Histogram of Oriented Gradients (HOG)

Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.

Resubmission 1

In this resubmission I have transferred the following functions to an .py file to have shorter and readable code in jupyter notebook.

- **bin_spatiall**
- **convert_color**
- **color_hist**
- **get_hog_features**

The name of python file is features.py and it is imported in Jupyter notebook.
import features

Histogram of Oriented Gradient

I have a function for extracting the Histogram of Oriented Gradient with following header which has been called under "4. Extracting the Histogram of Oriented Gradient (scikit-image HOG)" cell.

def get_hog_features(myimg, orient, pix_per_cell, cell_per_block,vis=False, feature_vec=True):

The next figure is the visualization of the output of the *get_hog_features* function.

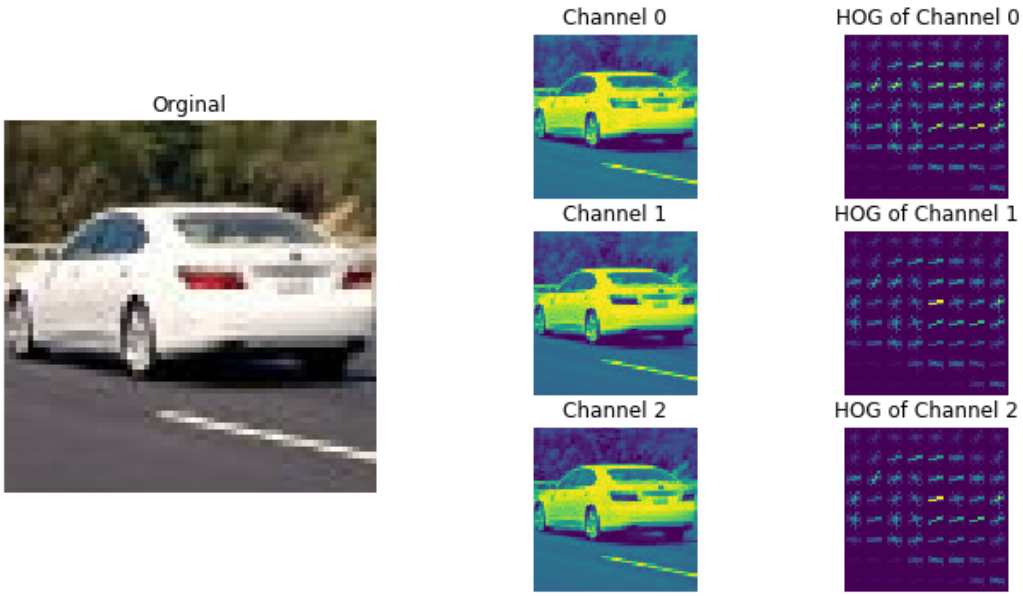


Figure 1: orient=9, pix_per_cell=8, cell_per_block=2

The main function which is used to extract the image HOG is hog function as in training codes. The get_hog_features function has been used in two different part of this project.

- Under “Extract Dataset's features” cell: via calling extract_features function from training materials.

```
car_features = extract_features(cars, cspace = color_space,
                        spatial_size = spatial_size,
                        hist_bins = hist_bins,
                        hist_range = hist_range)
notcar_features = extract_features(notcars, cspace = color_space,
                          spatial_size = spatial_size,
                          hist_bins = hist_bins,
                          hist_range = hist_range)
```

- Under “Function for processing each frame” cell: via calling find_cars function.

```
predicated_windows = find_cars(image, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell,
cell_per_block, spatial_size, hist_bins)
```

As we can see in the two previous code snippets we have variables which determine which feature must be considered for training and for vehicle finding as well. These variables have been defined under “Global values” cell.

Variable name	Value	Description
spatial_feat	True or False	True means spatial feature is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts.
hist_feat	True or False	True means Histogram of Color is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts.
hog_feat	True or False	True means Histogram of Oriented Gradient is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts.

About orient, pix_per_cell, cell_per_block variables

I have tested the “Extracting the Histogram of Oriented Gradient (scikit-image HOG)” code snippet not only for 8 pixels per cell but also for 4 and 2 pixels per cell and all of them with 2 cells per block.

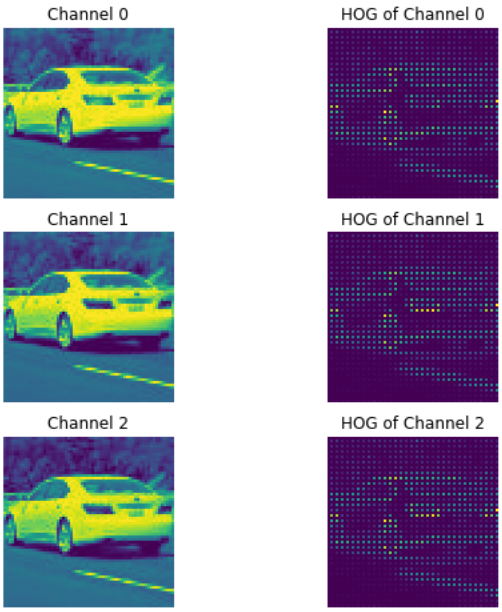


Figure 2: orient=9, pix_per_cell=2, cell_per_block=2

For example, this figure at the left side is a sample of pixel per cell 2 and cell per block 2.

The pattern of vehicle is clearer than the previous one, but the performance is worse than pix_per_cell=8, cell_per_block=2

Therefore, I have developed my project with orient = 9, pix_per_cell=8 and cell_per_block=2 which are defined and assigned under “Global values” section in Jupyter Notebook.

About color space variable

I have also tested all the color spaces which have been defined in project as shown in following figure.

Figure 3: RGB color space

Figure 4: HSV

Figure 5: LUV

Figure 6: HLS

Figure 7: YUV

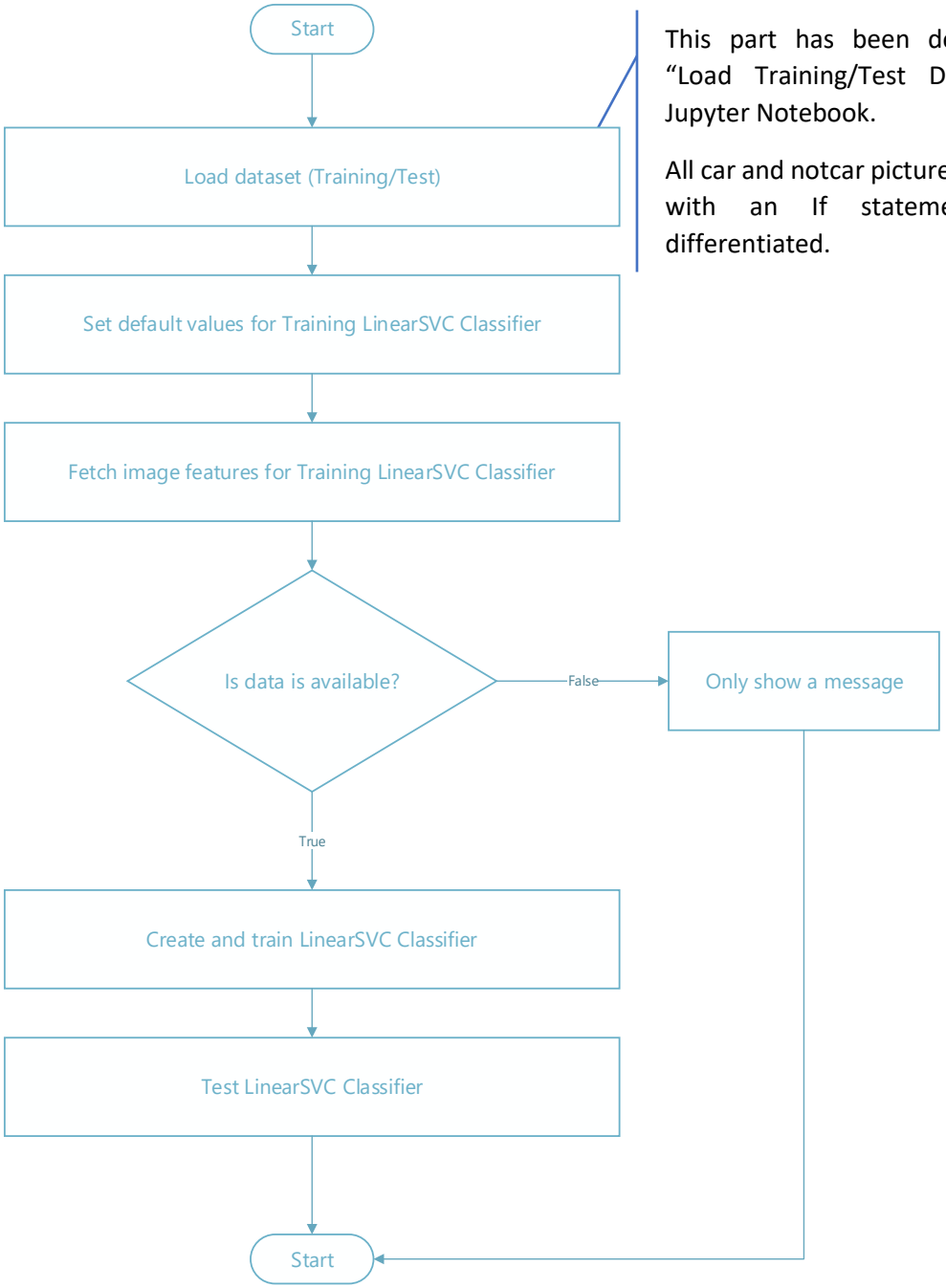
Figure 8: YCrCb

But I have decided to use YCrCb color space.

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

The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.

In the following figure we have the flowchart of my training and test Linear SVC Classifier mechanism.



This part has been developed under "Load Training/Test Dataset" cell in Jupyter Notebook.

All car and notcar pictures are loaded but with an If statement they are differentiated.

The training code is started from cell "Global values", where I have defined the variables and values for training the model as in following.

```
color_space = 'YCrCb'
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL"
spatial_size = (16, 16)
hist_bins = 128
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
hist_range = (0, 256)
```

In “Load Dataset” cell I have defined two loops to make two lists of vehicle and non-vehicle image names for training and testing the linear SVC model as shown in the following code snippet,

```
images = glob.iglob('./all/vehicles/**/*.png',recursive=True)
cars = []
notcars = []
for image in images:
   cars.append(image)
images = glob.glob('./all/non-vehicles/**/*.png',recursive=True)
for image in images:
   notcars.append(image)
```

After preparing the vehicle and non-vehicle lists, I extract the feature/s of the car and non-car images with *extract_features* function in cell “Extract Dataset's features” and create an array stack of the extracted features to pass to *StandardScaler().fit()* function as developed in “Train Linear SVC(support vector classifier) Classifier” cell.

StandardScaler().fit() fits the scale per-column and I apply the calculated scaler to the created array stack as shown below.

```
# Scale the feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)

# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))

# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)

# Apply the scaler to X
scaled_X = X_scaler.transform(X)
```

After transforming the data, they must be split into two Train and Test dataset and I have developed a random logic to calculate a random percentage for splitting the dataset as shown in the following code snippet.

```
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)
```

As next step the model is trained and as I have explained in the yellow box. The optimization in this step is the C parameter of the LinearSVC function.

Resubmission 1

Another point that I have considered in this resubmission is the fine tuning of the C parameter of the LinearSVC classifiert as following:

```
svc = LinearSVC(C=0.0005)
```

```
svc = LinearSVC(C=0.0005)
svc.fit(X_train, y_train)
```

Although it wasn't requested in the project to save the model but I have developed a cell in Jupyter notebook to save the mode as I have explained in the bellow yellow box.

Resubmission 1

Another new logic that I have developed in code is saving the variables and model because it was not time efficient for me to run the whole code each time that I wanted to develop the steps after training the mode like developing find_cars function. This logic is developed in "Save model" cell.

```
obj = {
      "svc": svc,
      "scaler": X_scaler,
      "orient" : orient,
      "pix_per_cell": pix_per_cell,
      "cell_per_block": cell_per_block,
      "spatial_size": spatial_size,
      "hist_bins": hist_bins,
      "color_space": color_space,
      "hog_channel": hog_channel,
      "spatial_feat": spatial_feat,
      "hist_feat": hist_feat,
      "hog_feat": hog_feat,
      "hist_range":hist_range
   }
pickle.dump(obj, open('svc_pickle.p', 'wb'))
```

And in "Read model" cell there is a logic for reading/reloading the saved variables and model.

Sliding Window Search

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?

A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.

The sliding, scaling and overlapping are done in find_cars function in "Using LinearSVC Classifier to detect vehicles" cell. I'll explain about each one in the following.

Overlapping

As it has been mentioned in training materials instead of the overlapping we can use the following code line.

```
cells_per_step = 2  #Instead of overlap, define how many cells to step
```

I have trained the code with different values. In case of better value, I didn't have efficient running time. Therefore, I decided to have 2 cells per step overlapping because the result was satisfactory, and the result wasn't long.

sliding

The sliding window mechanism, which I have used is from training materials.

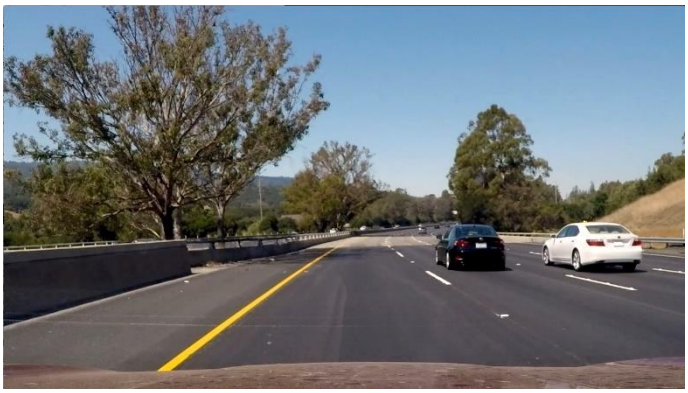
Scale

In “Read model” cell I have considered 7 different scales. Because vehicles in different distances have different scales.

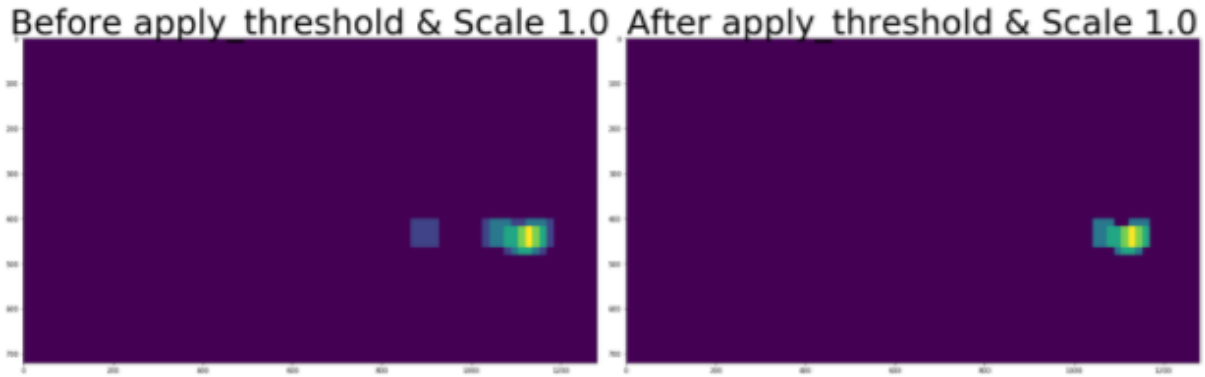
Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?

Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)

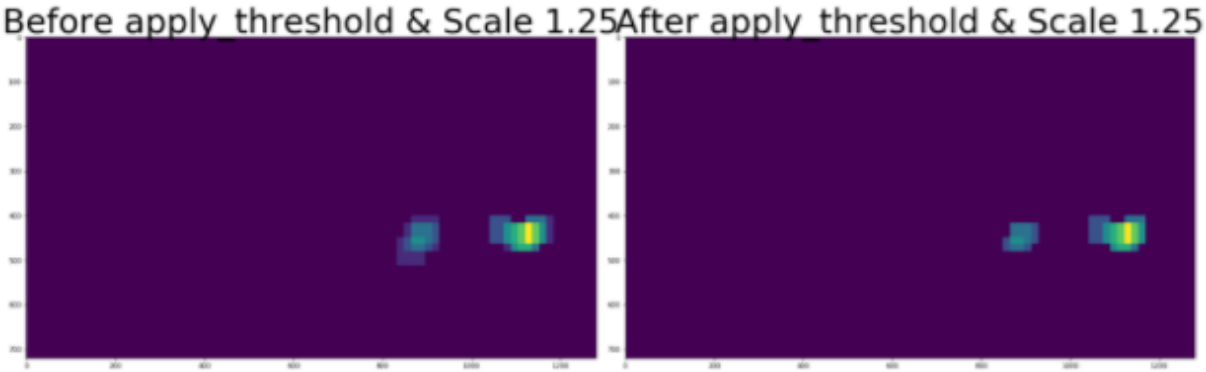
In brief and short description of the pipeline. At first, we don’t know the scale with which we should process the image, but we can guess min scale can be 1.0 and max scale can be 2.0.

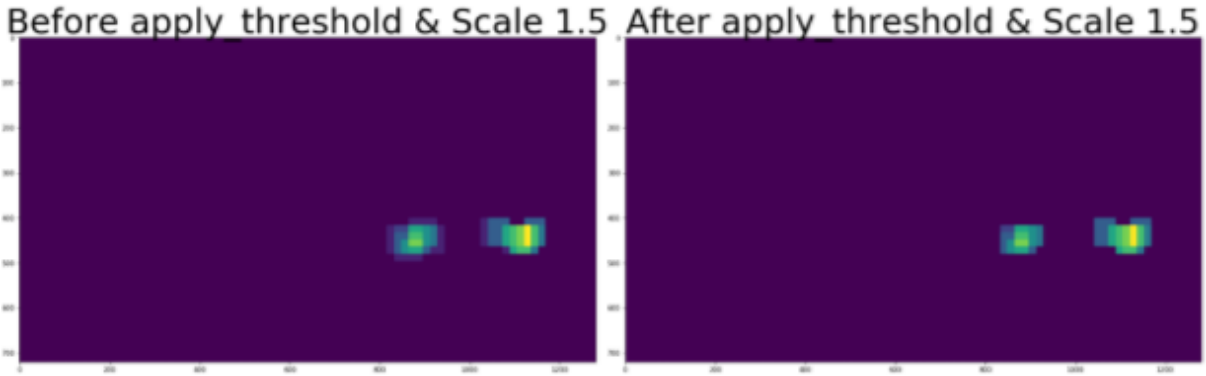


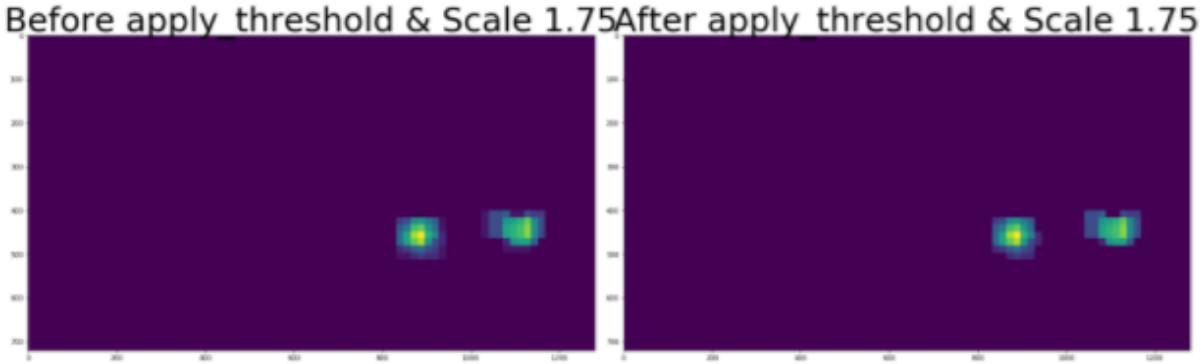
The next figure left is heat variable before applying threshold and right is after applying threshold. We can see the small fleck is removed after using threshold.

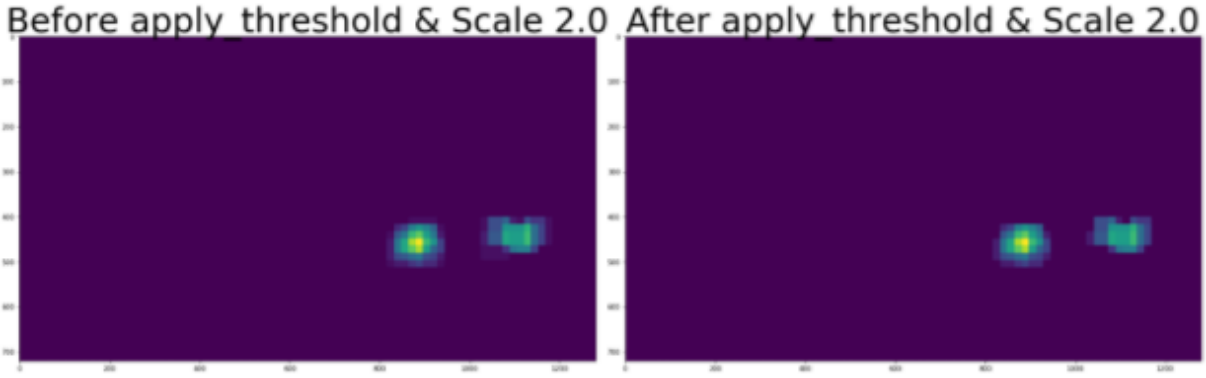


We see the removed fleck is appeared again in next iteration with scale 1.25 because it has been predicted again as car and it’s not removed from heat after applying threshold.

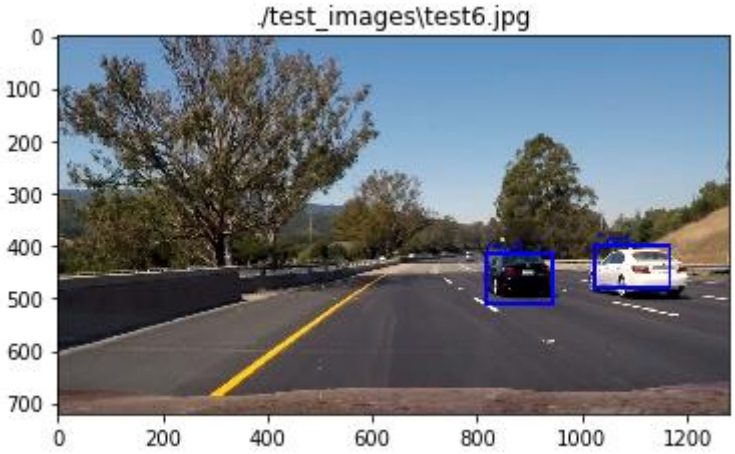








The two predicted objects in the above figures are detected correctly. Because I keep the first two predicted objects and each iteration I add the previously detected to newly detected and then apply threshold. Therefore the correct detected objects will be detected always and the false positive will not be considered as shown in the next figure.



Video Implementation

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.)

The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

The output video is available.

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.

A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

Resubmission 1

Another logic that I have changed is in this part. The reviewer suggested to use deque object.

```
from collections import deque
history = deque(maxlen = 8)
...
history.append(current_heat_map)
...
```

The length of deque is 10 in my code to collect the predicted and actual boxes from each frame.

In the previous review of my submission the reviewer has suggested to use the deque object to keep the correct or it's better to say correct predicted boxes.

The boxes, which are correct will be saved in a deque object with length of 10 and in the next iteration they will be added to the new predicted boxes.

In "Multiple Detections & False Positives" cell I have used the add_heat function from training material as show bellow.

```
def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

    # Return updated heatmap
    return heatmap# Iterate through list of bboxes
```

In this logic a heat_map will be created for all the pixels within window where a positive detection has been predicted by the classifier. Finally, by imposing a threshold we can reject the false positives.

```
def apply_threshold(heatmap, threshold): # Zero out pixels below the
threshold heatmap[heatmap <= threshold] = 0 # Return thresholded map
return heatmap
```

But after running the code I have noticed that the classifier has detected some slides as positive, but they were false positives and I needed a logic to compare the previously predicted position of the vehicle with new predicted positions. This logic has been developed in process_bboxes function.

To have a reliable result I have changes the heat-map technique a little via collecting the previous predicted boxes. In each iteration that the heat_map will be calculated, the previously detected or predicted boxes will be inserted to the currently predicated boxes and the threshold will be applied to this list. Therefore, the probability that the false positives are removed from the result is higher.

Discussion

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 had problem with distinguishing between cars and noncars; therefore, I had to use deque object to work out this issue.