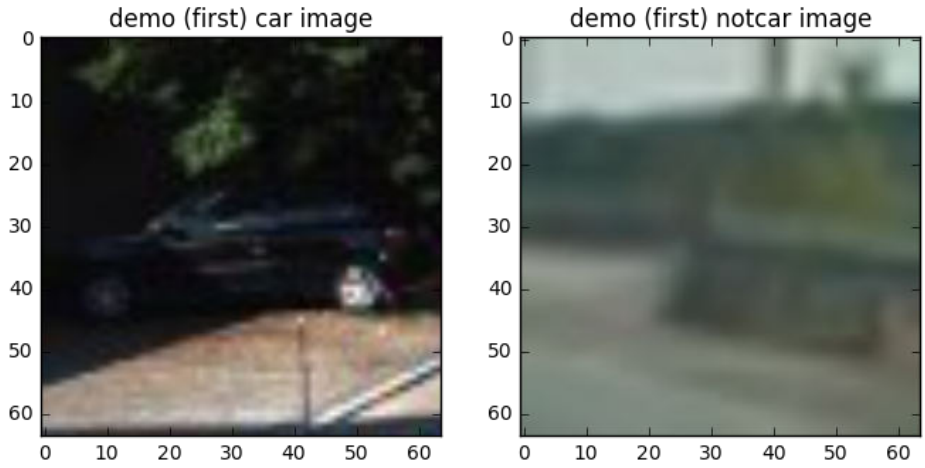
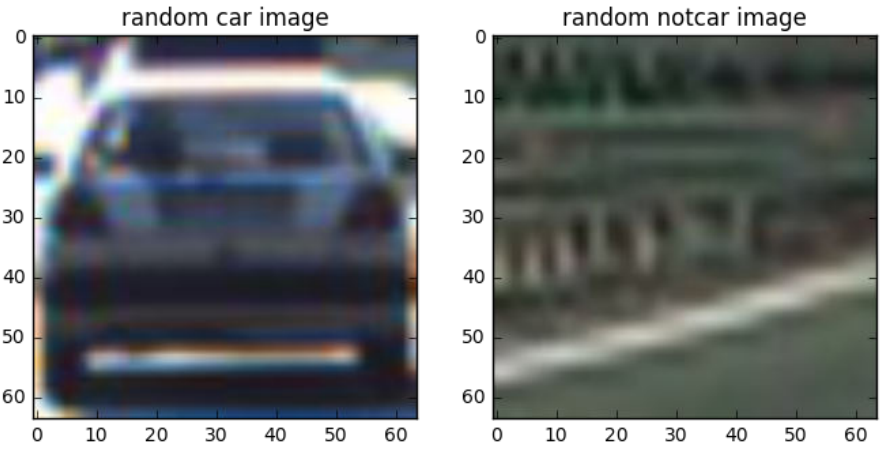
Project5 –-Vehicle Detection Project

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.

First, I have imported images stored in the computer so that I can run SVM classifier later on to classify images. I have used Udacity small dataset of images. I will import Full dataset later on. Although I could have imported full dataset right now, but I have used small dataset to visualise images. As smallset was available, so I was trying to use it as well. Here are the first **car and notcar** images from this dataset. Also **random car and notcar** images are shown below.





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 the **lines 95-200 of the file called Project5-vehicleDetection.py**.

Main code to get hog parameters for visualisation=True is below-

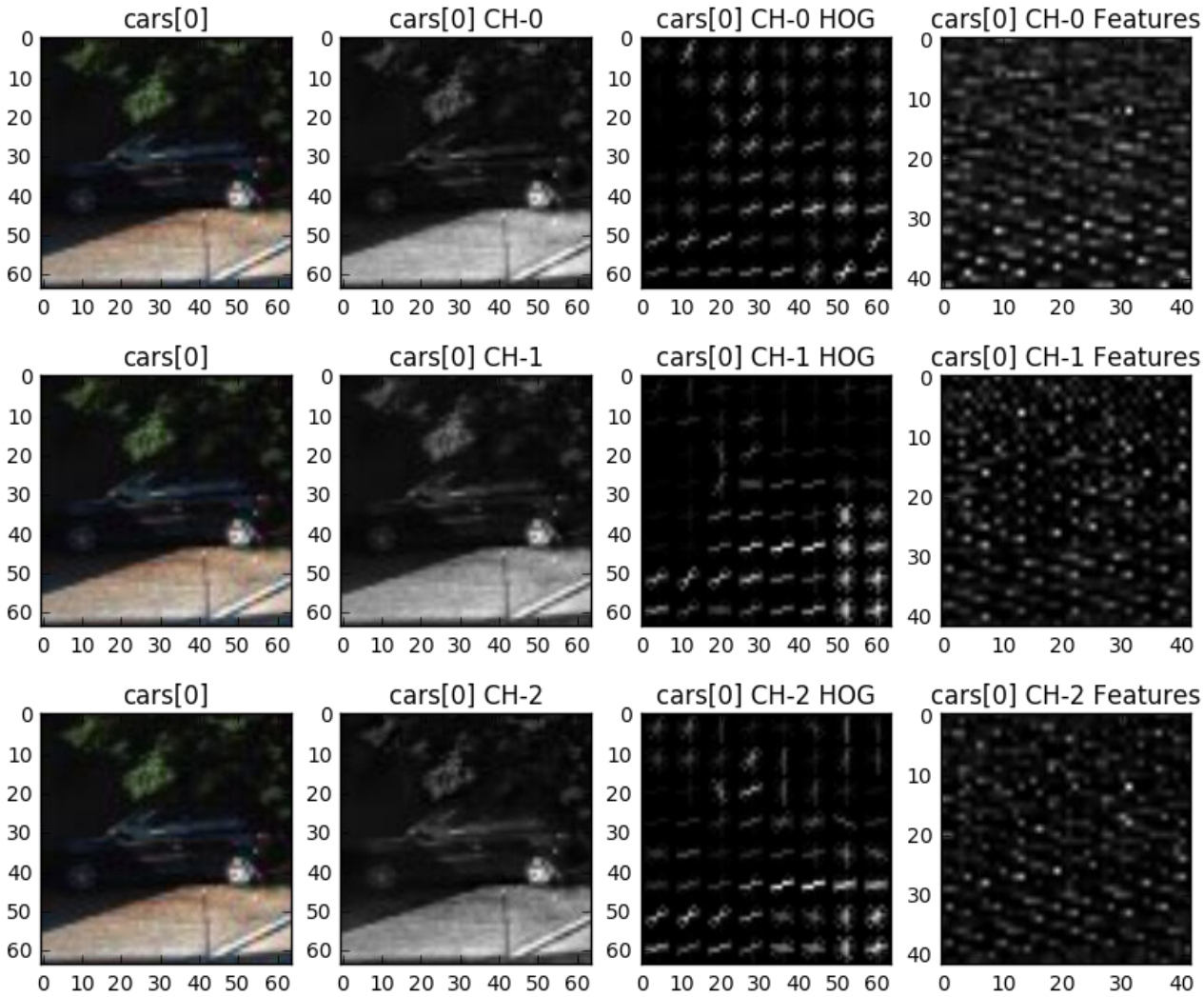
```
def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                     vis=False, feature_vec=True):
    # Call with two outputs if vis==True
    if vis == True:
        features, hog_image = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  cells_per_block=(cell_per_block, cell_per_block),
                                  visualise=vis, feature_vector=feature_vec, transform_sqrt=True)
        #print('vis=true, feature & image')
        return features, hog_image
    # Otherwise call with one output
    else:
        features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block),
                       visualise=vis, feature_vector=feature_vec, transform_sqrt=True)
```

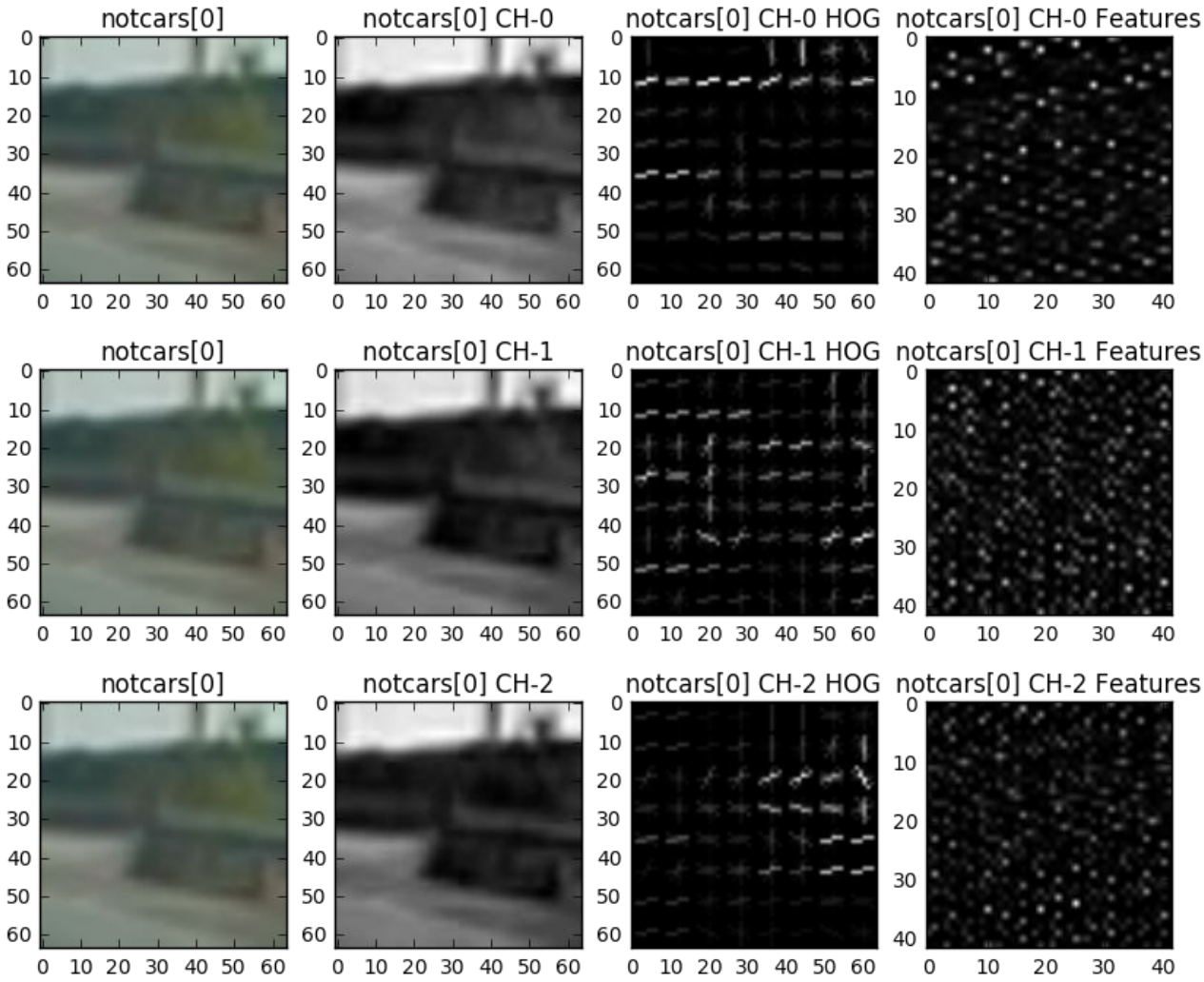
2. Explain how you settled on your final choice of HOG parameters.
I tried various combinations of parameters and...

I tried various combination of parameters and after doing some trial and run, I finally setlled for these parameters as they were showing decent result and quick processing-

```
colorspace = 'YCrCb'
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
# generate the hog_image as well
```

Here is an example using the YCrCb color space and HOG parameters of orient=9, pixels_per_cell=(8,8), cell_per_block=(2,2), hog_channel=ALL





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).

SVM accuracy attained was **more than 98.5% on test set**

I trained a linear SVM using..

Support Vector Machines technique has been used to train the classifier and Features has been scaled **using StandardScaler**

```
# #Create an array stack of feature vectors

X = np.vstack((car_features_now, notcar_features_now)).astype(np.float64)

# Fit a per-column scaler

X_scaler = StandardScaler().fit(X)

# Apply the scaler to X

scaled_X = X_scaler.transform(X)



# Split up data into randomized training and test sets

rand_state = np.random.randint(0, 100)

X_train, X_test, y_train, y_test = train_test_split(

    scaled_X, y, test_size=0.2, random_state=rand_state)



# Use a linear SVC (support vector classifier)

svc = LinearSVC()

# Check the training time for the SVC

t=time.time()

# Train the SVC

svc.fit(X_train, y_train)
```

Sliding Window Search

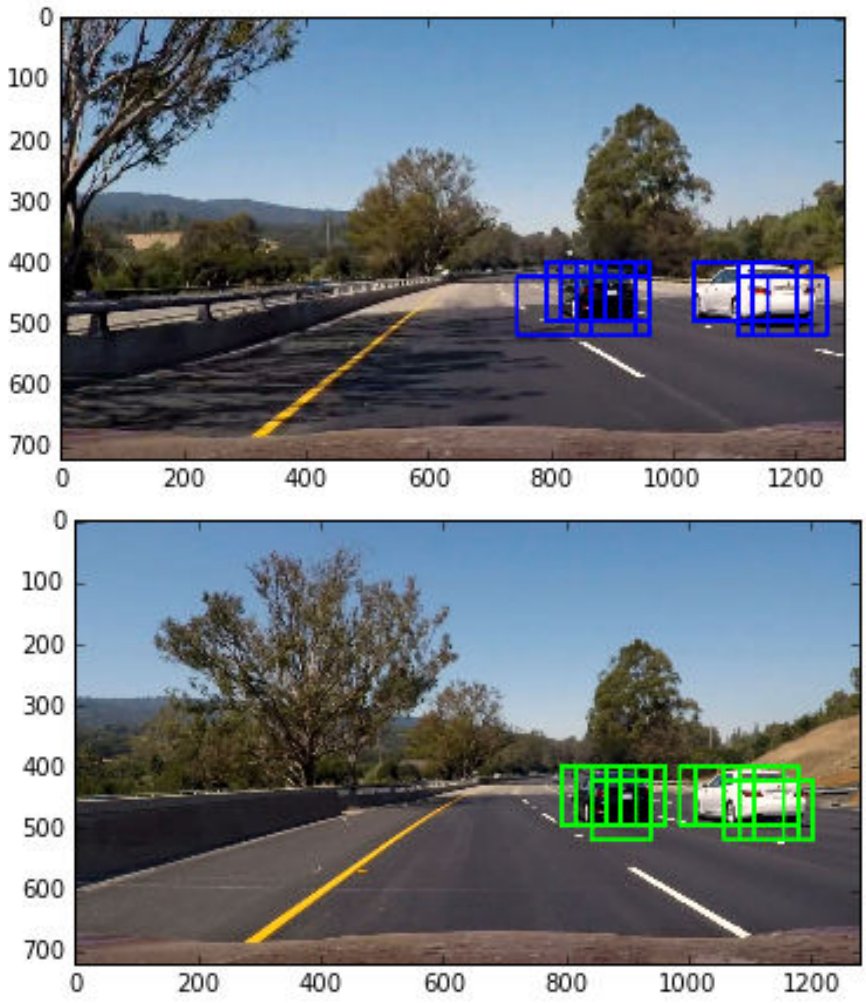
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

After trying for various scales combinations of **2.0, 1.75, 1.6,1.5, 1.4,1.25, 1.0,0.75,0.5,** I found that scale=1.5 was performing best. For any other scale which I chose there has been lot of false positives so I avoided using all other scales. Lines(530-660)

Ultimately I searched on one scale using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

```
spatial_size = (32, 32)  # (32,32)

hist_bins = 16  # 16

spatial_feat=True

hist_feat=True

hog_feat=True
```

Here are some example images:



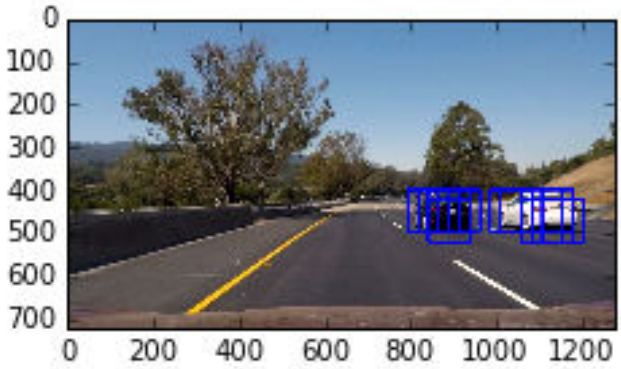
Variables values are stored and restored as pickle data **(lines- 1000-1030)**

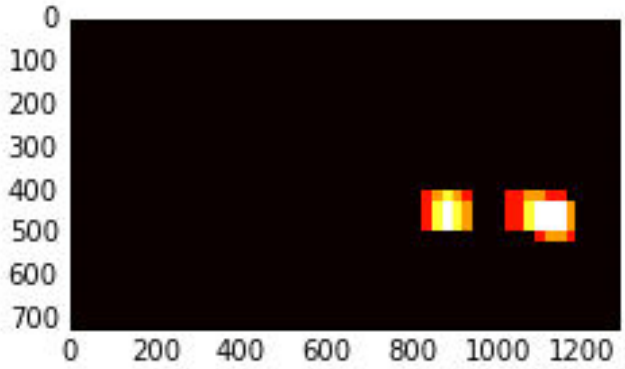
```
# # Restroing the saved pickle data

veh_pickle = pickle.load( open("saved_veh_det_pickle.p", "rb" ) )

svc1 = veh_pickle["svc"]
```

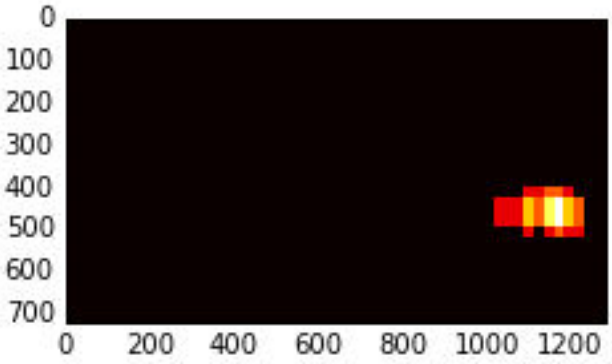
```
X1_scaler = veh_pickle["X_scaler"]

colorspace = veh_pickle["colorspace"]

orient = veh_pickle["orient"]

pix_per_cell = veh_pickle["pix_per_cell"]

cell_per_block = veh_pickle["cell_per_block"]

spatial_size = veh_pickle["spatial_size"]

hist_bins = veh_pickle["hist_bins"]

spatial_feat = veh_pickle["spatial_feat"]

hist_feat = veh_pickle["hist_feat"]

hog_feat = veh_pickle["hog_feat"]
```

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.

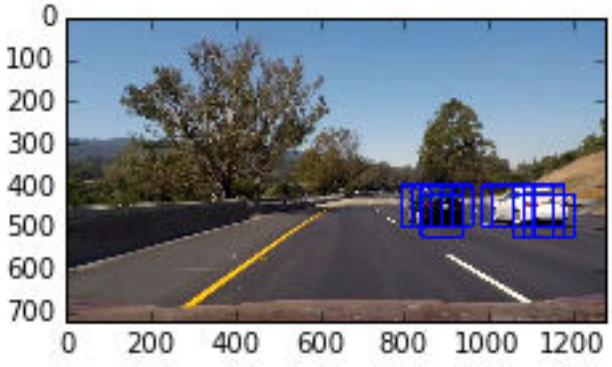
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions (lines – 770-840, lines-660-675). I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.
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:

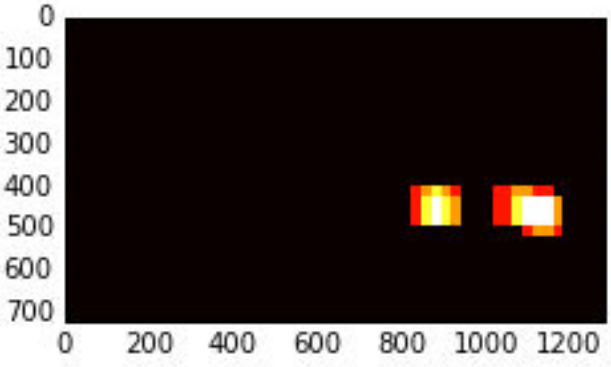




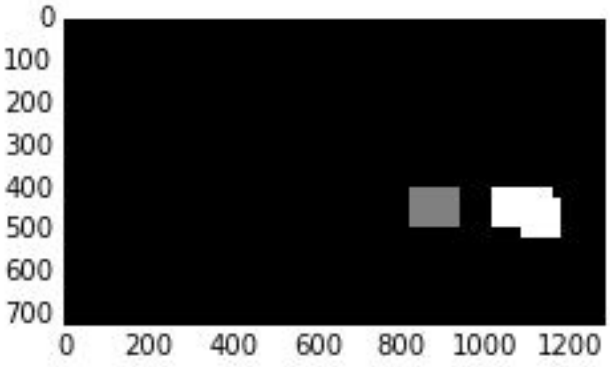


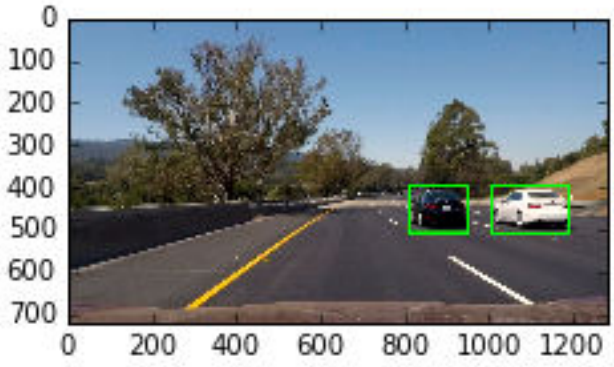
Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from last added frames:





Here the resulting bounding boxes are drawn onto the last frame in the series:





Example of test image to demonstrate how your pipeline is working.

At Last, (lines- 1080 - 1154) various videos are imported which were provided by UDACITY and final function (combining all other functions) is performed to obtain final resultant video with annotated vehicle detection. Most of the part of importing videos if from code of project 1 itself, hence self explanatory. Result of images and videos are uploaded with the project.

GIF of test video solution---

Also **youtube links of videos** are provided below-

Project Video Solution without skipping frames-

https://www.youtube.com/watch?v=jp1jREvQU0w

Project Video Solution with skipping frames-

https://www.youtube.com/watch?v=XUVyZbpv2TE

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?

The problems faced by me in project video was to chose appropriate scales range so as to avoid false positives but simultaneously not to loose information about car present in frame or not. Also I had to do a lot of trial and run to find appropriate value of threshold for cars and how many heatmaps of previous frames to add up.

The pipeline takes some time to process if large no. of scales are chosen. It looks to me that at various lighting conditions, I have to employ more scales values, which will take real time video to take more time to process. Hence, it don't look like a perfect solution for real time video. Pipeline may fail at such extreme lightning or weather conditions.

I think Deep Learning techiniques (which we learned in Project 2) provide better solution to detect cars present or not for real time applications. To make pipeline more robust, I think we should employ Deep learning techniques too with this method, which can provide better solution.