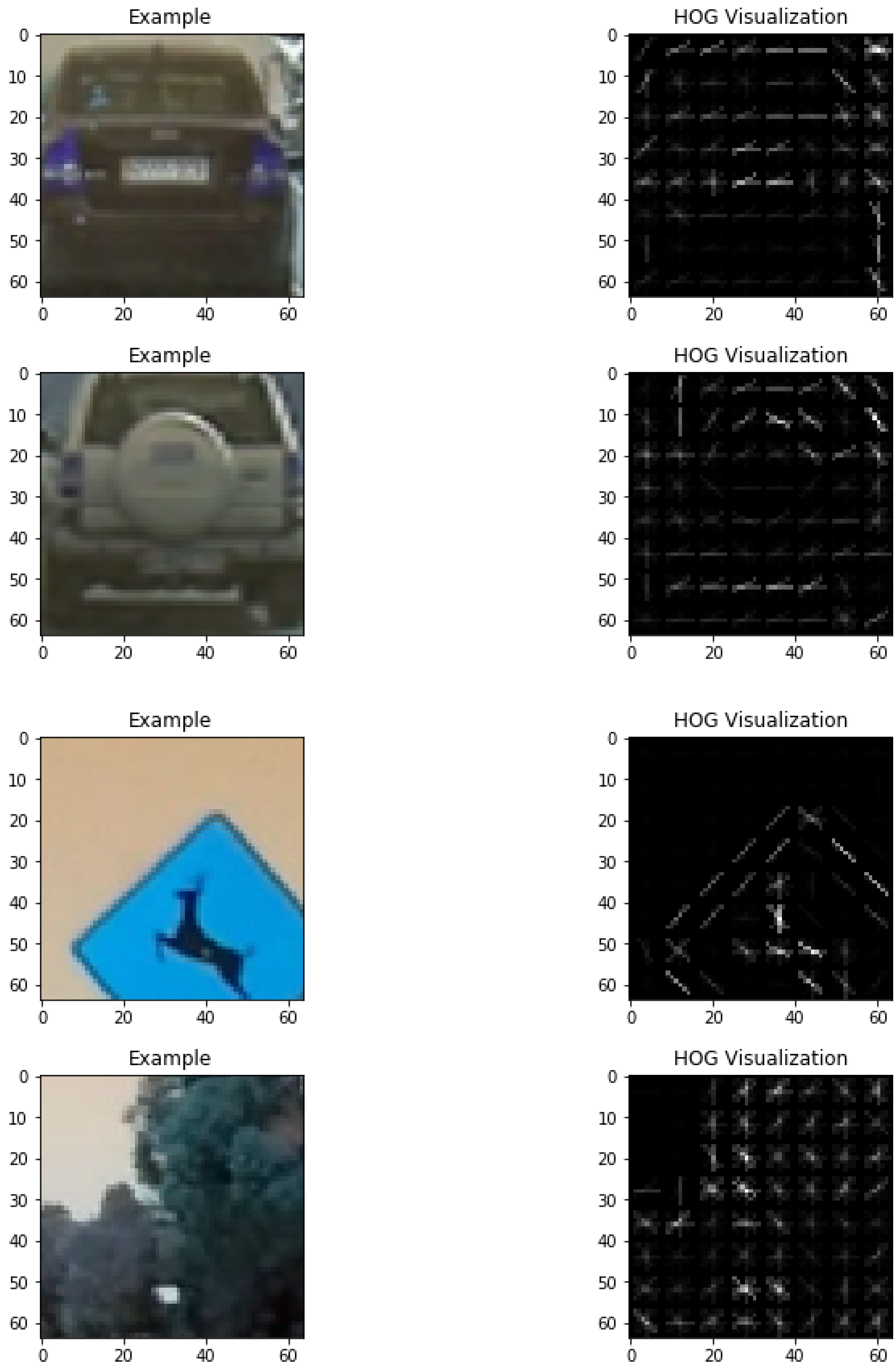

HOG Visualization

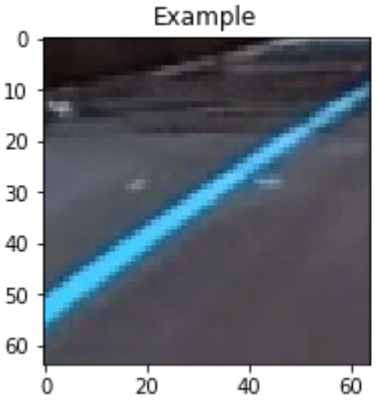

Example

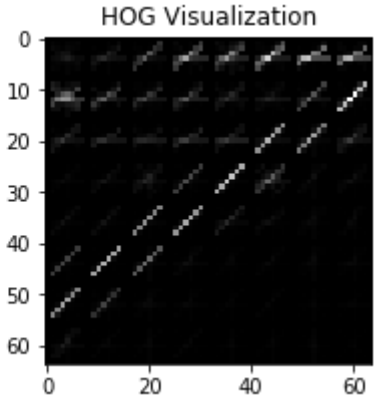

HOG Visualization

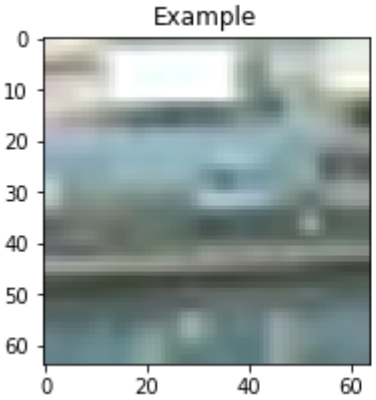

Example

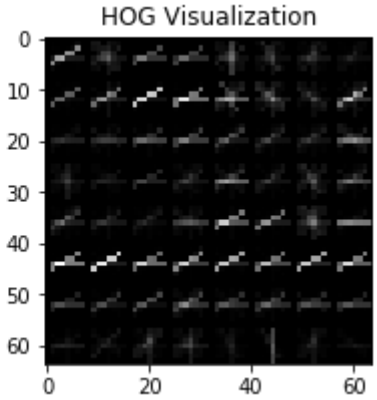

HOG Visualization

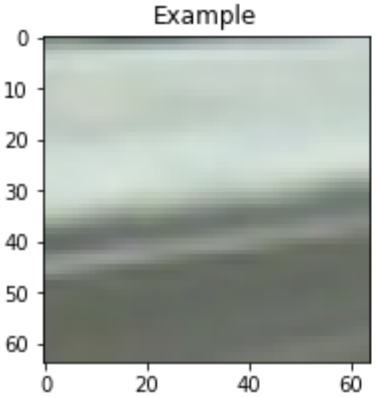

Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization
Example
HOG Visualization

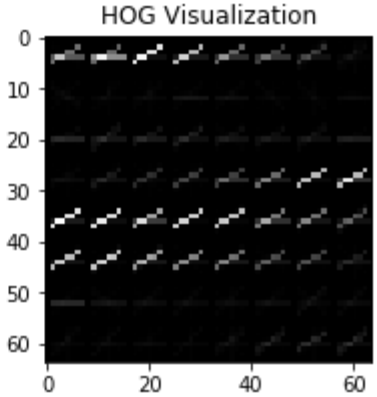

Example

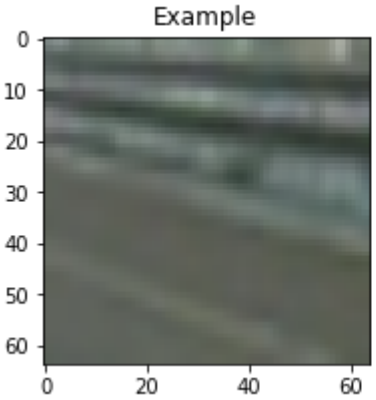

HOG Visualization

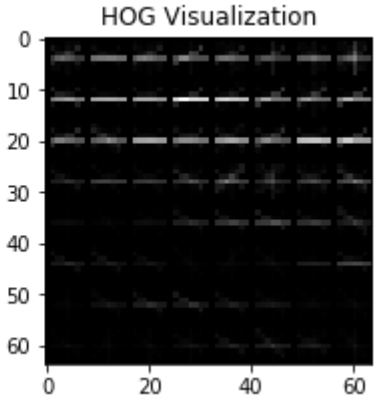

Example


HOG Visualization


Example


HOG Visualization


Example


HOG Visualization

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

I tried various combinations of parameters for the Orientations, pix_per_cell, and cell_per_block, in addition to the color space. For all of the various combinations, I have extracted the features using the classifier Linear SVM. I have splitted the (Vehicle) and (non-Vehicle) data we have into training and testing data with 80%, and 20% respectively. I have trained the classifier over the training data, then test on the testing data then check the accuracy of the classifier. For these parameters, I have achieved **99.2%** accuracy which was the highest accuracy I have reached from the whole combinations.

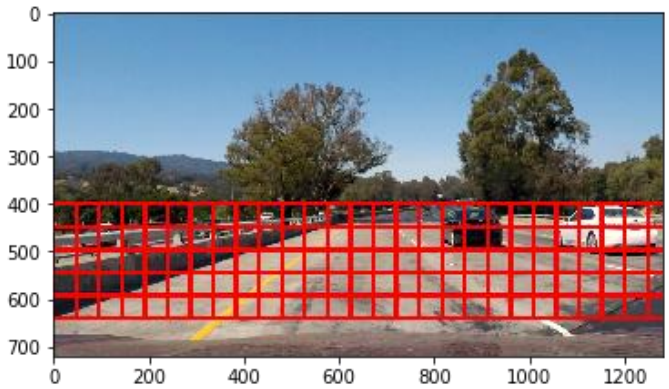
3- **Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).**

I trained a linear SVM using Linear_SVC() with its normal parameters. Training for the SVM classifier in the ipython notebook is found in the 5th Section which is named by (Build Classifier (Normalize, Train, Test, Accuracy calculation)). I have also depend on the color features extraction in order to help the classifier having enough and different features for the vehicles and non-vehicles data which make it easy to differentiate between them.

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?**

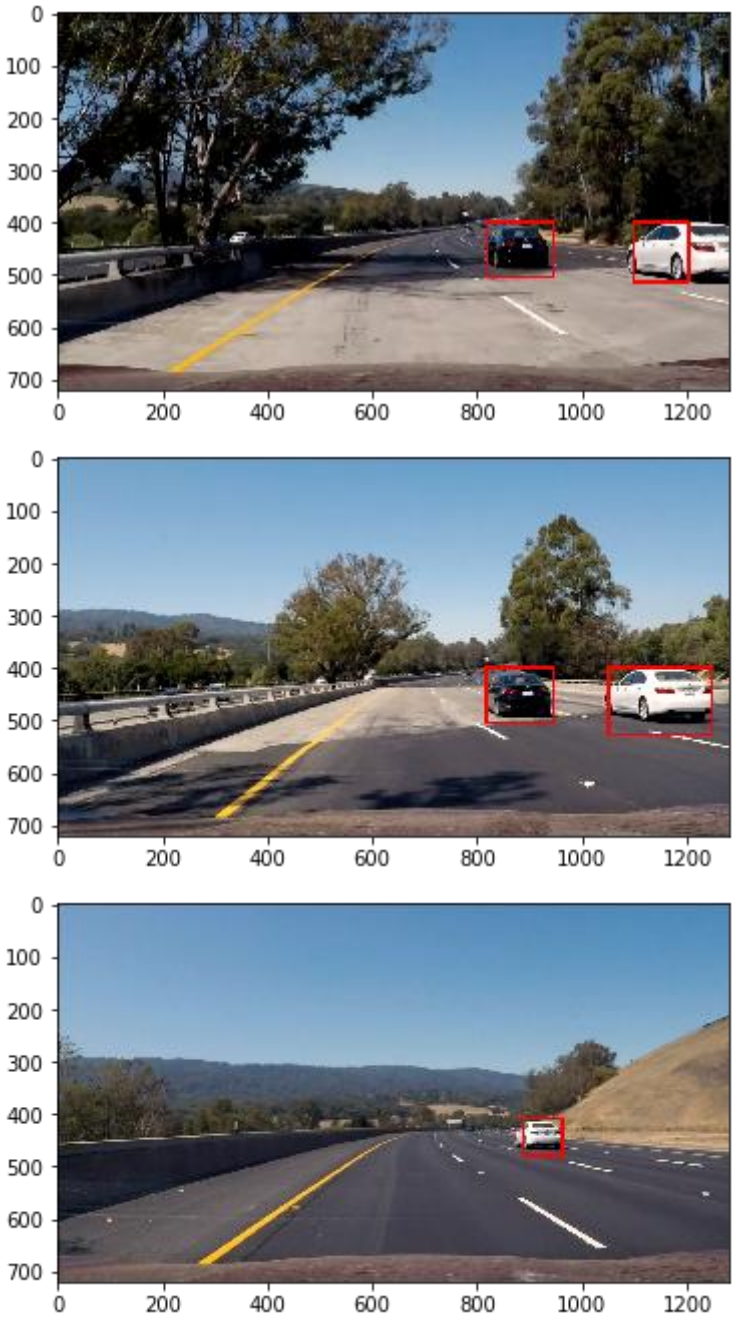
I have used the Sliding window technique for searching which is found in the 6th Section in the Ipython notebook named by (Sliding Windows method). I monitored, from the testing images, that the road X-limit starts approximately from 400 to 680 due the appearance of the car in the images, so I prevented having boxes for this part. After that, the Y-limit is extended in the whole image, so there is no limit for the Y component. Here is only an example for the Sliding window Searching method:



The previous example has an overlap with (0.5), but actually we found that this overlap is not enough for having heatmap with a high value, so I have used an overlap factor with (0.9) as you can see in 8th Section when applying the full pipeline for finding Cars. In addition to that, I didn't use a fixed size for the sliding window for searching as I have applied **three sliding windows in different regions in the image.** I have decomposed the scene into two parts:

1) The first part has X-limits from 400 to 600. This part contains far vehicles, so I have applied square window whose length equals to 64 pixels.
2) The Second part has X-limits from 400 to 680 which is the Region of Interest that is mainly contains vehicles. I have applied two squared windows whose lengths equal to 96, and 128 pixels for the aim of having bigger cars (more nearer).

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 features vector, which provided a nice result. Here are the testing images:


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

Video Implementation

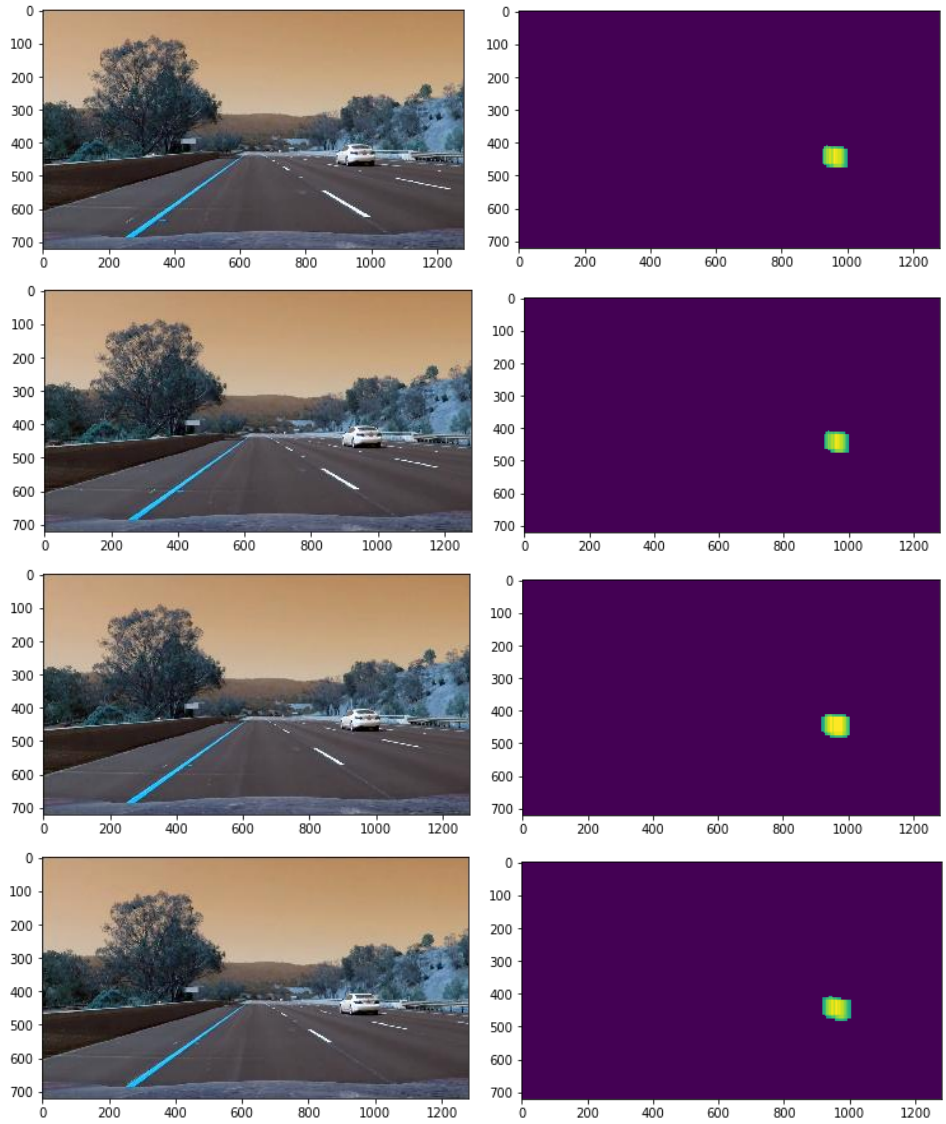
1- **Provide a link to your final video output. Your pipeline should perform reasonable well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)**
The Project Video is attached in the github repo sent for the project which is called (output.mp4).

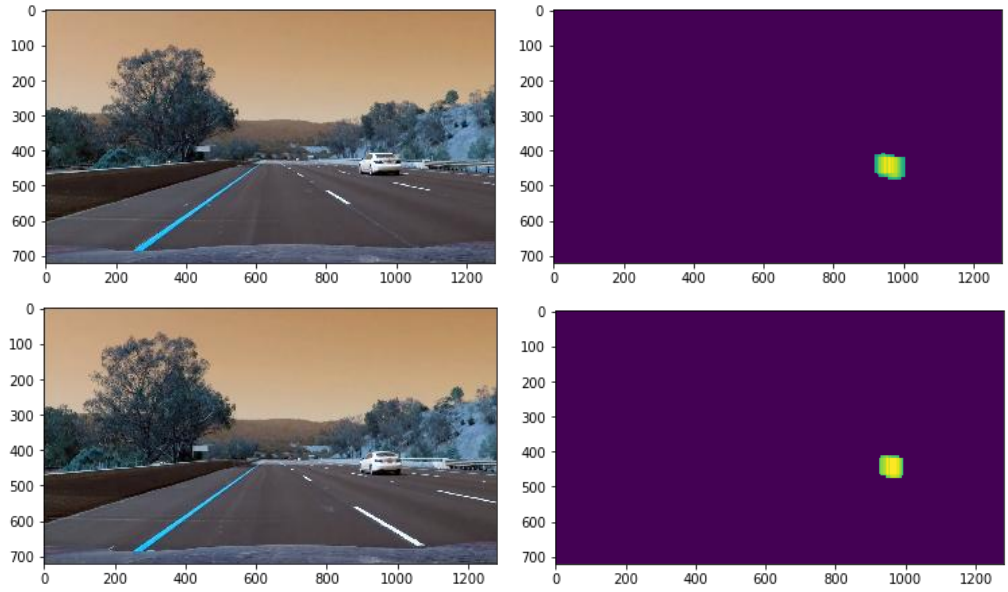
2- **Describe how (and identify where in your code) you implemented some kind of filter for false positives and some methods for combining overlapping bounding boxes.**
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that heatmap to identify vehicle positions. I then "scipy.ndimage.measurements.label()" to identify individual blobs in the heatmap. I then **assumed** each blob corresponded to a vehicle. Then before constructing a bounding box, I have put a threshold value that can reject the false positive detected from other non-vehicle data in the scene. Finally, I constructed

bounding boxes to cover the area of the blob detected after applying a threshold value. Here are some examples results showing the heatmap from the testing images we have, the result of scipy.ndimage. measurements.label() and the bounding boxes then overlaid on the images.

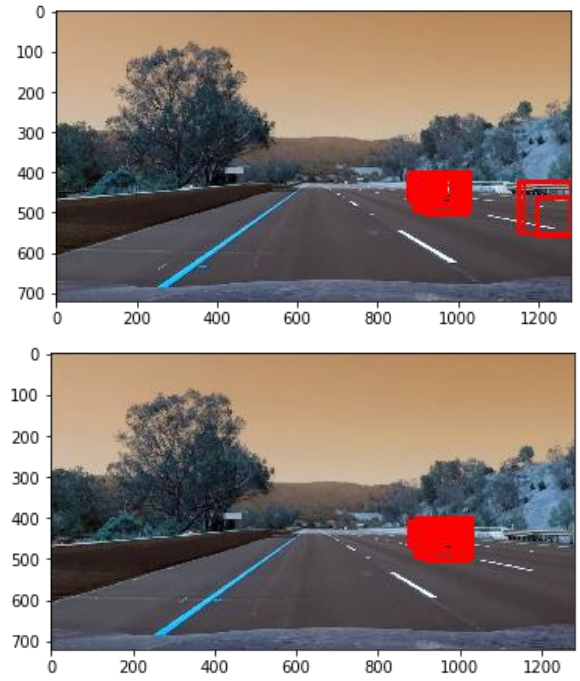
This part is found in the 8th and 9th Sections in the Ipython notebook in which we have applied the Full Pipeline for finding the cars in the scene.

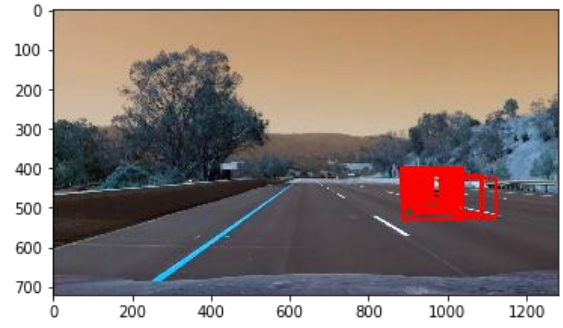
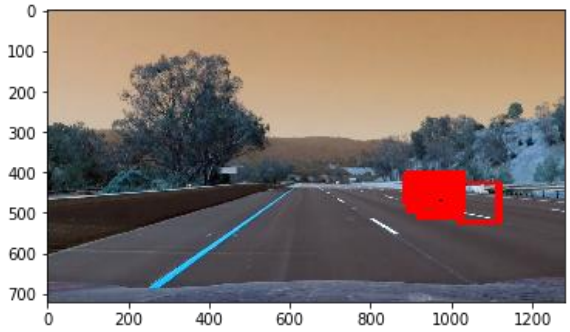
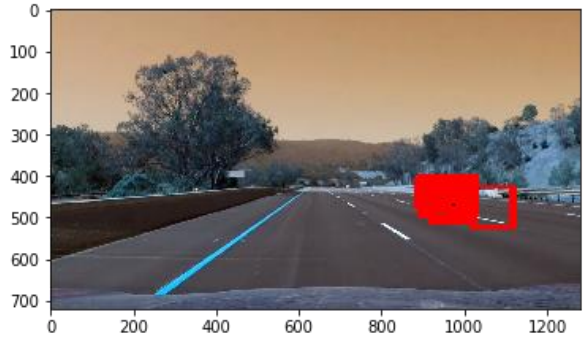
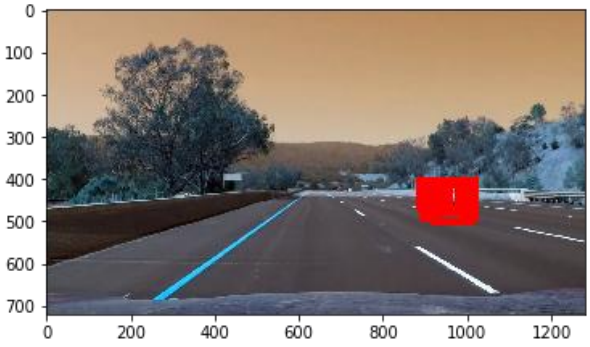
Here are six frames and their corresponding heatmaps:



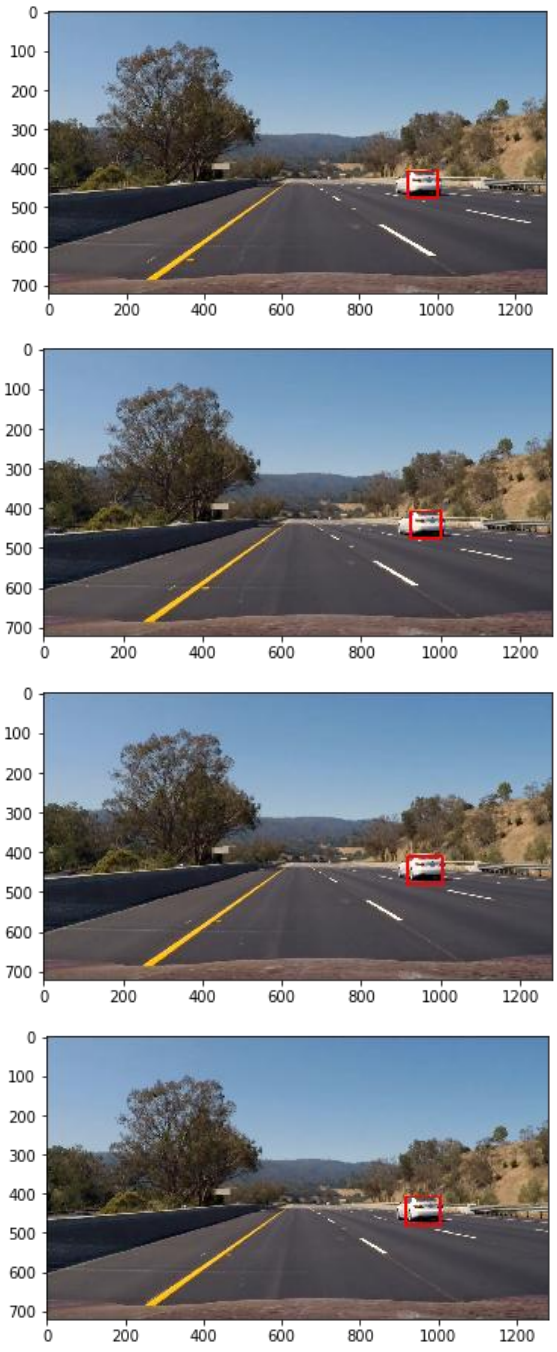


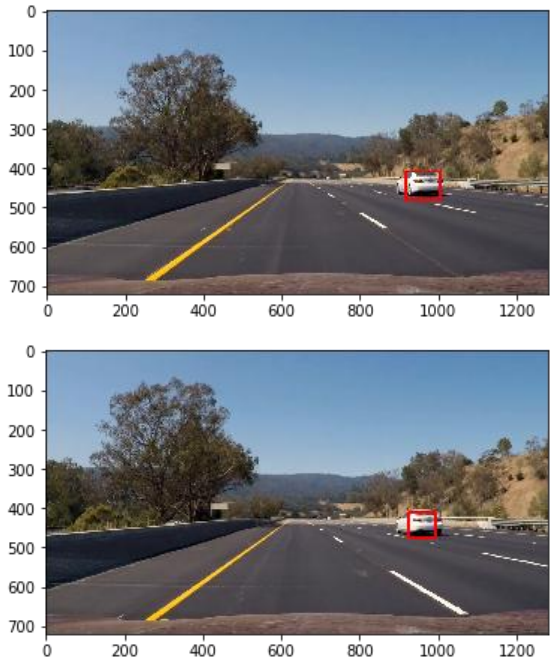
Here is the output of "scipy.ndimage. measurements.label()" on the integrated heatmap from all six frames:



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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

Actually I have faced many problems in this project as follows:

1) The tuning parameters for the feature extraction feed into the SVM classifier was difficult for the aim of achieving a higher accuracy over the testing data we have especially when combining the three types of features: Hog, color_histogram and the bin_spatial.

2) The second problem can be formulated in the processing done on the Project_Video image by image due to applying the (find_cars) functionality on the video image by image. The output video is extracted after about 24 hours.

The project may fail when there are some features in the scene approximately like the cars features. This is actually is happened on the Project_video, but due to the applying a sufficient threshold value on the heatmap, I have rejected many false positives detected from the sliding window searching method. This threshold value also causes the Bounding box sometimes to fluctuate greatly which is obvious from the output video, in addition to that the threshold value is large in order to remove the false positives totally, but it's a tradeoff because sometimes the bounding bos is not covering the whole car in the scene.

I have two proposals which are the state of arts and very essential nowadays:

1) For the tracking problem, we can use Kalman Filter. This will increase the robustness of the algorithm greatly because it will reduce the fluctuating of the bounding box, so we will have a smooth box around the vehicle.
2) For the Detection problem, we are for sure can use CNN instead of the SVM classifier. This will facilitate the detection problem. In addition to that, it will reduce the processing done the video as we will feed the image directly as an inference to test the learnt model. The main algorithms used in the detection problems: RCNN, fast-RCNN, Faster-RCNN.