Vehicle Detection and Tracking

For this project, I used a Jupyter notebook to write and test the code. A significant portion of the basic image processing pipeline is borrowed from the lessons, and I have attributed accordingly in the comment in the notebook cells. I will focus on discussing my unique modifications and improvements.

Feature Extraction

For my implementation, I used the following three features from the training images.

1. HOG features
    a. colorspace = 'YCrCb'
    b. orient = 9
    c. pix_per_cell = 8
    d. cell_per_block = 2
    e. hog_channel = 'ALL'
2. Spatial features
    a. spatial_size = (16, 16)
3. Color histogram features
    a. hist_bins = 16

I used all 3 channels of the YCrCb color space and tuned the parameters based on experimentation

After extraction, the feature vector for each image is normalized with StandardScaler.

```
X_scaler = StandardScaler().fit(X)
scaled_X = X_scaler.transform(X)
```

I then saved the scaler so that it can be used in the future

```
scaler_filename = "scaler.save"
joblib.dump(X_scaler, scaler_filename)
```

Train Classifier

I chose SVM as the classifier to use for this project.

First I preprocessed the training data to sample only every 5th image. This is done to account for the time-series data in the “car” category, to guarantee that that randomized training and test data will be sufficiently different as to prevent overfitting.

I then used GridSearch to find the optimal hyperparameters for the classifier using the entire dataset as training data

```
parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 1, 10], 'gamma':[0.1,
1, 10]}
svr = SVC()
CLASSIFIER = GridSearchCV(svr, parameters)
CLASSIFIER.fit(scaled_X, y)
```

After the best hyperparameters were found, I manually configured the classifier and split the test and train data. I then retrained the new SVM classifier. The classifier is saved for reuse later.

```
CLF = SVC(kernel="linear", C=0.1, gamma=0.1)
CLF.fit(X_train, y_train)

classifier_filename = "classifier.save"
joblib.dump(CLF, classifier_filename)
```

The hyperparameter settings are as follows

- kernel = linear
- C = 0.1
- gamma = 0.1

The classifier CLF produced an accuracy of 0.9845 on the test set.

Sliding Window Search

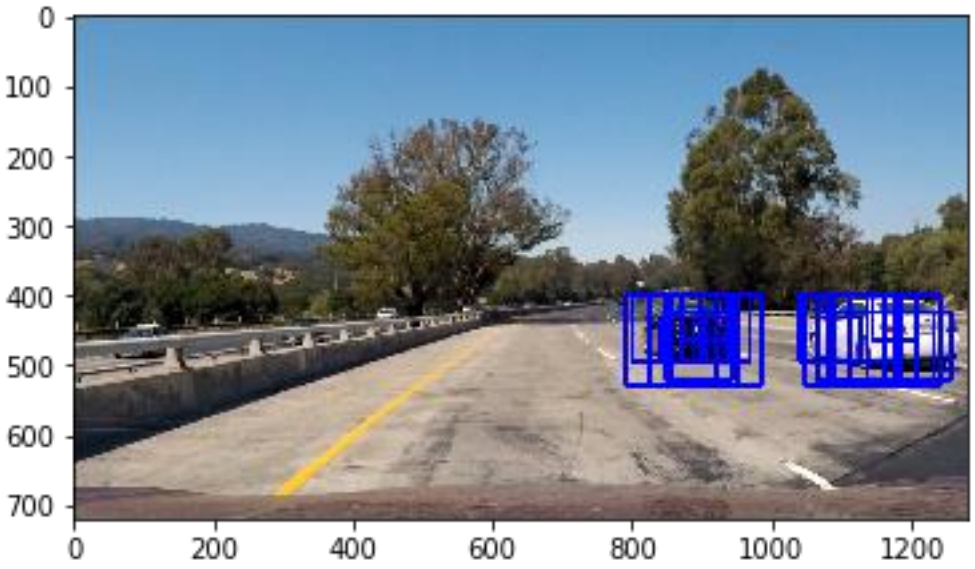
I used the approach presented in Lesson 35 – Hog Sub-sampling Window Search.

My implementation is unique in that it searches the image with a tuned cropped window that eliminates the sky as well as the left side of the image where there are no cars. I also iterate over multiple scales to find cars of various sizes at different distances from the camera.

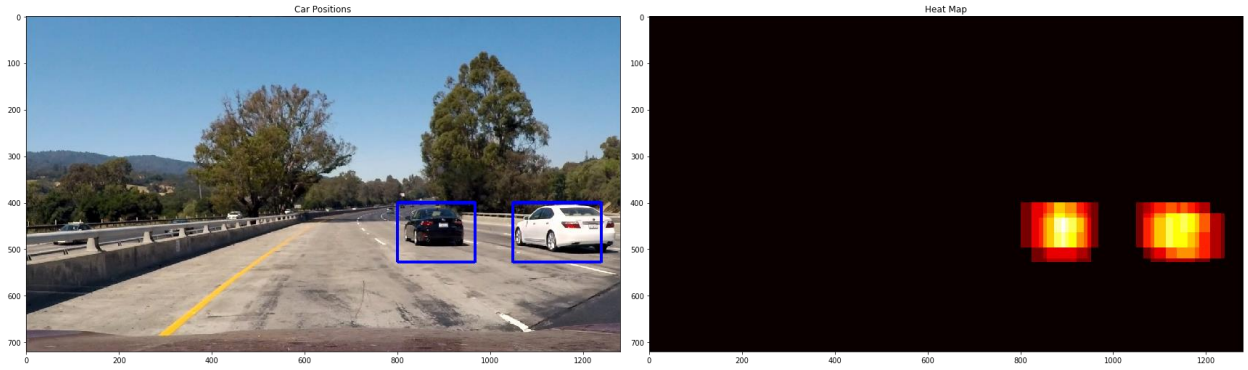
Here are the cropping and scale parameters I used.

```
XSTART = 440
YSTART = 400
YSTOP = 656
SCALES_LIST = [1, 1.5, 2]
```

I tested out the sliding window search on "test1.jpg".



The heatmap and final bounding box was then generated based on the sliding window output.



The images are saved in the output_images folder with the following file names.

- test1_boundbox.jpg
- test1_heatmap.jpg
- test1_window.jpg

Video Pipeline

output.mp4

The video pipeline is defined as the function process_image(frame).

```
def process_image(frame):

    global AVG_HEATMAPS
    global colorspace
    global orient
    global pix_per_cell
    global cell_per_block
    global hog_channel
    global spatial_size
    global hist_bins
    global XSTART
    global YSTART
    global YSTOP
    global SCALES_LIST

    # Find all matching window of different frame sizes
    box_list = find_boxes(frame, XSTART, YSTART, YSTOP, SCALES_LIST, orien
t, pix_per_cell, cell_per_block, spatial_size, hist_bins)

    # Read in image similar to one shown above
    heat = np.zeros_like(frame[:,:,0]).astype(np.float)

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

    # Append heat map to running average of 10 previous frames
    AVG_HEATMAPS.append(heat)
```

```
    # Sum heatmaps from previous 10 frames
    heat_sum = sum(AVG_HEATMAPS)

    # Apply threshold to help remove false positives
    # Set threshold level to 5
    thresh_heat = apply_threshold(heat_sum, 5)

    # Find final boxes from heatmap using label function
    labels = label(thresh_heat)
    output_img = draw_labeled_bboxes(np.copy(frame), labels)

    return output_img
```

To remove false positives, I implemented a moving average heatmaps of the previous 10 frames and only apply labels to data that has been thresholded to ensure high confidence detection.

The threshold limit I set is 5.

Discussions

1. Determining the size of the training dataset was tricky because it requires removing redundant data while still maintaining an acceptable level accuracy. Through experimentation, I've determined that it is possible to achieve good predication performance with only 1/5th of the original training dataset.
2. The hyperparameters used for feature extractions are also tricky to tune. I started out with something like the Udacity code examples and made minor tweaks.
3. GridSearch was very useful in finding the optimal SVM settings. It did take over an hour to converge on the best settings due to the large number of possible combinations.
4. The algorithm can be further improved by tuning the number of frames in the rolling average, as well as the threshold value for heatmap feature determination. I noticed that in the final output video, there seems to be a little bit of lag in the bounding box tracking. This can be remedied by reducing the number of frames to average. The heatmap threshold can also be reduced or increased to bound car more tightly. The total processing time for the video is 1 hr 39 min.
5. The pipeline will likely fail at smaller objects or objects near the edges of the frames. If the car is going down a steep hill, the horizon will be higher in the frame, so the cropped search area will need to be enlarged so as not to miss cars.