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. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Histogram of Oriented Gradients (HOG)

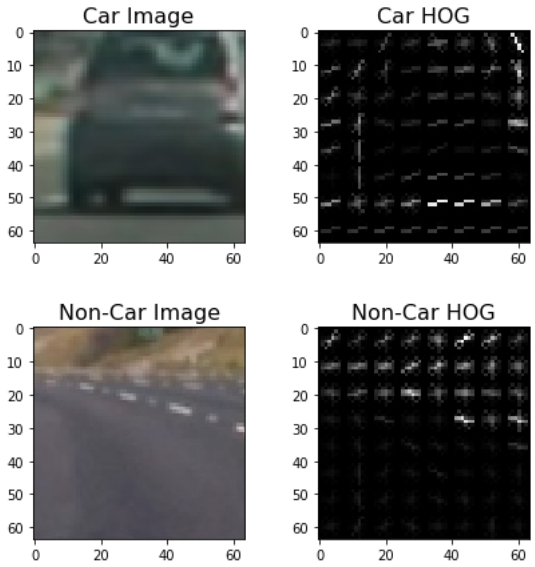
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this section is primarily under the Step 1 and Step 2 of the `Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier` section of the iPython notebook `P5_Vehicle_Detection_and_Tracking.ipynb`

I started by reading in all the `vehicle` and `non-vehicle` images. I used all the different database images that were provided to us. This is roughly 8700 images in each category. Here are some sample images.

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I tried several permutations on the properties of possible `color_space`, `orientations` and `hog_channels` which can be seen in the ipynb notebook under the second block of `Step 3. Train a Linear SVM classifier`.

Here is an example using the `HLS` color space and HOG parameters of `orientations=12`, `pixels_per_cell=(16, 16)` and `cells_per_block=(2, 2)`:



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

As mentioned in the above sections, I initially ran a set of permutations that I thought would be good to understand which broad color_space will be good for this task. The outcome of that test is:

Colorspace	Orientations	pix_per_cell	cell_per_block	hog_channel	spatial_size	hist_bins	Feature_extraction_time	Classifier_train_time
RGB	8	8	2	0	(16, 16)	16	46.42	12.52
RGB	8	8	2	1	(16, 16)	16	47.18	11.32
RGB	8	8	2	2	(16, 16)	16	46.14	11.72
RGB	8	8	2	ALL	(16, 16)	16	141.9	30.83
RGB	9	8	2	0	(16, 16)	16	63.48	12.35
RGB	9	8	2	1	(16, 16)	16	60.11	11.39
RGB	9	8	2	2	(16, 16)	16	57.89	12.64
RGB	9	8	2	ALL	(16, 16)	16	133.98	33.7
RGB	10	8	2	0	(16, 16)	16	64.34	13.65
RGB	10	8	2	1	(16, 16)	16	61.73	14.12
RGB	10	8	2	2	(16, 16)	16	66.98	14.07
RGB	10	8	2	ALL	(16, 16)	16	131.12	39.12
RGB	11	8	2	0	(16, 16)	16	71.25	15.51
RGB	11	8	2	1	(16, 16)	16	71.41	14.39
RGB	11	8	2	2	(16, 16)	16	70.44	14.89
RGB	11	8	2	ALL	(16, 16)	16	134.92	41.26
RGB	12	8	2	0	(16, 16)	16	184.69	23.28
RGB	12	8	2	1	(16, 16)	16	305.81	20.0
RGB	12	8	2	2	(16, 16)	16	308.21	20.09
RGB	12	8	2	ALL	(16, 16)	16	291.78	45.75
HSV	8	8	2	0	(16, 16)	16	47.26	12.36
HSV	8	8	2	1	(16, 16)	16	46.01	13.13
HSV	8	8	2	2	(16, 16)	16	46.52	10.71
HSV	8	8	2	ALL	(16, 16)	16	138.16	11.05
HSV	9	8	2	0	(16, 16)	16	69.22	14.06
HSV	9	8	2	1	(16, 16)	16	67.13	13.93
HSV	9	8	2	2	(16, 16)	16	62.71	11.84
HSV	9	8	2	ALL	(16, 16)	16	138.37	28.57
HSV	10	8	2	0	(16, 16)	16	72.51	15.06
HSV	10	8	2	1	(16, 16)	16	67.42	14.35

HSV	10	8	2	2	(16, 16)	16	69.44	12.62
HSV	10	8	2	ALL	(16, 16)	16	143.22	6.87
HSV	11	8	2	0	(16, 16)	16	76.75	16.88
HSV	11	8	2	1	(16, 16)	16	75.42	16.07
HSV	11	8	2	2	(16, 16)	16	75.54	13.9
HSV	11	8	2	ALL	(16, 16)	16	133.05	7.7
HSV	12	8	2	0	(16, 16)	16	182.61	21.63
HSV	12	8	2	1	(16, 16)	16	311.76	23.52
HSV	12	8	2	2	(16, 16)	16	306.19	19.42
HSV	12	8	2	ALL	(16, 16)	16	303.26	7.82
LUV	8	8	2	0	(16, 16)	16	50.19	10.63
LUV	8	8	2	1	(16, 16)	16	47.5	12.01
LUV	8	8	2	2	(16, 16)	16	48.36	14.91
LUV	8	8	2	ALL	(16, 16)	16	144.0	27.54
LUV	9	8	2	0	(16, 16)	16	68.47	11.25
LUV	9	8	2	1	(16, 16)	16	72.21	13.17
LUV	9	8	2	2	(16, 16)	16	65.93	14.95
LUV	9	8	2	ALL	(16, 16)	16	143.48	32.27
LUV	10	8	2	0	(16, 16)	16	71.6	11.77
LUV	10	8	2	1	(16, 16)	16	70.47	14.54
LUV	10	8	2	2	(16, 16)	16	69.97	16.03
LUV	10	8	2	ALL	(16, 16)	16	148.28	34.8
LUV	11	8	2	0	(16, 16)	16	80.28	12.67
LUV	11	8	2	1	(16, 16)	16	75.47	14.95
LUV	11	8	2	2	(16, 16)	16	76.95	17.58
LUV	11	8	2	ALL	(16, 16)	16	140.04	36.55
LUV	12	8	2	0	(16, 16)	16	197.89	21.11
LUV	12	8	2	1	(16, 16)	16	319.39	22.11
LUV	12	8	2	2	(16, 16)	16	328.19	23.61
LUV	12	8	2	ALL	(16, 16)	16	308.28	45.27
HLS	8	8	2	0	(16, 16)	16	49.9	12.76
HLS	8	8	2	1	(16, 16)	16	46.79	10.07
HLS	8	8	2	2	(16, 16)	16	46.79	14.91
HLS	8	8	2	ALL	(16, 16)	16	152.45	28.96
HLS	9	8	2	0	(16, 16)	16	68.04	15.15
HLS	9	8	2	1	(16, 16)	16	71.17	11.32
HLS	9	8	2	2	(16, 16)	16	67.48	16.47
HLS	9	8	2	ALL	(16, 16)	16	146.91	31.78
HLS	10	8	2	0	(16, 16)	16	75.62	15.53
HLS	10	8	2	1	(16, 16)	16	69.51	12.35
HLS	10	8	2	2	(16, 16)	16	72.66	19.16
HLS	10	8	2	ALL	(16, 16)	16	151.98	7.18
HLS	11	8	2	0	(16, 16)	16	81.13	18.88
HLS	11	8	2	1	(16, 16)	16	77.85	13.25
HLS	11	8	2	2	(16, 16)	16	81.17	18.98
HLS	11	8	2	ALL	(16, 16)	16	141.89	15.75
HLS	12	8	2	0	(16, 16)	16	196.22	23.4
HLS	12	8	2	1	(16, 16)	16	321.8	20.04

HLS	12	8	2	2	(16, 16)	16	327.79	25.68
HLS	12	8	2	ALL	(16, 16)	16	316.24	7.94
YUV	8	8	2	0	(16, 16)	16	50.51	9.71
YUV	8	8	2	1	(16, 16)	16	48.04	11.5
YUV	8	8	2	2	(16, 16)	16	48.46	12.82
YUV	8	8	2	ALL	(16, 16)	16	150.52	23.28
YUV	9	8	2	0	(16, 16)	16	67.29	10.71
YUV	9	8	2	1	(16, 16)	16	73.35	12.55
YUV	9	8	2	2	(16, 16)	16	69.02	15.42
YUV	9	8	2	ALL	(16, 16)	16	148.12	26.48
YUV	10	8	2	0	(16, 16)	16	73.58	11.78
YUV	10	8	2	1	(16, 16)	16	71.9	13.6
YUV	10	8	2	2	(16, 16)	16	72.19	17.46
YUV	10	8	2	ALL	(16, 16)	16	151.5	29.12
YUV	11	8	2	0	(16, 16)	16	81.49	13.29
YUV	11	8	2	1	(16, 16)	16	79.55	14.67
YUV	11	8	2	2	(16, 16)	16	80.27	17.14
YUV	11	8	2	ALL	(16, 16)	16	144.63	13.62
YUV	12	8	2	0	(16, 16)	16	221.48	20.82
YUV	12	8	2	1	(16, 16)	16	331.63	21.56
YUV	12	8	2	2	(16, 16)	16	344.98	25.13
YUV	12	8	2	ALL	(16, 16)	16	317.86	7.55
YCrCb	8	8	2	0	(16, 16)	16	50.21	9.95
YCrCb	8	8	2	1	(16, 16)	16	49.57	11.54
YCrCb	8	8	2	2	(16, 16)	16	48.54	13.53
YCrCb	8	8	2	ALL	(16, 16)	16	150.18	23.51
YCrCb	9	8	2	0	(16, 16)	16	70.16	10.99
YCrCb	9	8	2	1	(16, 16)	16	75.52	12.8
YCrCb	9	8	2	2	(16, 16)	16	70.57	14.76
YCrCb	9	8	2	ALL	(16, 16)	16	155.27	27.46
YCrCb	10	8	2	0	(16, 16)	16	83.06	12.07
YCrCb	10	8	2	1	(16, 16)	16	71.88	13.92
YCrCb	10	8	2	2	(16, 16)	16	72.51	15.82
YCrCb	10	8	2	ALL	(16, 16)	16	157.18	7.91
YCrCb	11	8	2	0	(16, 16)	16	83.31	13.36
YCrCb	11	8	2	1	(16, 16)	16	81.62	15.18
YCrCb	11	8	2	2	(16, 16)	16	82.89	17.29
YCrCb	11	8	2	ALL	(16, 16)	16	144.84	34.92
YCrCb	12	8	2	0	(16, 16)	16	194.02	22.44
YCrCb	12	8	2	1	(16, 16)	16	318.46	22.4
YCrCb	12	8	2	2	(16, 16)	16	329.38	24.18
YCrCb	12	8	2	ALL	(16, 16)	16	315.03	7.81

I finally settled down on the following parameters after running it through the entire pipeline to produce the outputs for each of the minor tweaks that I performed for the `YCrCb` color space.

Parameters:

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 11  # HOG orientations
pix_per_cell = 16 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
```

```
hist_bins = 24    # Number of histogram bins
spatial_feat = True
hist_feat = True
hog_feat = True
```

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 classifier based on the Linear SVC classifier using the `sklearn.svm` library. This section of the code where I pass in the training and validation examples to the classifier at a later point can be seen under the section that says `Step 3. Train a Linear SVM classifier.` in the `P5_Vehicle_Detection_and_Tracking.ipynb`

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?

For this section, I pretty much reused most of the lesson code that was provided to us.

There are 2 main approaches that I considered for this. One is the simpler approach where we initially calculate the number and all the possible windows in the image based on certain parameters (like `window_size` and `overlap`) and then search for vehicles in this set of windows.

The other approach is to 'predict' where the car can be in the next frame based on the current parameters that were provided to the Linear SVC classifier. This is also augmented with HOG sub sampling for all the 3 channels. All these parameters are flattened and passed to the classifier's predict function.

For the first approach, the code can be found under `helperfunctions.py` under the following methods:

```
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
                    xy_window=(64, 64), xy_overlap=(0.5, 0.5)):

def search_windows(img, windows, clf, scaler, color_space='RGB',
                                    spatial_size=(32, 32), hist_bins=32,
                                    hist_range=(0, 256), orient=9,
                                    pix_per_cell=8, cell_per_block=2,
                                    hog_channel=0, spatial_feat=True,
                                    hist_feat=True, hog_feat=True):
```

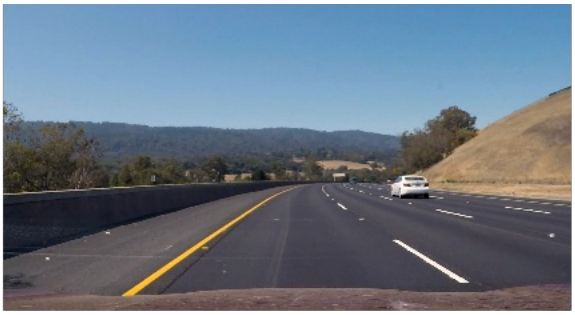
For the second approach, the code can be found in `helperfunctions.py` under the method:

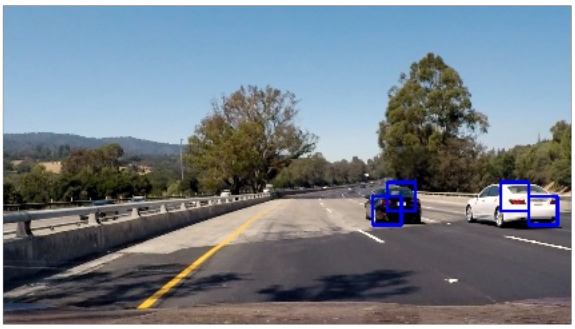
```
def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins):
```

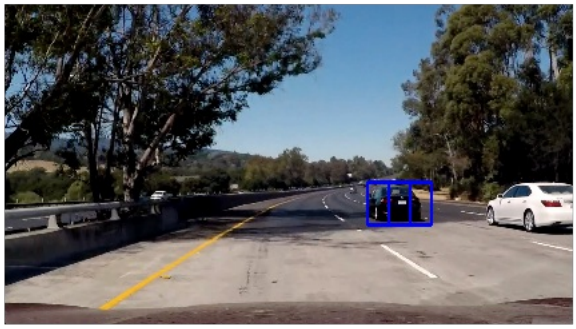
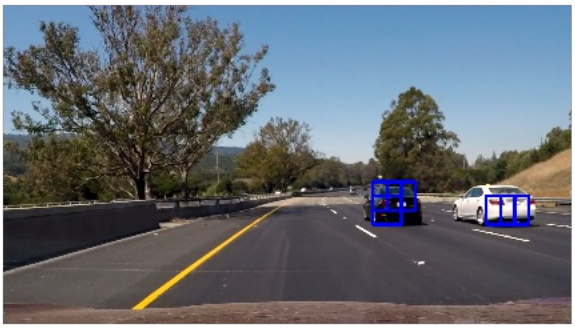
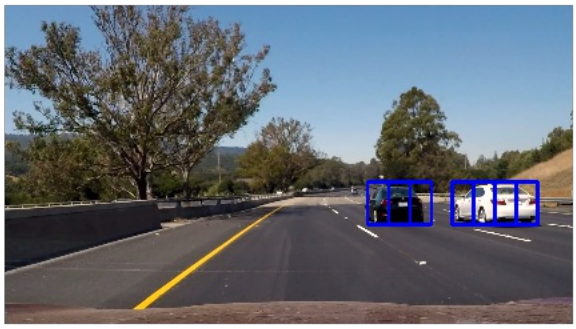
Once I decided that using the HOG Sub Sampling method gives me the better result, I went to further tweak the HOG parameters till I ended up with the ones mentioned above.

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?

I performed an analysis on which variant of the sliding window technique I would go with and then ended up with the HOG Sub sample methodology. Sample images from the analysis:





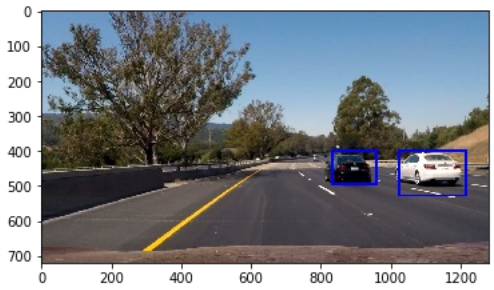





Once this w as done, I w ent ahead to implement a multi w indow boundary search w ith varying box sizes. This is to take benefit of the fact that the vehicles w hen far aw ay from the car w ill be smaller than w hen they are nearer to you. With this, I used the follow ing boundaries:

```
# Bounding window 1
y_start = 400
y_stop = 470
scale = 1.0

# Bounding window 2
y_start = 430
y_stop = 500
scale = 1.0

#Bounding window 3
y_start = 400
y_stop = 580
scale = 1.5

#Bounding Window 4
```

```
    y_start = 400
    y_stop = 530
    scale = 2.0

    #Bounding window 5
    y_start = 400
    y_stop = 550
    scale = 2.5

    #Bounding window 6
    y_start = 460
    y_stop = 660
    scale = 2.5
```

Sample output w ith the boxes mentioned above:



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.)

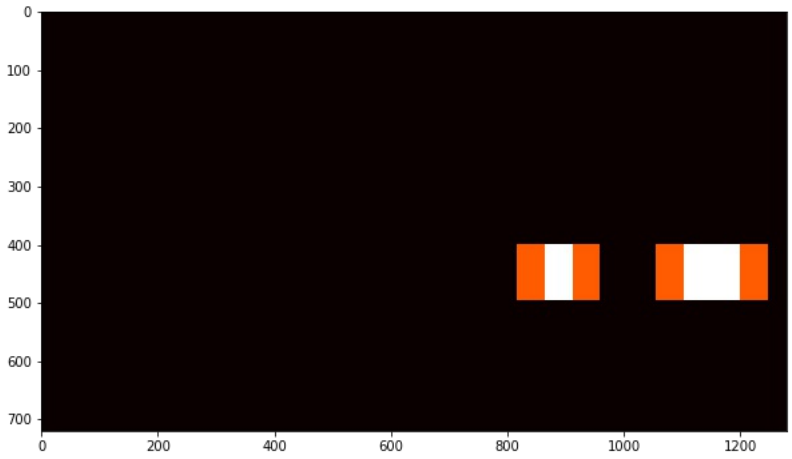
Here's a link to my video result

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.

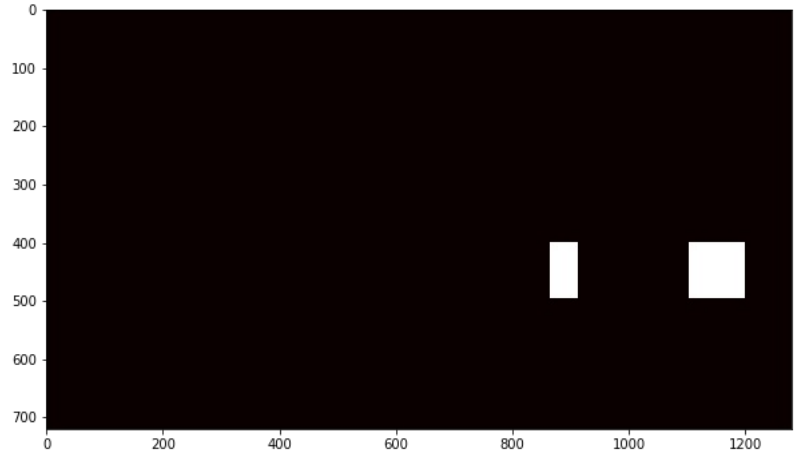
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Sample images for each stage of heatmap and overlapping boundary detection:

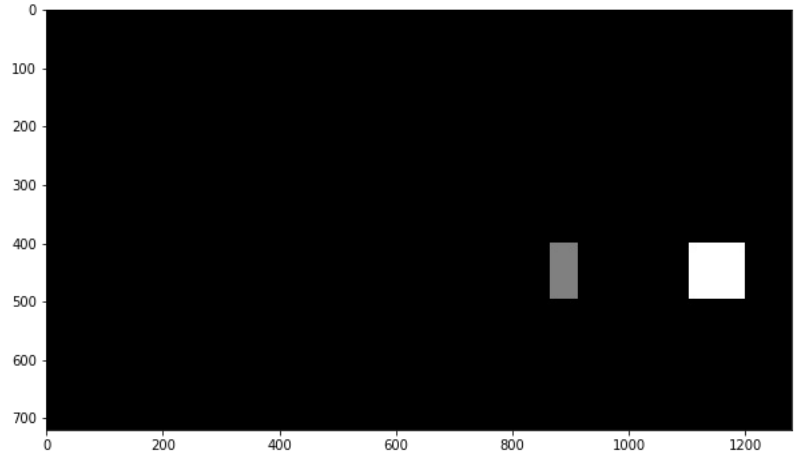
Raw heatmap image:



Threshold image:



Applying `measurements.label()`:



Final image:

With the help of the heatmaps that I had generated for each frame, I also saved the last `n` frames in a buffer of sorts. Code for this is present in the ipynb notebook under:

```
class FrameBuffer():
```

When a frame has hits on the vehicle detection, it is added to this frame buffer. If there were no hits for a given frame, the oldest one from the buffer is removed. This helped with the minor false positives that appear in one or two frames but not in all the frames. That way, I do not have to wait for the frame buffer to fill up to remove the false positive that was added.

The code also checks to ensure that the threshold for the heat map is done for at least half the current size of the frame buffer. This also helped reduce the false positives to quite an extent.

The code for the entire pipeline including all the false positive filters are in:

```
def process_image_v2(image):
```

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?

I spent quite sometime playing around with the parameters for the CV image detection portion of the code. The finetuning of the parameters is one thing that I think I still have to nail down. Even with the CV finetuning, the classifier being used has several parameters that I can play with, which I have not taken an advantage of in this project. This, when used properly, is something that I feel will yield me some better results.

As seen from the output video, there are some amounts of false positives being generated. Some of them, expecially in the left side of the video, is something that I would not determine as false positive as it did detect a big truck that was seen over the barricade. I can overcome this by adding a masking layer, but this I feel defeats the purpose of image detection. What if some idiot drives on the shoulder of the road when I am in the left most lane? I would want to know that.

Once again, there are several places where I think I can improve on this, but these are the few places that I can think of now. Searching about this online, I stumbled across YOLO Object detection paper that seems really interesting. This has quite a high speed of object detection that we can use here where we need faster analysis of the video. Currently my code works at 2-3 frames per second. Which is not optimal.