Vehicle Detection Project

1 Goals of this project

Objective of this project is to create a Python script that can detect cars from the Video Image using a trained Classifier model by following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Create a 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.
- Recreate the Video Stream with the output of pipeline

2 Write up/Readme

2.1 Readme

- Frames from Video is extracted and each image is passed to Pipeline function (Process_Image) and output images are written back as Video output using Write_Videofile function
- The program consists of two parts:
    - Training SVC Linear mode
    - Identifying cars using the Linear Model
- Training of model is processed by below two functions
    - Extract_data – This function takes set of JPEG files Cars and Non Cars provided by Udacity and converts them to features
    - Train Classifier - Trains the SVC with the extracted features
- Identifying cars is processed by Find_cars function.

- Used visualizations to check if images are loaded correctly

Below are the list of function/procedures with their implementation

Draw_boxes	Uses cv2. Rectangle to draw boxes
Add_heat	Adds heat map
Apply_threshold	Uses threshold to remove false positives
draw_labeled_bboxes	Apply labels on the image
draw_simple_chart	Takes two images and shows them in output screen with caption
extract_data	Extract features of training images
train_classifier	Training of classifier
find_cars	Utilized sliding window and identifies car
test_model_on_multi_images	Test model with multiple images for given start, stop and scale
tune_threshold_for_multi_images	A partial implementation of pipeline for tuning threshold

process_image	Full implementation of pipeline

2.2 Histogram of Oriented Gradients (HOG)

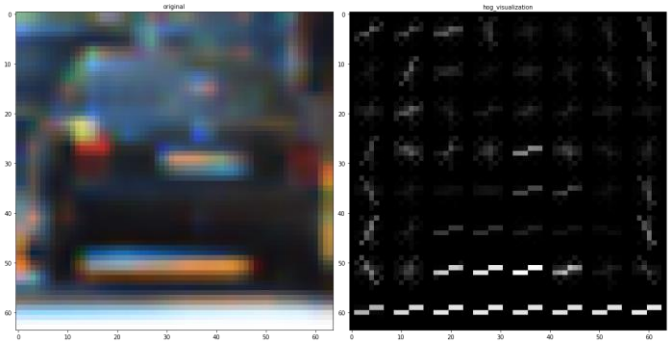
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

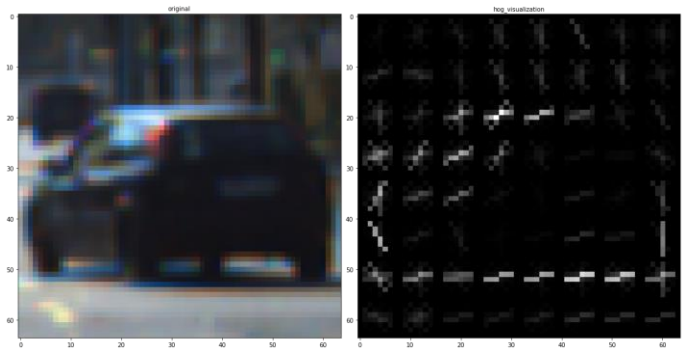
HOG features are extracted using extract_hog_features routine. Extract_data is used to call Extract_hog_Features to extract HOG features of cars and noncar training images into arrays car_features and notcar_features

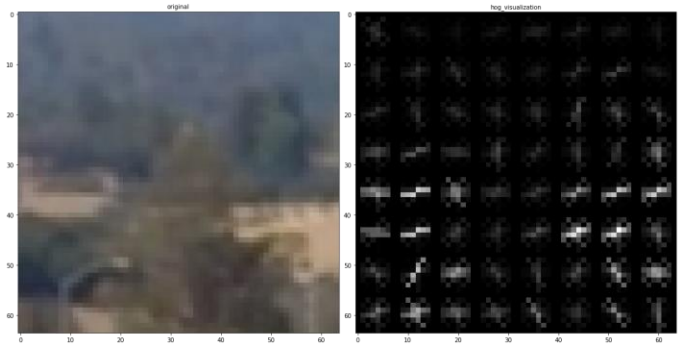
```
65
66
67 def extract_hog_features(imgs, cspace='RGB', orient=9,
68                           pix_per_cell=8, cell_per_block=2, hog_channel=0):
69     # Create a list to append feature vectors to
```

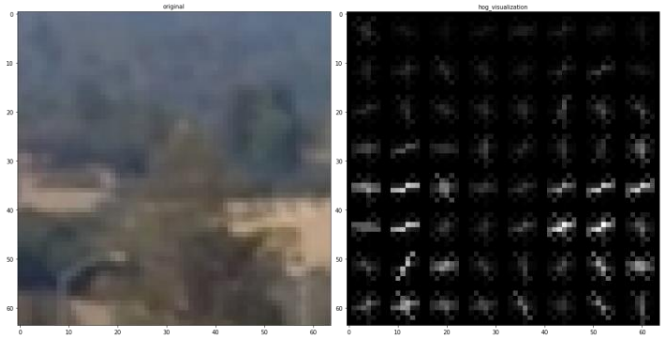
```
1 #---------------------------------------------Plotting and prediction routines ----------------------------
2
3 def extract_data(cars,notcars, cspace='RGB',orient=9,pix_per_cell=8,
4                   cell_per_block=2,hog_channel=0,spatial_size=(16,16),hist_bins=16,hist_range=(0, 256),
5                   spatial_feat=True, hist_feat=True, hog_feat=True):
6
```

I have added hog_visualization to create visualization of HOG images as given below









As you can see the HOG spaces are having different features for Cars and Not Cars and this is the basic concept behind identifying Cars and Non Car images

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

I created a setup of scripts as shown below that provides extracts features and train model for various parameters using various combination of CSPACE, ORIENT and Pix_per_Cell

- # run one time only
- cspace_list = ['HSV','LUV','HLS','YUV']

- orient_list = [9,10,11,12]
- pix_per_cell_list =[8,16]

```
# extract features and prepare data and train model
# run one time only

cspace_list = ['HSV','LUV','HLS','YUV']
orient_list = [9,10,11,12]
pix_per_cell_list =[8,16]

cell_per_block=2
hog_channel='ALL'
spatial_size=(32,32)
hist_bins=16
hist_range=(0, 256)
spatial_feat=True
hist_feat=True
hog_feat=True
extract_feature_type="hog"
```

This produced various results for multiple combination. I chose the combinations that gave top accuracy and then selected an optimal combination that provided the best result

```
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9438
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9438
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9458
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9375
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8417
cpsace- RGB orient- 8 hog_channel- 0 pix_per_cell 32 spatial_size (32, 32) accuracy 0.8646
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9562
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9562
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9333
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9271
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8833
cpsace- RGB orient- 8 hog_channel- 1 pix_per_cell 32 spatial_size (32, 32) accuracy 0.8833
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9438
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9396
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9417
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9521
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8479
cpsace- RGB orient- 8 hog_channel- 2 pix_per_cell 32 spatial_size (32, 32) accuracy 0.8458
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 8 spatial_size (16, 16) accuracy 0.9583
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 8 spatial_size (32, 32) accuracy 0.9583
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 16 spatial_size (16, 16) accuracy 0.9562
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 16 spatial_size (32, 32) accuracy 0.9604
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 32 spatial_size (16, 16) accuracy 0.9083
cpsace- RGB orient- 8 hog_channel- ALL pix_per_cell 32 spatial_size (32, 32) accuracy 0.8854
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9521
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9521
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9458
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9229
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8354
cpsace- RGB orient- 9 hog_channel- 0 pix_per_cell 32 spatial_size (32, 32) accuracy 0.85
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9438
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9583
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9312
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9292
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8625
cpsace- RGB orient- 9 hog_channel- 1 pix_per_cell 32 spatial_size (32, 32) accuracy 0.8604
```

```
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 8 spatial_size (16, 16) accuracy 0.9562
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 8 spatial_size (32, 32) accuracy 0.9562
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 16 spatial_size (16, 16) accuracy 0.9104
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 16 spatial_size (32, 32) accuracy 0.9396
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 32 spatial_size (16, 16) accuracy 0.8583
cpsace- RGB orient- 9 hog_channel- 2 pix_per_cell 32 spatial_size (32, 32) accuracy 0.8396
cpsace- RGB orient- 9 hog_channel- ALL pix_per_cell 8 spatial_size (16, 16) accuracy 0.9604
```

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).

Training of classifier is done using Train_classifier with the Training data obtained from extract_cars. The svc mode is saved into a pickle file so that next time the training need not be done again.

```
X_train,X_test,y_train,y_test = extract_data(cars,notcars, cspace,orient,pix_per_cell,
                                             cell_per_block,hog_channel,
                                             spatial_size,hist_bins,hist_range,
                                             spatial_feat,hist_feat, hog_feat)

svc, accuracy = train_classifier(X_train,X_test,y_train,y_test)

print("length of X_train",len(X_train))
print("length of X_test",len(X_test))
print("length of Y_train",len(y_train))
print("length of Y_test",len(y_test))



acc = 100 * accuracy
print("accuracy is", '%.2f'  % acc,"%    cpsace " ,cspace,"     orient-",orient,"     hog_channel-",hog_channel,"     pi
     "     spatial_size",spatial_size)
```

```
#Load svc
with open('classifier.pkl', 'rb') as fid:
    svc = pickle.load(fid)

```

I tested the effectiveness of the model using the procedure “test_model_on_multi_images” with a predefined combination like start, stop, cspace etc

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 re-used find_cars from Project work out and then made few changes to it. I first used function to check whether boxes are identified correctly. I tried various combination of start, stop and threshold to check if cars are identified.

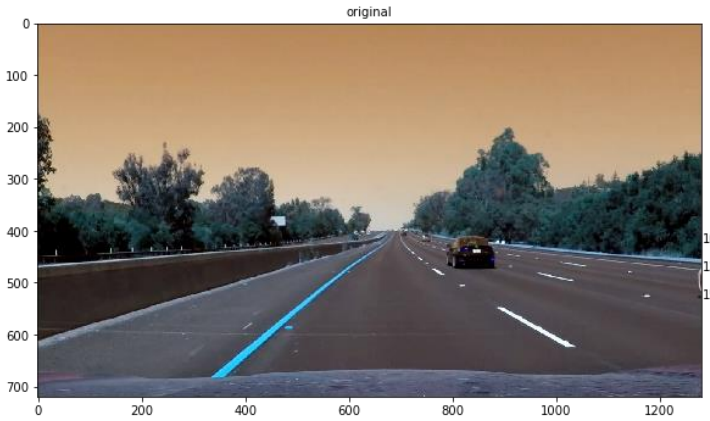
Then I used tune_threshold_for_multi_images and fed saved images from test and sample video to fine tune the threshold. The routine uses a multiple combination of start stop for both X and Y axis with various combination of scale.

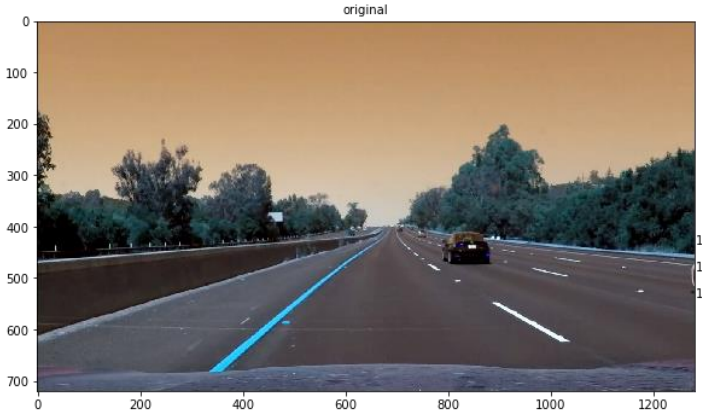
2. Show some examples of test images to demonstrate how your pipeline is working.

Finally, I settled for the below parameters that provided better results. You can see below few output iamges. I also fine tuned threshold to avoid false positives

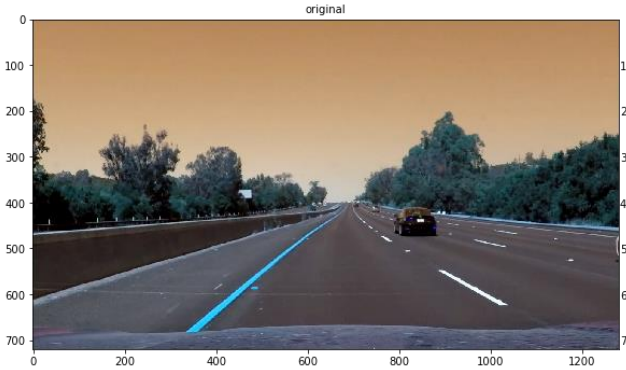
- cspace='HSV'
- orient= 8 #9
- pix_per_cell= 6# 8
- cell_per_block=2 #2
- hog_channel='ALL'
- spatial_size=(16,16)
- hist_bins=8
- hist_range=(0, 256)

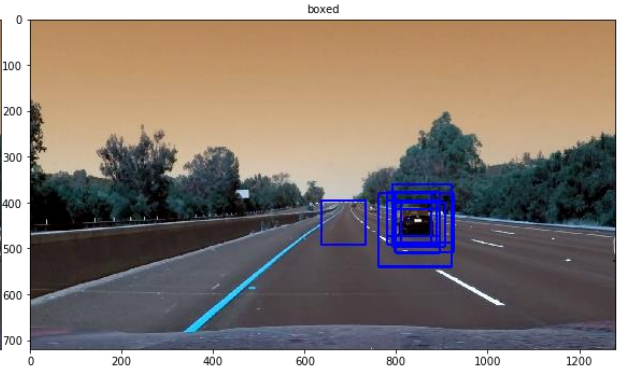
Below is example of a original image vs clipped image for various X start, stop and Ystart and Y stop

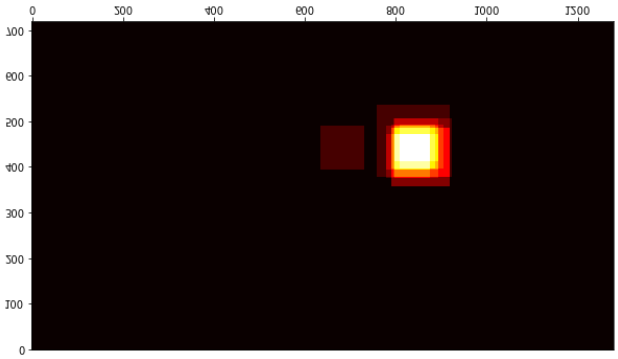


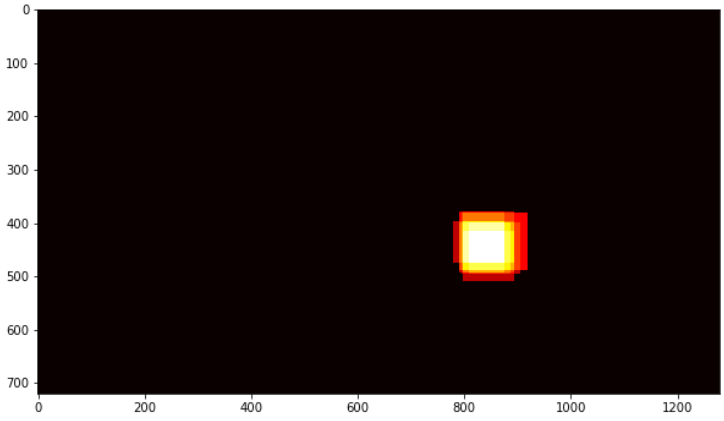

original

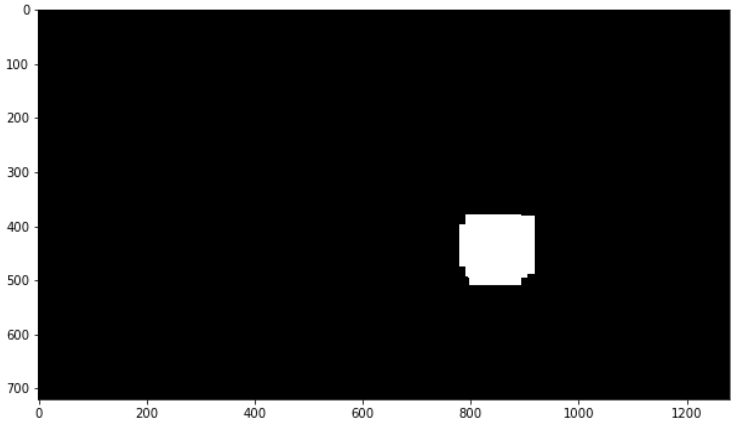

scale1.5xstart600xstop1100ystart360ystop510

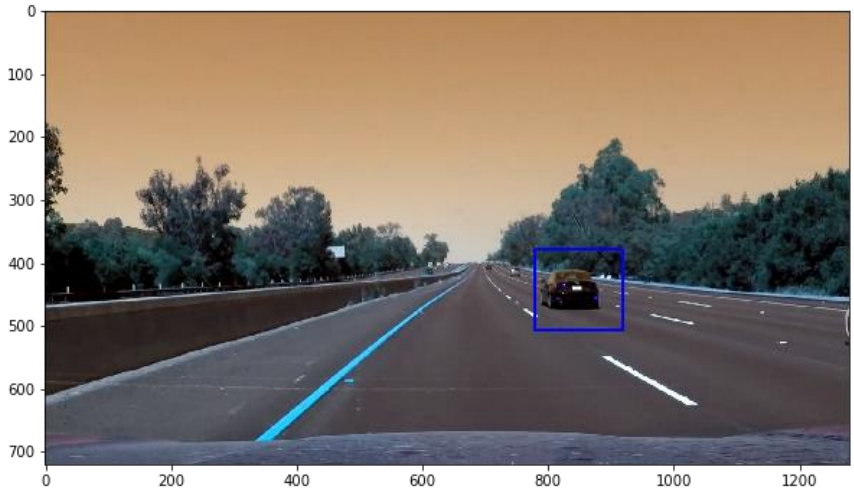

original


boxed




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


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


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

What did you do to optimize the performance of your classifier?

Below are few things that I did to optimize the whole performance:

1. used various combination of stop, stop and scales. I started with a smaller scale and then slowly increased. I also used X start and stop to avoid cars to be detected on unwanted places

```
        #print("ygap",np.max(nonzeroy) - np.min(nonzeroy))
        if ((np.max(nonzerox) - np.min(nonzerox)) > 30) and  ((np.max(nonzeroy) - np.min(nonzeroy)) > 30):
            bln_found = True
            bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))
            bboxes_list.append(bbox)
        #else:
```

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.)

Video is uploaded

In [58]:
```
rect = rectangles()
debug_flag = False
output_flag = False
test_out_file = 'project_video_lane_out.mp4'
clip_test = VideoFileClip('project_video_lane.mp4')
clip_test_out = clip_test.fl_image(process_image)
%time clip_test_out.write_videofile(test_out_file, audio=False)
```

```
[MoviePy] >>>> Building video project_video_lane_out.mp4
[MoviePy] Writing video project_video_lane_out.mp4
100%|[redacted]| 1260/1261 [00:43<00:00, 28.66it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: project_video_lane_out.mp4

CPU times: user 25.8 s, sys: 2.88 s, total: 28.7 s
Wall time: 44.9 s
```

In [ ]:
```
```

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 stored the identified boxes in a class rectangle. This class helps to perform below functions
    1. Saves identified boxes so that the same box can be cached. Get_bboxes and save_bboxes are used for this
    2. Check_scan_status helps ot cache the image. The boxes are identified for first image and then same boxes are cached for n count of image. After this again the boxes are calculated for next image and so on.
    3. This skipping is increasing performance and also flickering of boxes. Also there should be no issue as 20 frames are generated before any significant movement can be detected
    4. If anytime there are no boxes detected, then boxes are cached for a predefined count
- Also there were many false positives which I avoided by tuning threshold and also ensuring that boxes with uneven sizes are ignored

Below are the places where I have implemented these in code

```python
class rectangles():

    def __init__(self):
        self.bbox_list = []
        self.count = 0
        self.buff_count = 0
        #print("self initialized")

    def save_bboxes(self,bboxes=[], *args):
        #print("saved box",bboxes)
        del self.bbox_list
        self.bbox_list = bboxes

    def get_bboxes(self,bboxes =[], *args):
        return self.bbox_list

    def check_first_time(self):
        #print("checking first time")
        #print("self.count",self.count)
        return (self.count == 0)

    def inc_count(self):
        self.count += 1

```

```
    def check_scan_status (self,count):
        #print("check reset flag running")
        #print("self.count",self.count)
        #print("self.count mod",self.count % count)
        self.count += 1
        if ((self.count % count) == 0 ):
            return True
        else:
            return False

    def check_empty_status(self,count,bboxes=[],*args):
        if debug_flag == True:
            print("entering check_empty_Status")
            print(bboxes)
            print(len(bboxes))

        for bbox in bboxes:
            if len(bbox) > 0 :
                bln_empty = False
                return False

        if debug_flag == True:
            print("emtpy frame found")
        self.buff_count += 1
        if (self.buff_count <= count):
            if debug_flag == True:
                print("buffered")
            return True
        else:
            if debug_flag == True:
                print("threshold exceeded. empty frame not buffered")
```

I created a simple python script Image extractor that can extract a portion of video into images. I tested the pipeline using project video and then identified the places where there were false positive or no identification. I adjusted threshold and start and stop of X and Y axes to avoid these.

With this I was able to reduce the false positives less than 5 .

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?

Following are the problems:

- My project is fine tuned for this video. There could be many false positives or no identifications for other videos without fine tuning.
- Also I used caching to avoid frames where there were no cars detected and used position of previous images.
- Also shadows are creating challenge in identification.
- Cars moving in the slowest lane at a far distance is also creating issues as these cars overlaps with adjacent things like hills, tress etc and result in issue and as a result these are not detected