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.

I have a function for extracting the Histogram of Oriented Gradient with following header which has been defined under “Extracting the Histogram of Oriented Gradient” cell.

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

Furthermore, I have developed a small code snippet under “Visualization of extracted Histogram of Oriented Gradient” cell to visualize the output of get_hog_features function. The following figure is the visualization of the output of HOG function.

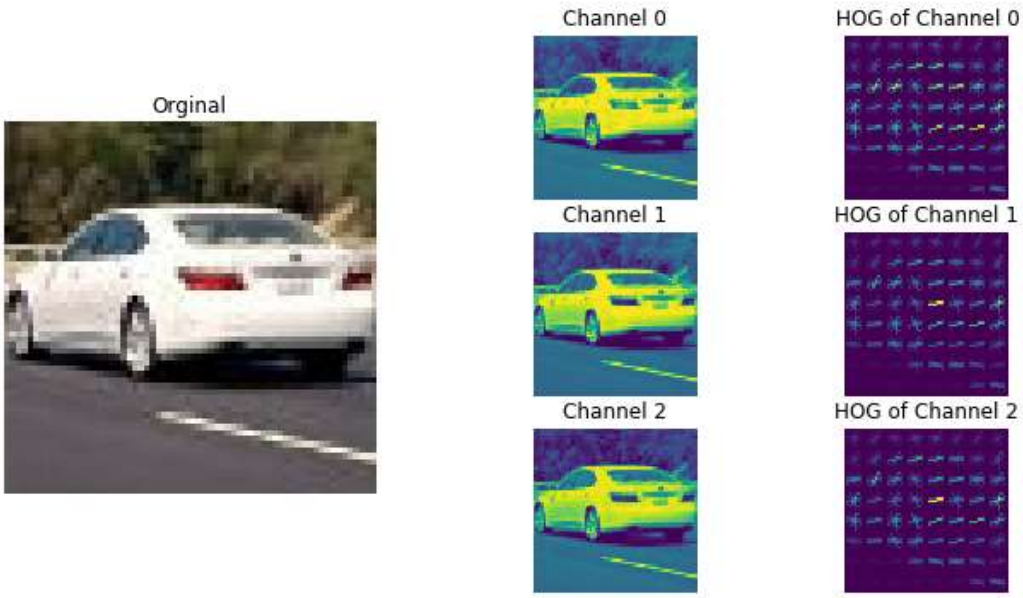


Figure 1: orient=9, pix_per_cell=8, cell_per_block=2

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 “Train LinearSVC Classifier” cell: via calling extract_feature function from training materials.

```
car_features = extract_features(cars, color_space=color_space, orient=orient,
                                pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                                hog_channel=hog_channel, spatial_size=spatial_size, hist_bins=hist_bins ,
                                spatial_feat = spatial_feat, hist_feat = hist_feat, hog_feat = hog_feat)
notcar_features = extract_features(notcars, color_space=color_space, orient=orient,
                                    pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                                    hog_channel=hog_channel, spatial_size=spatial_size, hist_bins=hist_bins ,
                                    spatial_feat = spatial_feat, hist_feat = hist_feat, hog_feat = hog_feat)
```

- Under “Using LinearSVC Classifier to detect vehicles in an Image via defined function” cell: via calling find_cars function.

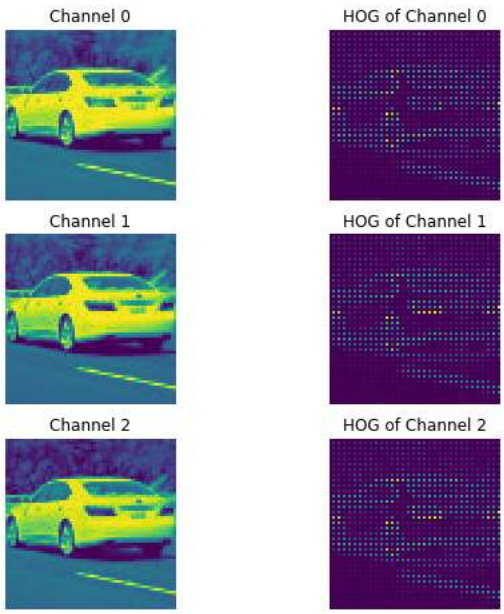
```
box_list = find_cars(np.copy(image), ystart, ystop, scale, svc, X_scaler,
                     orient, pix_per_cell, cell_per_block,
                     spatial_size, hist_bins, color_space,
                     spatial_feat = spatial_feat, hist_feat = hist_feat, hog_feat = hog_feat)
```

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



For example, the beside figure 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 “Train LinearSVC Classifier” section in Jupyter Notebook.

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

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

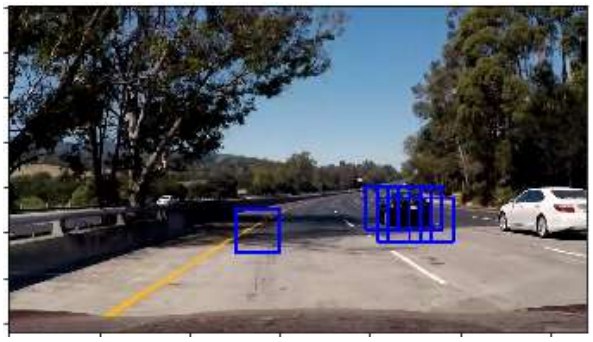
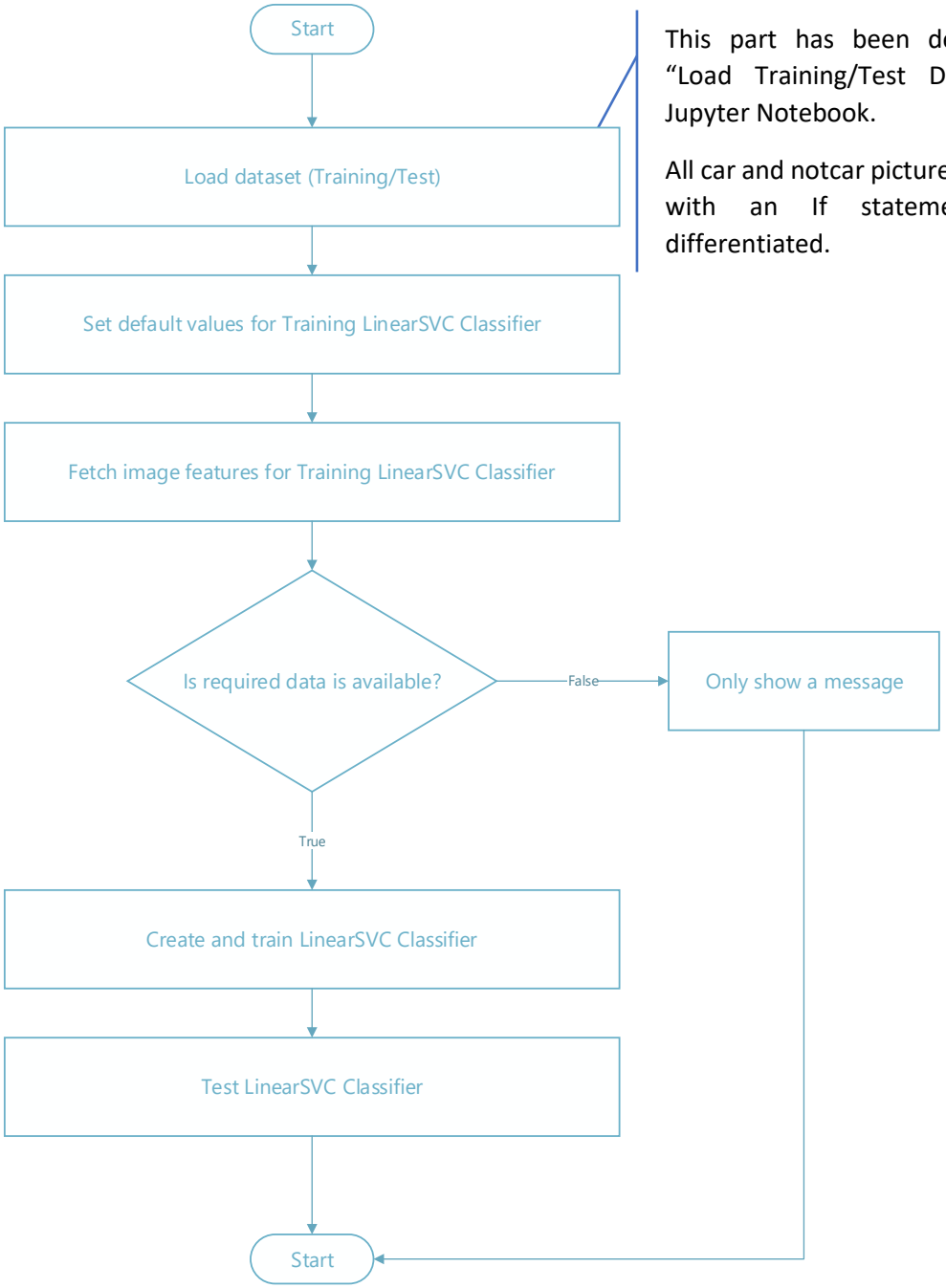


Figure 7: YUV

Figure 8: YCrCb

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

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.

Under "Train LinearSVC Classifier" cell I have defined the default values for training, these values are used for car detection as well. In the following code snippet, we can see the variables.

```
color_space = 'HSV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
#-----------------------------------
# Spatial Classify
#-----------------------------------
spatial_size = (16,16) # Spatial binning dimensions
spatial_feat = False # Spatial features on or off
#-----------------------------------
```

```
# Color Histogran Classify
#---------------------------------
hist_bins = 32    # Number of histogram bins
hist_feat = False # Histogram features on or off
#---------------------------------
# HOG Classify
#---------------------------------
orient = 11
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
hog_feat = True # HOG features on or off
```

After defining variables, I extract the feature/s of the car and notcar images with *extract_features* function and create an array stack of the extracted features to pass to *StandardScaler().fit()* function.

The features are extracted according to the *spatial_feat*, *hist_feat* and *hog_feat* variables, which are described in previous section.

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

```
# Create an array stack of feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)
```

After getting the scaled data, they must be splitted to two Train and Test datasets because the Train dataset should be passed to Model training as shown below.

```
svc.fit(X_train, y_train)
```

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?

sliding

The sliding window mechanism, which I have used is from training materials but with some changes. In this project I have combined “Find car” function and “Sliding” function and I consider scale in this function as well.

After I have combined these two with each other I have noticed that the trained Linear SVC Model cannot be used for all the scales because my first Model was trained with image size of (64x64).

The following example explains the reason, why it was not possible to use the Linear SVC Model for all the scales.

spatial_feat = False hist_feat = False hog_feat = True Image_size = 64x64 Scale = 1.0 color_space = 'RGB'	If I train a Model with these settings, only HOG feature is used. The original size of image is 64*64 and the color scape is RGB and if all extract the HOG for all layers, therefore has the output the size of 64*64*3. And the scaler which is created can normalize the objects with the same size.

spatial_feat = False hist_feat = False hog_feat = True Image_size = 64x64 Scale = 0.5 color_space = 'RGB'	But scaler which is calculated according to these settings can normalize the objects with size 32*32*3. We have used 32 because the original size was 64 and the scale was 0.5 therefore 32*32 is the size of the image.

Scale

Under “SVC Model for different scales” cell I have a code snippet, which contains a loop from scale=0.25 to scale=2.0. It calculates the scalers and models for each scale separately and assign them to different variable. These variables have been defined under “Different SVC models for different scale” cell. It means each scale has an individual model for itself.

The scales I have considered are 1.0 to 2.0 with counter step 0.25.

Scales = [1.00, 1.25, 1.50, 1.75, 2.00]

Overlapping

The over lapping that I considered for this project is 0.75 to have a better result. Higher overlapping causes to have the precise start edge of car (first x and y position, where the car appears in image) and precise end edge of car (the last x and y, where the car is not considered any more).

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

To have a reliable result I have changes the heat-map technique a little via summing all founded boxes form each scale iteration to each other and it's descripted briefly under the following section "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.".

The section technique that I have used is calculating a dynamic threshold for cooling the heat-map.

```
    percentage_fraction = 5
        ...
        # Apply threshold to help remove false positives
        heat_threshold = (max( heat.max(axis=0)) * percentage_fraction)/100
        if heat_threshold < 1 : heat_threshold = 1
        heat = apply_threshold(heat,heat_threshold)
```

I have seen a fix value for threshold doesn't remove the false positives and in my case because I accumulate all the boxes in one heat therefore the values which are small are irrelevant and they must be removed. The percentage_fraction says the values which are less than 5% of max_value must be removed.

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 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.

I have considered two extra features in my development for

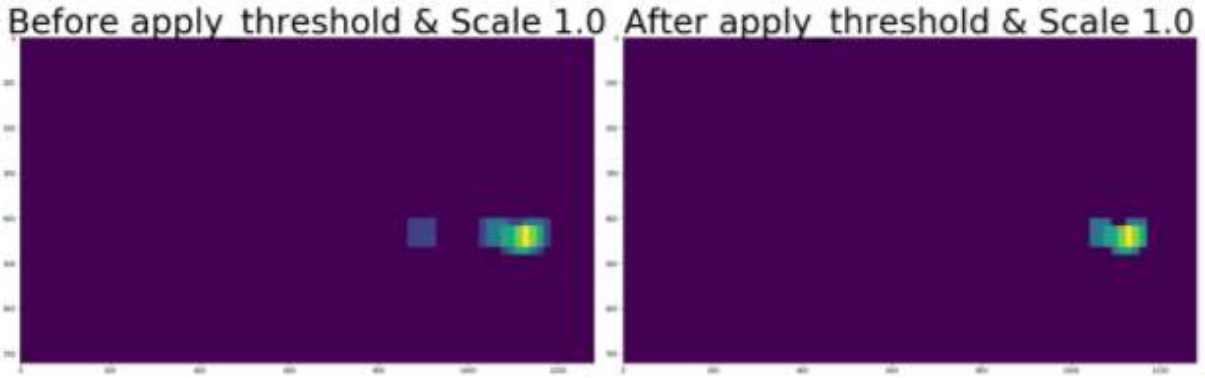
- Heat
- Threshold (was explained before under "Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining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

In sliding step each box, which was predicted as a car object, was added or appended to the box list and then we added heat to each box of list. Through the implementation I have noticed if I use threshold=1 or threshold<1 remain some flecks in heat after applying threshold which are considered as a box when I called label function.

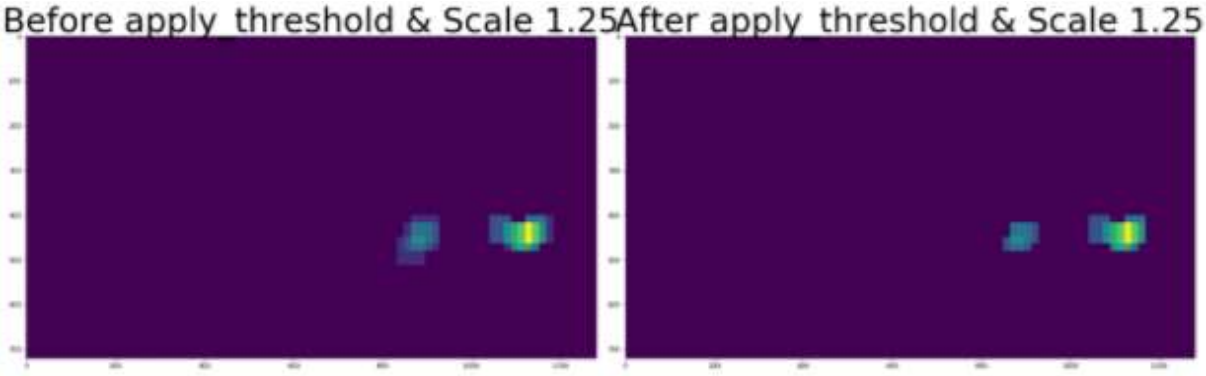
Therefore, I decided to make the boxes which are mostly predicted as car object more heater. This code snippet has been developed under "Using LinearSVC Classifier to detect vehicles in an Image via defined function" cell in "process_each_image" function.

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.

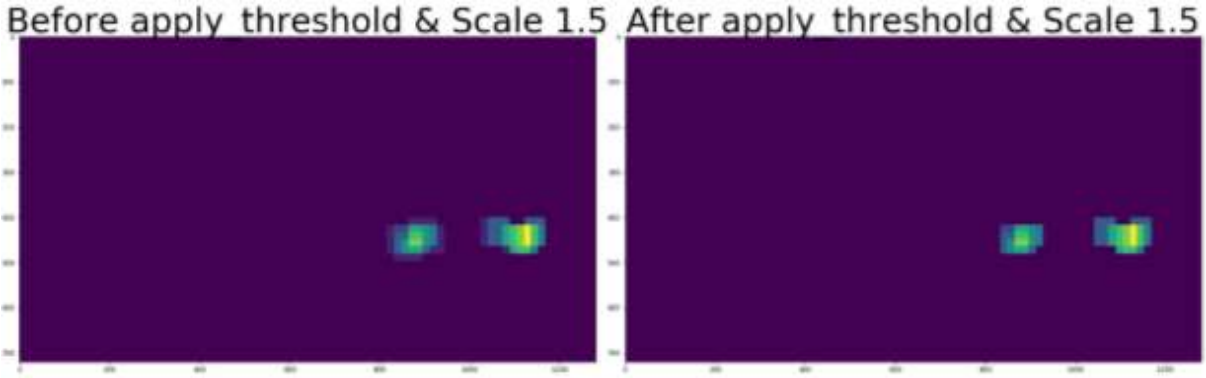
The next figure left is heat variable before applying threshold and left if after applying threshold. We a small fleck is removed.



We see the removed fleck is appeared again because it's predicted again as car and it's not removed from heat after applying threshold.

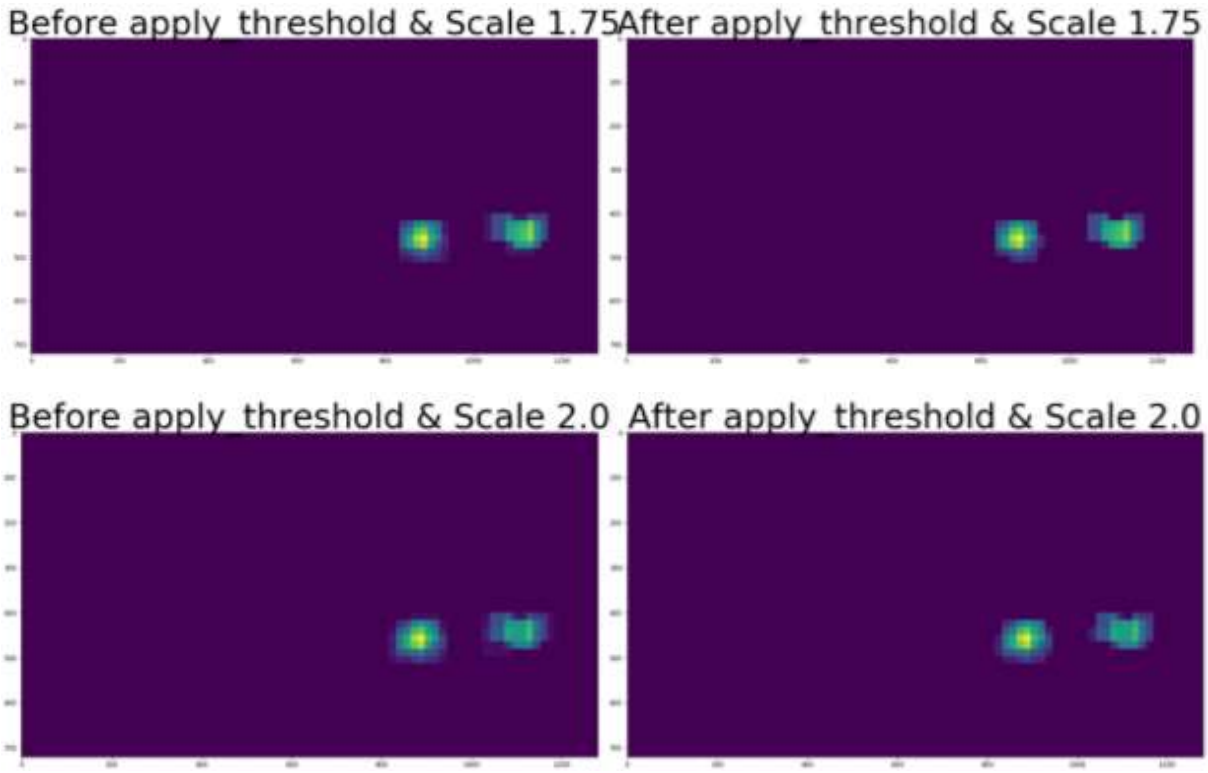


As we see the boxes which are really positive are detected again and the flecks are getting bigger it means the boxes which will be drawn around the objects will surround the whole object.



Furthermore, I sum the heat of current loop to the previous heat therefore each fleck which is a correct prediction get heater and bigger.

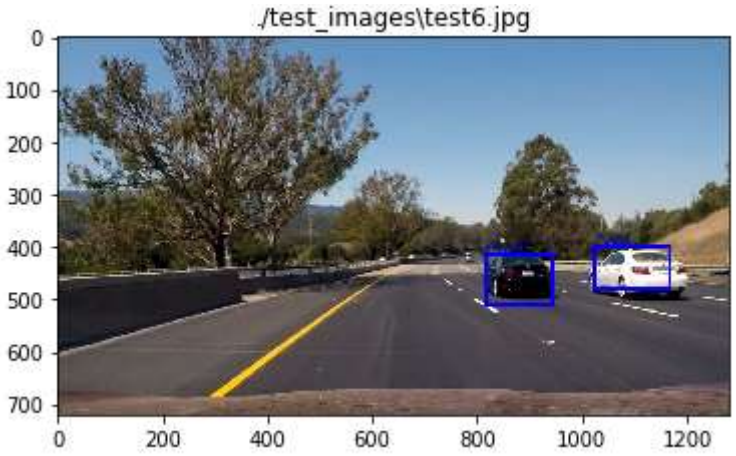
```
    #---------------------------------------------------
    # This line is for Multiple Detections & False Positives
    #---------------------------------------------------
    heat = np.zeros_like(image[:,:,0]).astype(np.float)

    scale = 1.0
    heat_threshold = 0
    percentage_fraction = 5
    while (scale <= 2):

        # Find boxes which are predicted as car object via find_carII
         # function and save in box_list

        # Add heat to each box in box list
        heat = add_heat(heat,box_list)

        # Apply threshold to help remove false positives
        heat_threshold = (max( heat.max(axis=0)) * percentage_fraction)/100
        if heat_threshold < 1 : heat_threshold = 1
        heat = apply_threshold(heat,heat_threshold)

        scale = scale + 0.25
        box_list.clear()
```



And finally, we can draw the boxes which are extracted via label function.

```
    # Visualize the heatmap when displaying
    heatmap = np.clip(heat, 0, 255)

    # Find final boxes from heatmap using label function
    labels = label(heatmap)
```

```
draw_img = draw_labeled_bboxes(np.copy(image), labels)
```



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 detecting the notcar objects. Maybe it's not a good idea to use only this method to distinguish car and notcar object from each other. Because for detecting all the notcar objects in the environment a collection of e.g. 1000 image is not enough and for car object as well. I think this method can be one of the mechanism with them we can classify car and noncar.
- I process a whole image for all the scales that I have but I should segment the whole image and assign a scale to each segment and then process the frame of image which I consider in sliding. It helps me to have better performance.