Writeup Template

Histogram of Oriented Gradients (HOG)

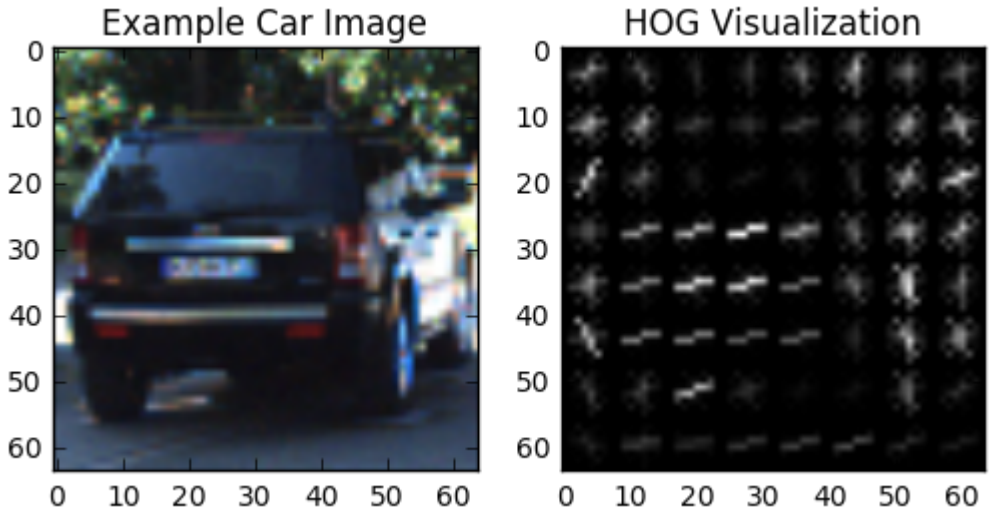
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first part of my IPython notebook: CarND_Vehicle Detection_and_Tracking_V1. ipynb

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

I then explored different color spaces and different parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`) to find the better features.

Here is an example using the `RGB` color space and HOG parameters of `orientations=8`, `pix_per_cell = 8 , cell_per_block = 2 :`



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

Before training the classifier, features were scaled to zero mean and unit variance. Then I combined spatial features, Histogram features and hog features to train my SVM classifier model. All the parameters listed as follows: hog_channel = 0, spatial_size = (16, 16) ,hist_bins = 32 , spatial_feat = True, hist_feat = True , hog_feat = True

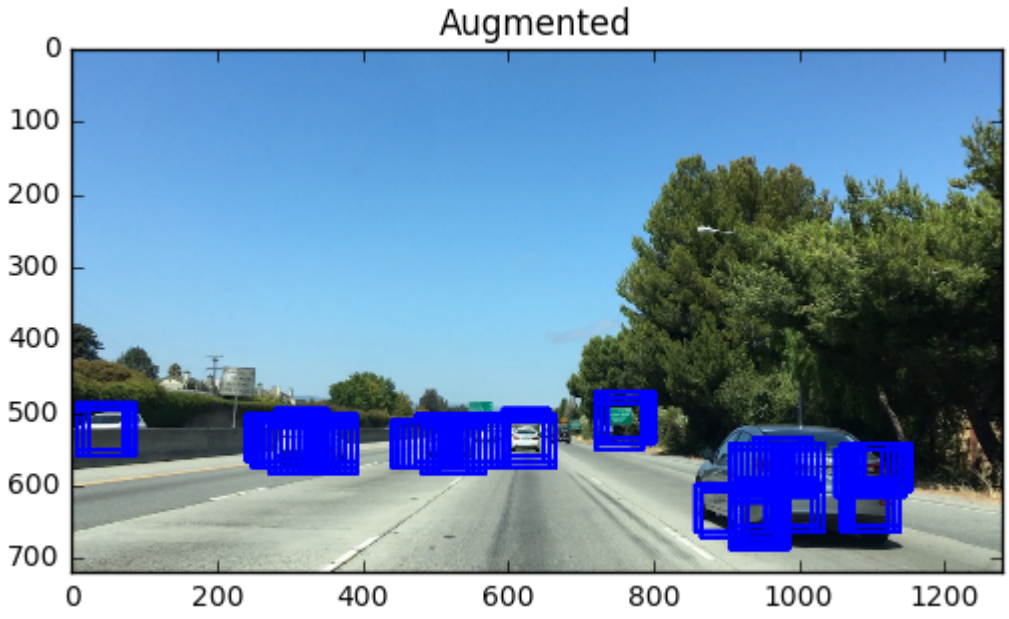
Sliding Window Search

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?

A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. . I tried different parameter, finally used y_start_stop = [390, 500] , x_start_stop=[700, 1300], then the results were better .

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

Ultimately I searched on two scales using RGB 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:



Video Implementation

1. 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 name of video is project_video_SVM_2.mp4

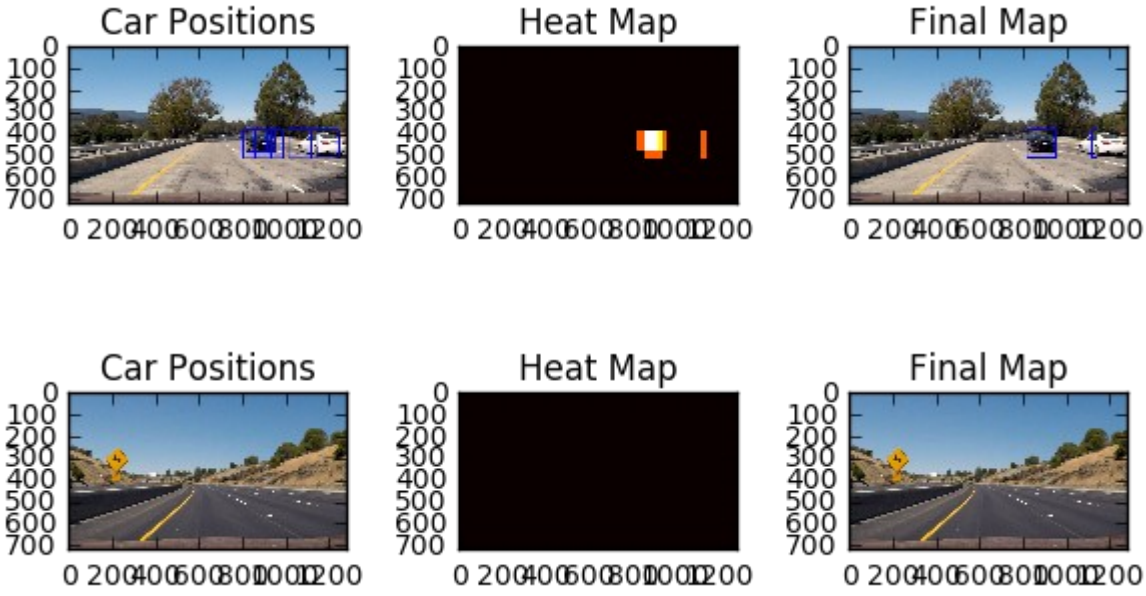
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

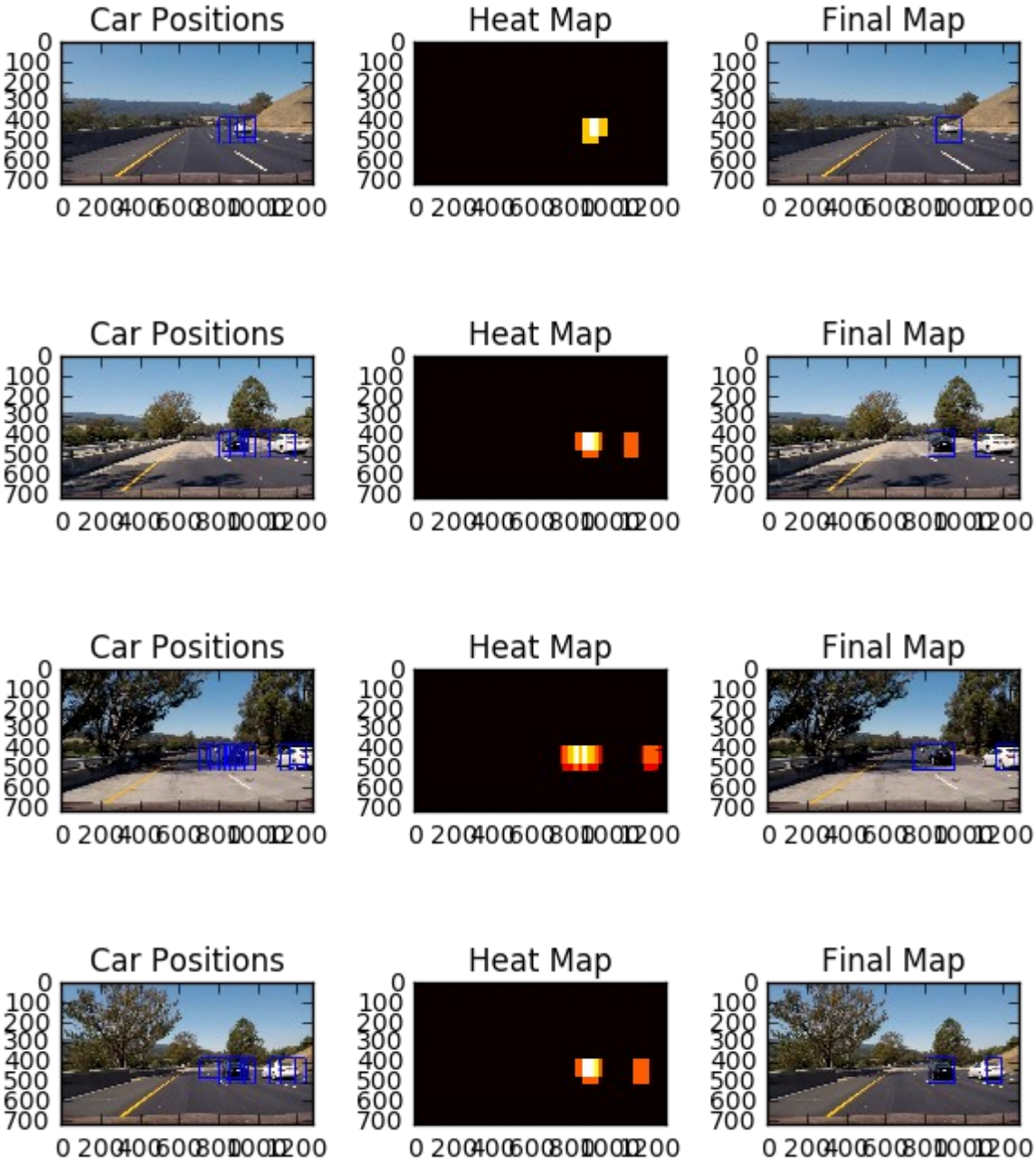
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `draw_hot_final()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to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

In order to reduce false positives, I tied to devise different blob in the same picture.

```
windows = slide_window(img, x_start_stop=x_start_stop, y_start_stop=y_start_stop,
                    xy_window=(96,96), xy_overlap=(0.6, 0.6))
windows += slide_window(img, x_start_stop=x_start_stop, y_start_stop=y_start_stop,
                    xy_window=(128, 128), xy_overlap=(0.6, 0.6))
windows += slide_window(img, x_start_stop=x_start_stop, y_start_stop=y_start_stop,
                    xy_window=(192, 192), xy_overlap=(0.6, 0.6))
```

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





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 think it's more tricky to decide which features we should use to train classified model. If we can use deep learning to train model, the result will improve a lot.