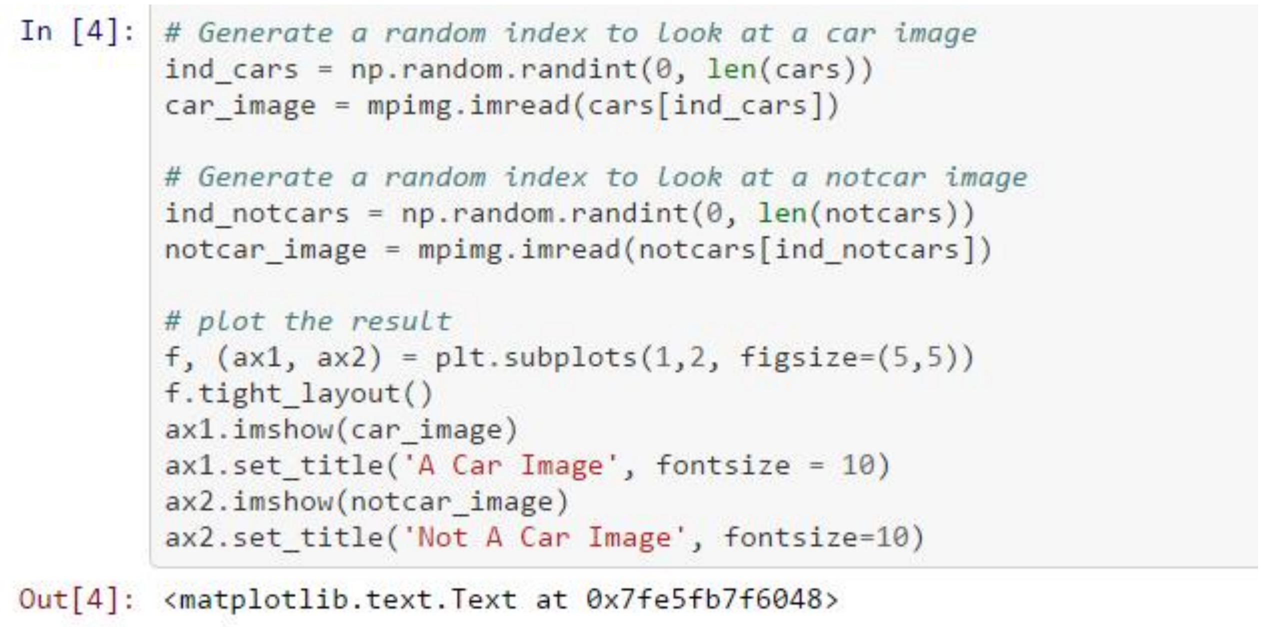
P5 Vehicle Detection Project   Denise James

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 in this section is found in the Searchclassifynotebook-20170216.ipynb file in cell 4. All files referred to in this readme write up document are on the github site provided at submission of this project.

I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes using the follow code:

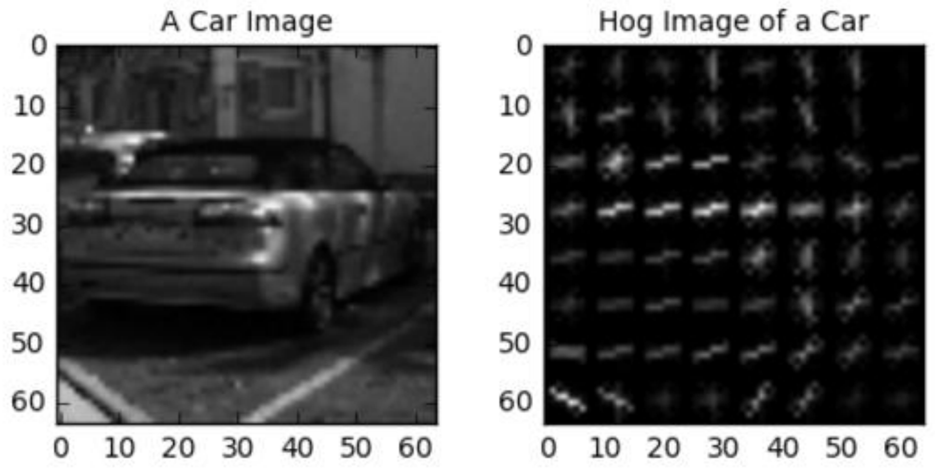


```
In [4]: # Generate a random index to look at a car image
        ind_cars = np.random.randint(0, len(cars))
        car_image = mpimg.imread(cars[ind_cars])

        # Generate a random index to look at a notcar image
        ind_notcars = np.random.randint(0, len(notcars))
        notcar_image = mpimg.imread(notcars[ind_notcars])

        # plot the result
        f, (ax1, ax2) = plt.subplots(1,2, figsize=(5,5))
        f.tight_layout()
        ax1.imshow(car_image)
        ax1.set_title('A Car Image', fontsize = 10)
        ax2.imshow(notcar_image)
        ax2.set_title('Not A Car Image', fontsize=10)

Out[4]: <matplotlib.text.Text at 0x7fe5fb7f6048>
```

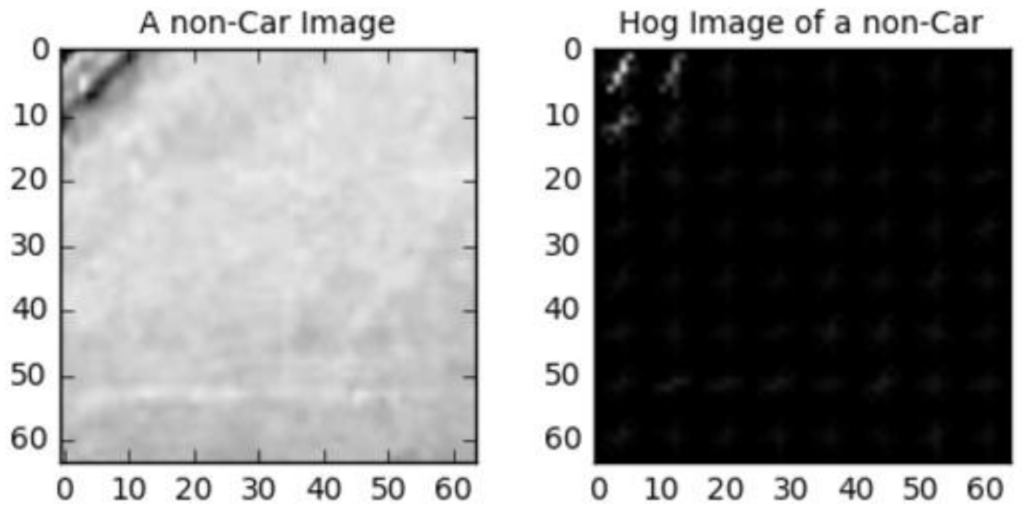
I next explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.

Here is an example using the YCrCb color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

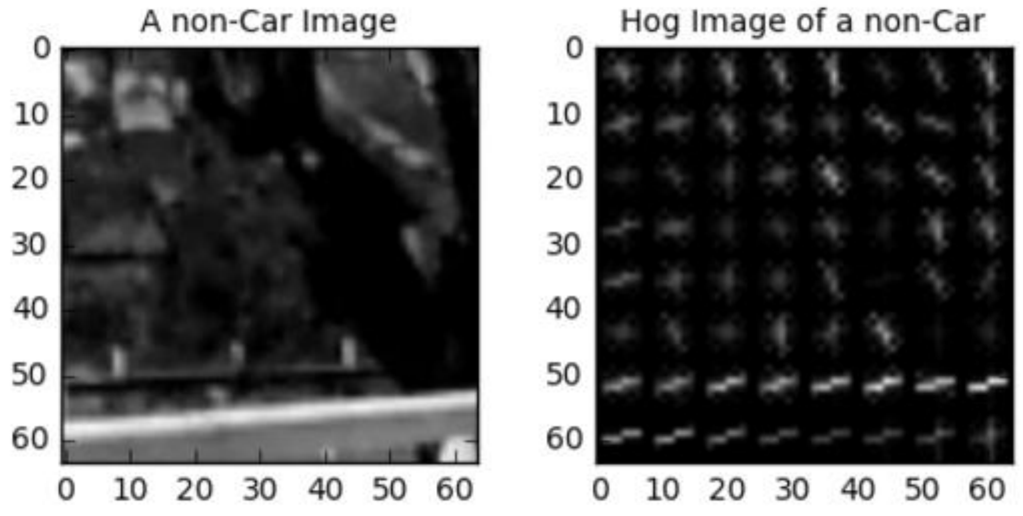
```
In [7]: gray_image = car_image[:,:,2]
        feature_vector, hog_image = hog(gray_image, orientations=orient,
                                        pixels_per_cell=(pix_per_cell, pix_per_cell),
                                        cells_per_block=(cell_per_block, cell_per_block),
                                        transform_sqrt=True,
                                        visualise=True, feature_vector=True)

        f, (ax1, ax2) = plt.subplots(1,2, figsize=(5,5))
        f.tight_layout()
        ax1.imshow(gray_image, cmap = 'gray')
        ax1.set_title('A Car Image', fontsize = 10)
        ax2.imshow(hog_image, cmap = 'gray')
        ax2.set_title('Hog Image of a Car', fontsize=10)

Out[7]: <matplotlib.text.Text at 0x7fe5f9e5e860>
```



```
Out[11]: <matplotlib.text.Text at 0x7fe5f9b0bac8>
```



```
Out[9]: <matplotlib.text.Text at 0x7fe5f9c6f0b8>
```



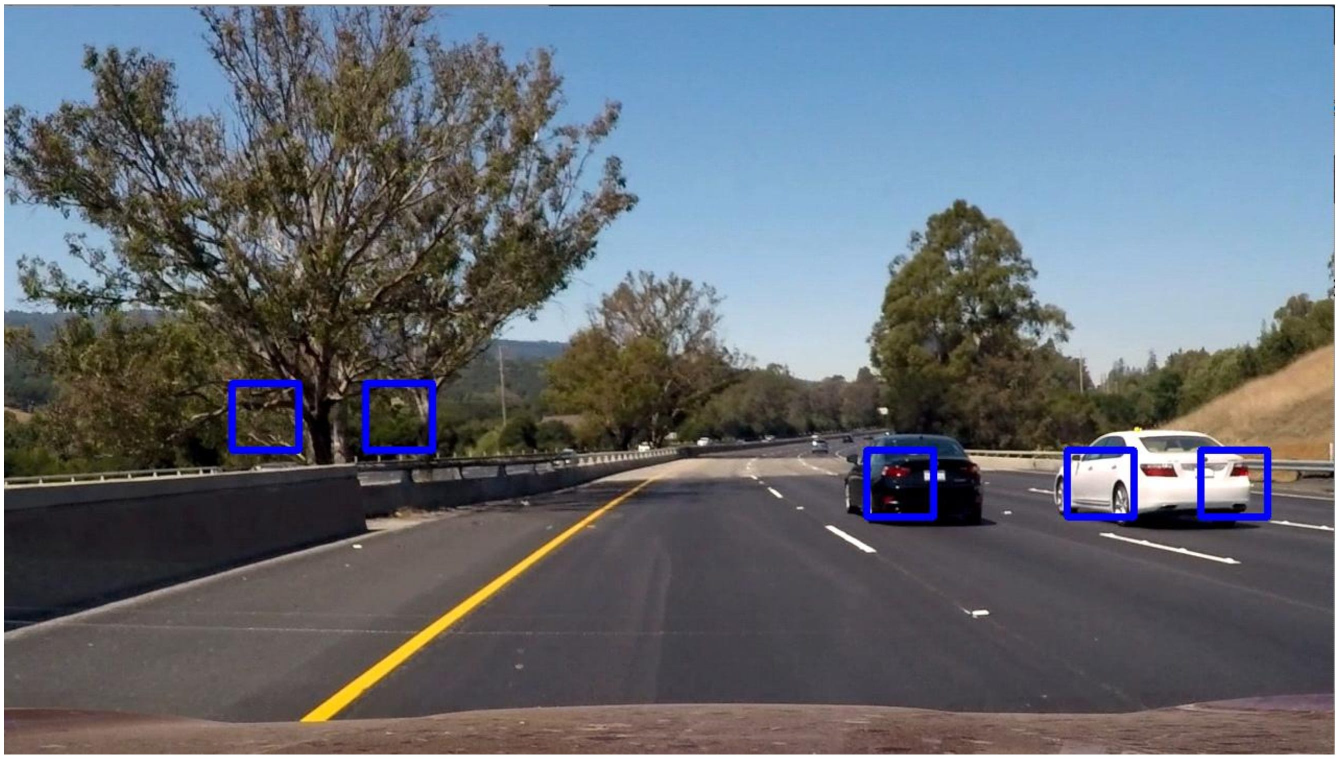
2. Explain how you settled on your final choice of HOG parameters.

I tried dozens of combinations of color space, bin spatial, and HOG parameters of orientations, pixels_per_cell, cells_per_block, xy_overlap and xy_window.

Here is an example using the YCrCb color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

```
y_start_stop = [330, 680] # Min and max in y to search in
slide_window()
x_start_stop = [0, 1260]
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV,
YCrCb
```

```
orient The code for the above is in= 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
```

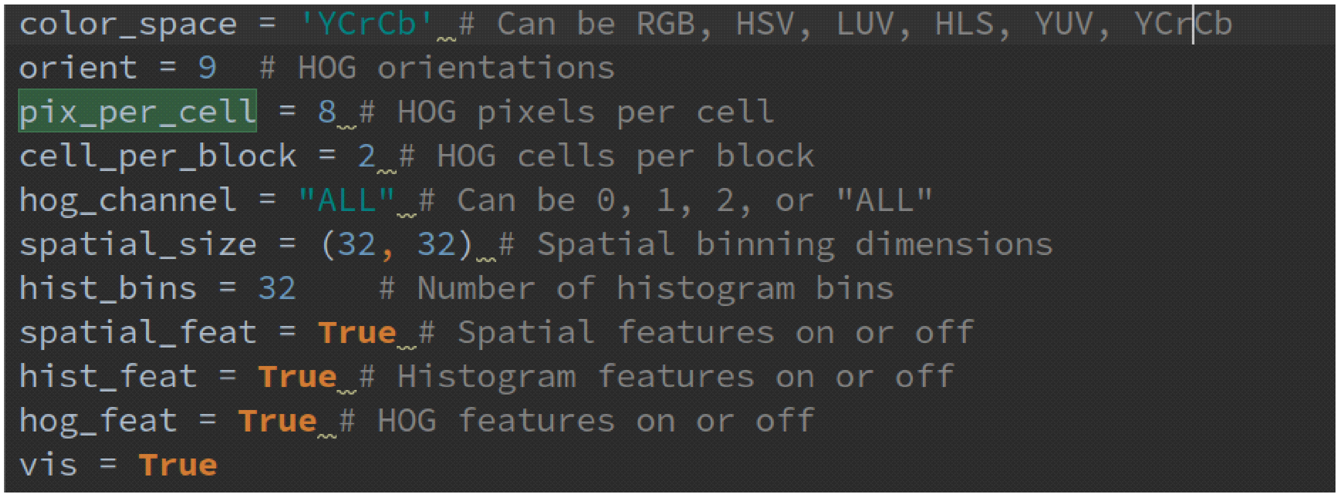


I decided to change the x range to eliminate the false positives on the left.

I tried training various combinations of parameters. Recently I became proficient in what needed to be trained. Training takes fifteen to thirty minutes on my computer to train. Variables that may be added later as xy_windows, xy_overlap, x_start_stop and y_start_stop may be modified after the classifier and stabilizers are loaded into the next part of the code.

HSV color gave the highest accuracy rating but YCrCb is better in detecting the cars. I choose YCrCb

The following parameters were used after experimenting and evaluating the accuracy rating and how the training results operated on new data. A 99.% accuracy may not perform as well as a training of 98% accuracy. These final training values were selected. This choose parameters are shown in the picture below.



```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimensions
hist_bins = 32    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
vis = True
```

With the following results.

```
/home/denise/anaconda3/envs/carnd-term1/bin/python
/home/denise/p5/Searchclassify_20170216.py
Number of vehicles images found 8792
Number of non-vehicles images found 8968
got past Scaled_X
Using: 9 orientations 8 pixels per cell and 2 cells per block
color_space YCrCb
Feature vector length: 8460
55.63 Seconds to train SVC...
Test Accuracy of SVC =  0.9916
```

The code for the above is in Searchclassify_20170216.py

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). The code for the above is in Searchclassify_20170216.py

I trained a linear SVM using.

```
X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler normalize
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)
print('got past Scaled_X')
# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
```

The most important function in extracting features list for bin spatial, HOG and color code is the extract_features function shown below.

```
# Have this function call bin_spatial() and color_hist()
```

```
def extract_features(imgs, color_space='RGB', spatial_size=spatial_size,
                        hist_bins=hist_bins, orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block, hog_channel=hog_channel,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        file_features = []
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        if color_space != 'RGB':
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: feature_image = np.copy(image)
        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        if hist_feat == True:
            # Apply color_hist()
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # Call get_hog_features() with vis=False, feature_vec=True
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):
                    hog_features.append(get_hog_features(feature_image[:,:,channel],
                                        orient, pix_per_cell, cell_per_block,
                                        vis=False, feature_vec=True))
                hog_features = np.ravel(hog_features)
            else:
                hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                            pix_per_cell, cell_per_block, vis=False, feature_vec=True)
            # Append the new feature vector to the features list
            file_features.append(hog_features)
        features.append(np.concatenate(file_features))
    # Return list of feature vectors
    return features
```

The code is found in the Searchclassify_20170215.py file on github.

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 experimenting with sliding windows of 64x64, 96x96 and 128x128 individually, I realized that each sliding window added to detecting they the vehicles. I decided to sum the windows of 64x64, 96x96 and 128x128. Happy with those results, I added even more more. I like big boxes.

Here is my sliding windows routine:

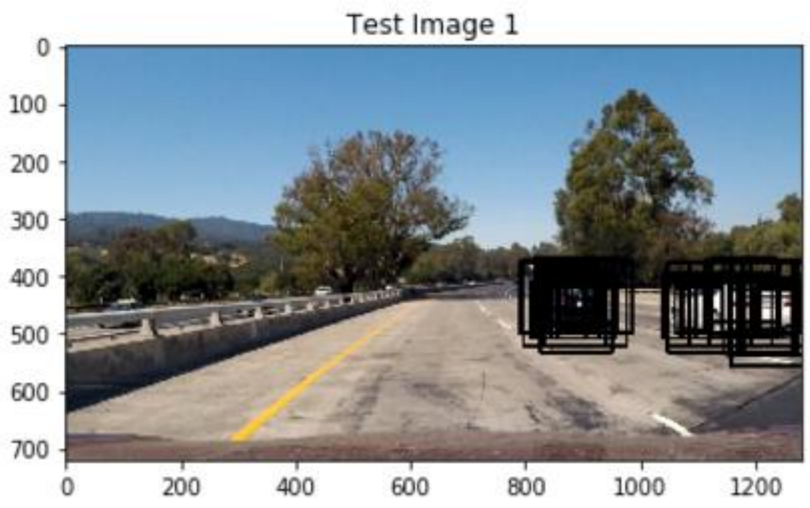
```
def slide_windows_cumulative(img):
    # create an empty list to append all windows
    all_windows = []
    for size in [64, 80, 96, 112, 128, 144]:
        windows = slide_window(image, x_start_stop=x_start_stop, y_start_stop=y_start_stop,
                  xy_window=(size, size), xy_overlap=xy_overlap)
        all_windows += windows
    return all_windows
```

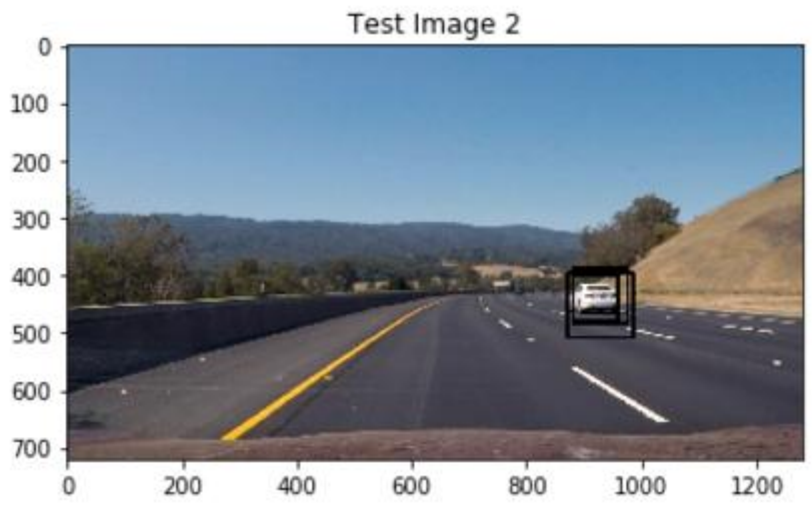
The above code is in plotslidingwindows.ipynb.

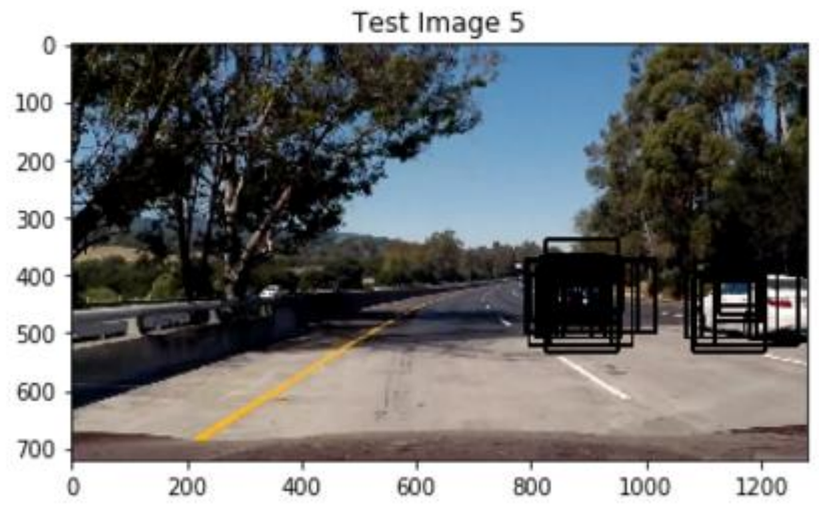
This is the code to selecting many detection windows:

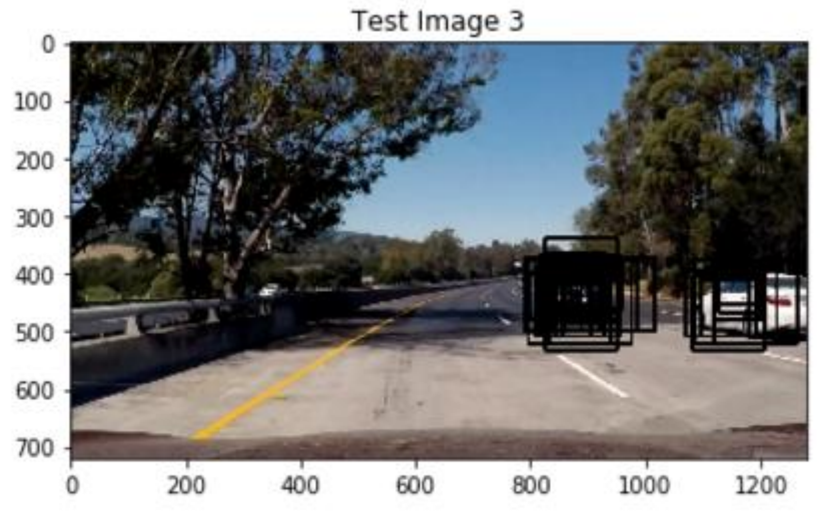
```
def slide_windows_cumulative(img):
    # create an empty list to append all windows
    all_windows = []
    for size in [64, 80, 96, 112, 128, 144]:
        windows = slide_window(image, x_start_stop=x_start_stop, y_start_stop=y_start_stop,
                        xy_window=(size, size), xy_overlap=xy_overlap)
        all_windows += windows
    return all_windows
```

Here are some test images processed with the above slide_windows_cumulative function.



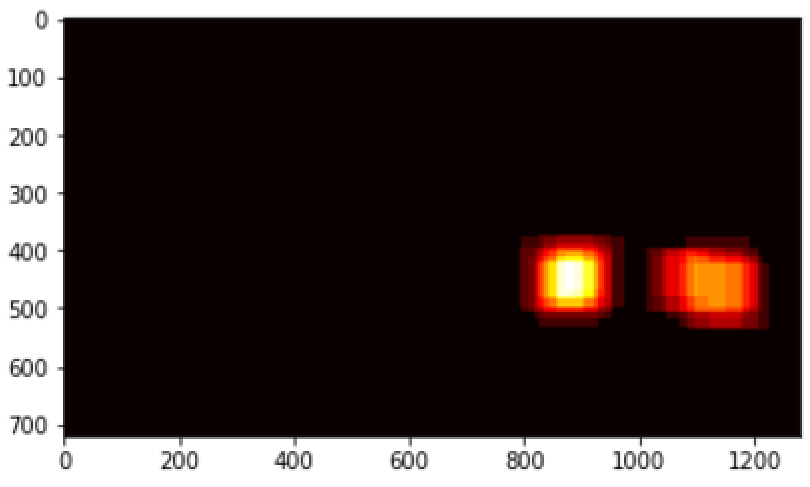






Here is a heat map of the vehicles. This heat map is used to help eliminate false detections. I choose a threshold of 2 as the lectures notes does.

```
Out[22]: <matplotlib.image.AxesImage at 0x7fd311f218d0>
```



Video Implementation

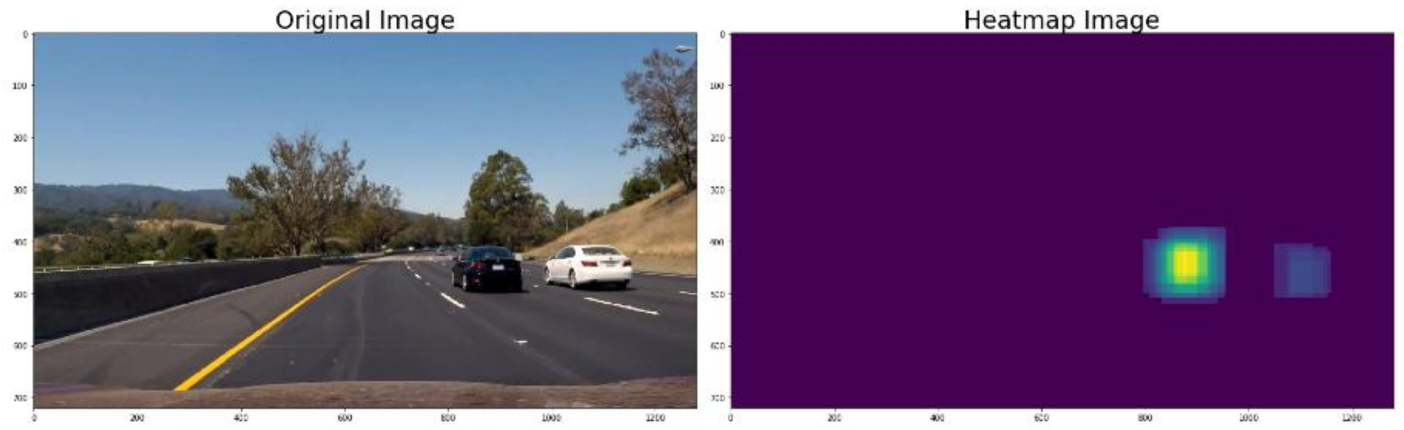
1. Video

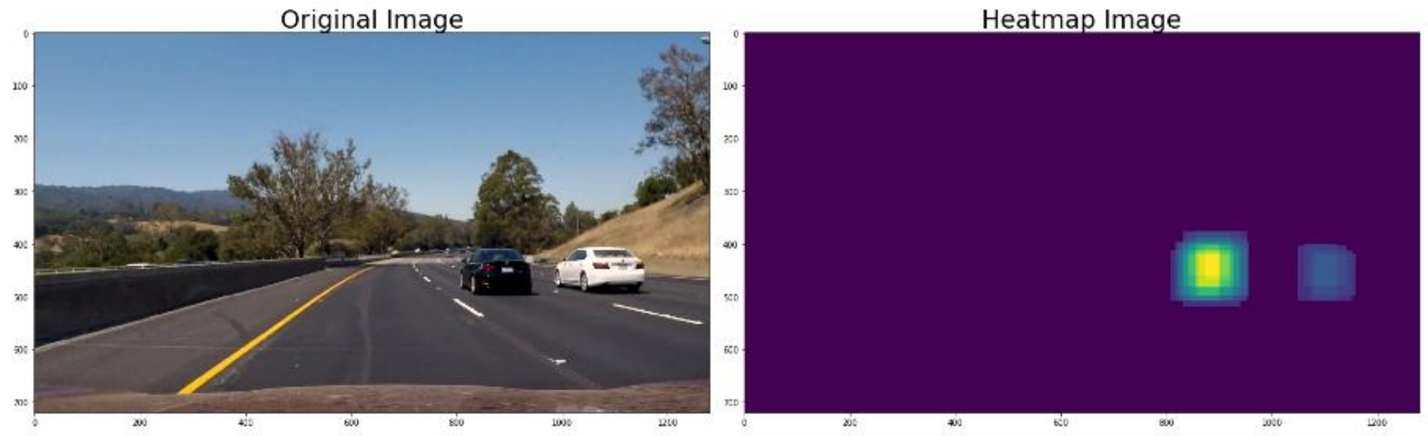
Here is a link to my video.

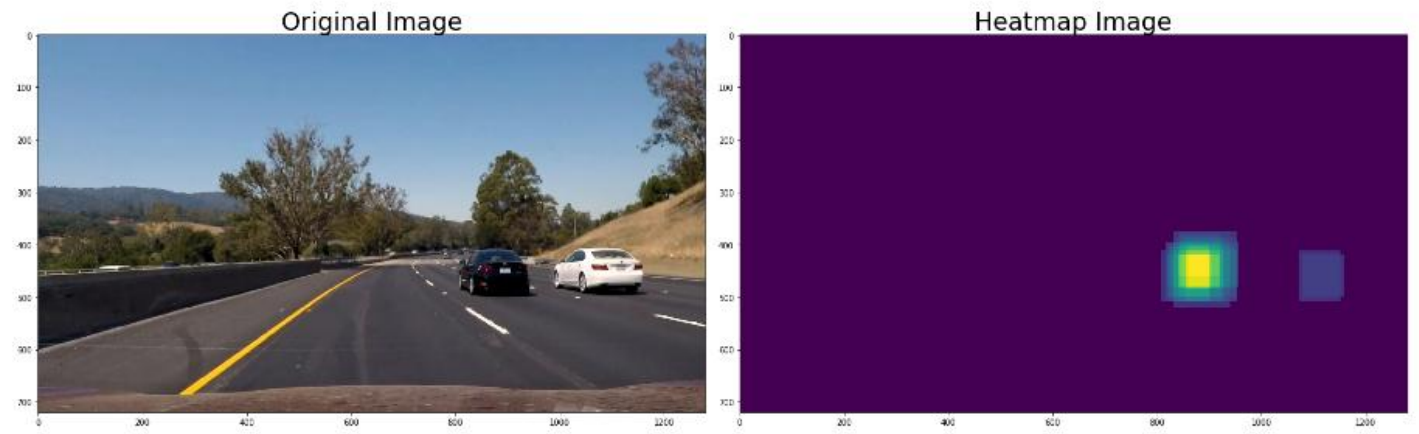
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

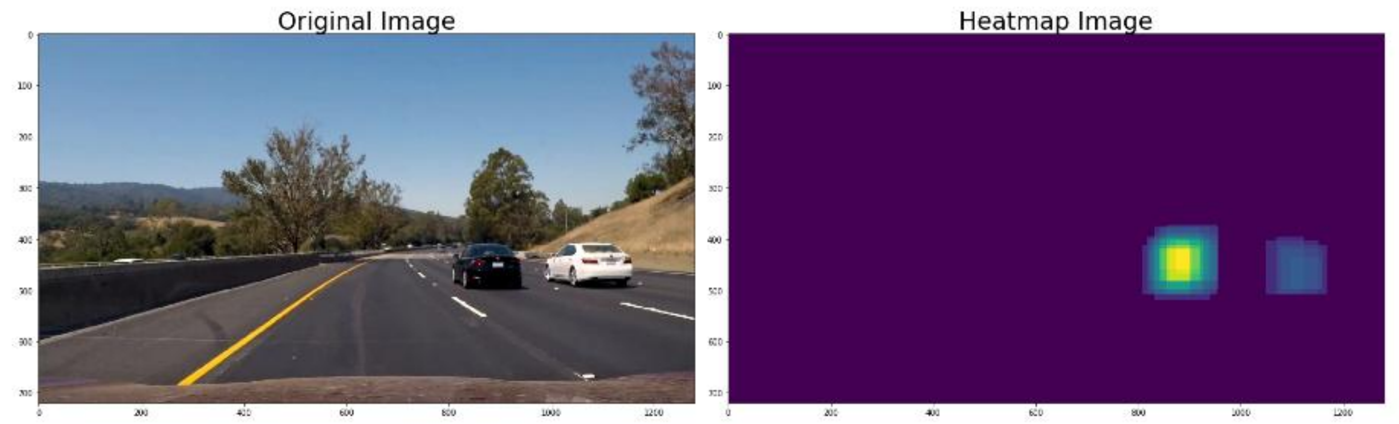
"I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each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

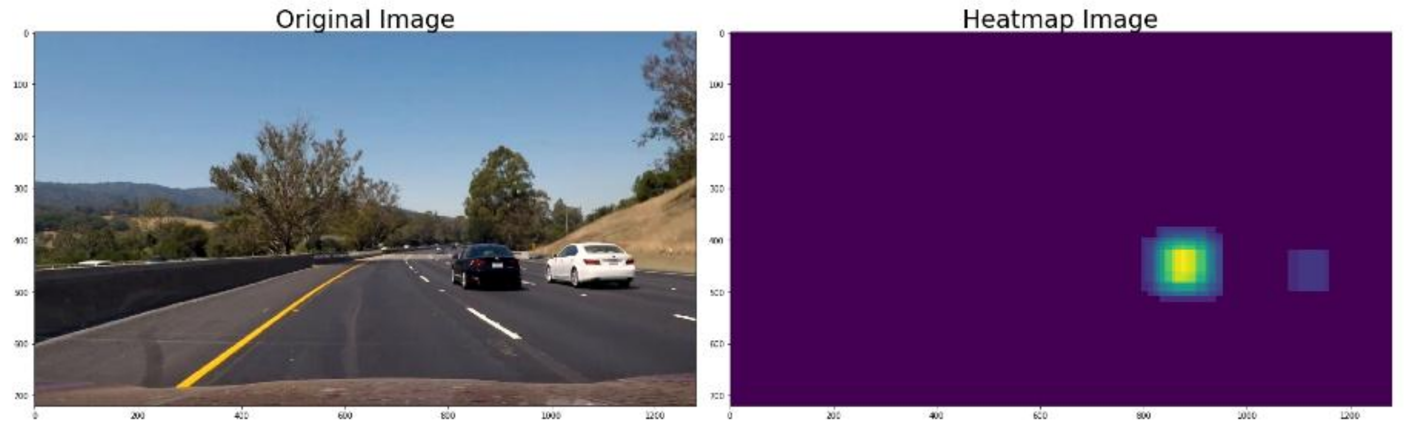
Here are six consecutive frames and their corresponding heatmaps:

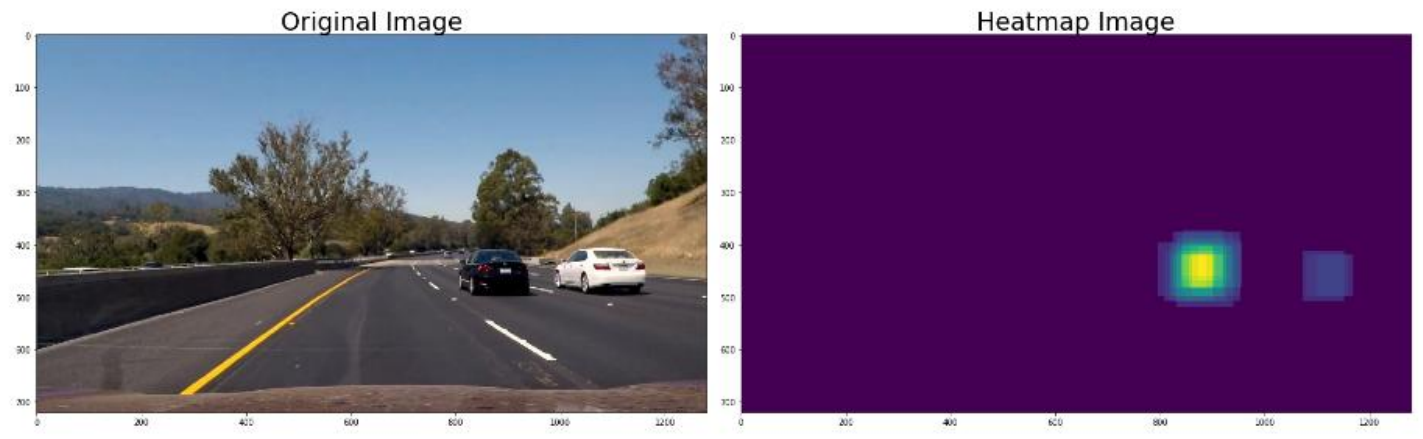





Original Image
Heatmap Image


Original Image
Heatmap Image


Original Image
Heatmap 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 had many IDE issues. I downloaded Pycharm for this project 5 and was later forced to download the latest Carnd-Term1 environment to use use Pycharm. Jupyter Notebook did not have enough resources to train the data as Pycharm is. Integrated Development Environments must be debugged such as the VideoClip did not work in Carnd-Term1 environment as it did in the environment I used for project 4.

I took a practical approach of not tracking cars across the barrier. I set x to eliminate false detections in that area.
My code would improve if I detected moving objects on the left shoulder.

I found out training the data in Pycharm and saving the classifier and normalizer to a pickle file, and uploading the classifier and normalize in Jupyter Notebook to test the other parameters saved a lot of time.

I