Vehicle Detection Writeup

Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Histogram of Oriented Gradients (HOG)

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

The code to convert to various colorspaces, extract the HOG features, as well as the color and spatial binning features from images is located in src/preprocess.py. The classifier class in src/classifier.py loads all the vehicle and non-vehicle images and calls the

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

When starting the coding I captured the various hyper parameters in src/config.py.

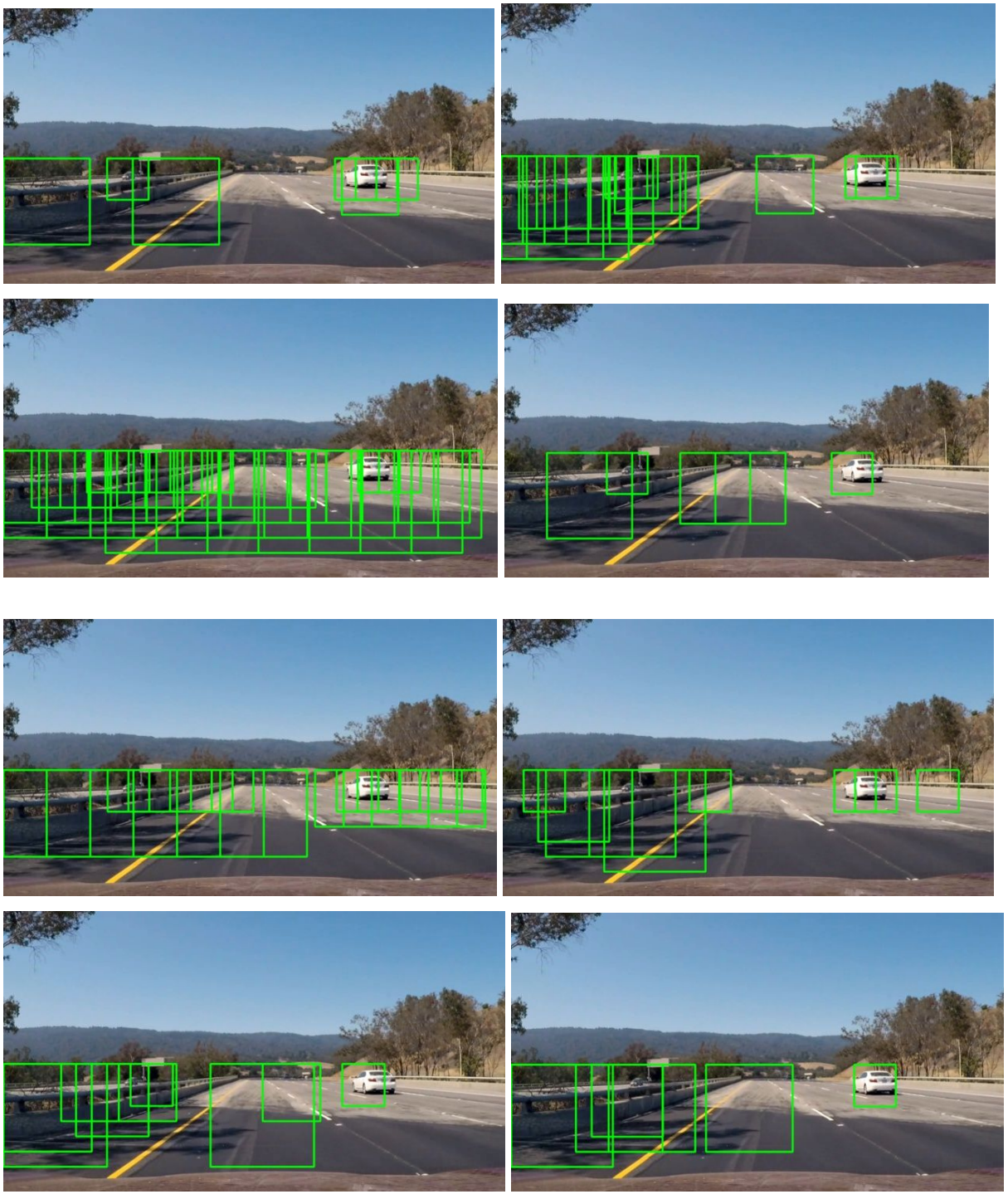
I decided to try different HOG and other features in a systematic way. src/test_image.py lists out a set of overrides

Example of configuration settings.

```
configs.append({
   "INPUT_CHANNELS": ['hls_s'],
   "HOG_CELLS_PER_BLOCK": (3, 3),
   "HOG_BLOCK_STEPS": 4,
   "USE_SPATIAL": False,
   "USE_COLOR_HIST": False,
   "RESULTS_FOLDER": "results/hls_s_no_extra_3x3_steps_4"
})
```

I then ran the test script for various config combinations, modifying such settings as color space, cells per block, spatial/color binning features. It trained each config on the full training set and then ran the pipeline on a selection of images from the test_images folder plus some consecutive images extracted from the video - in particular ones where I detected many false positives. Based on manual inspection of the results I chose the settings with the least amount of false positives and the most positive classifications on the vehicles in the images.

For example here is the same image tested with different color spaces and settings.



As you can see from the images the various different settings can lead to very different results.

Initially I focused on configurations that minimized the number of false positives and true positives for the vehicles. I.e. higher precision. However, I was unable to make the classifier that accurate. In the end I focused on the recall of the classifier, and used the fact that the vehicles are in nearly all the images , where as the various false positives don't last for that long.

Specifically for the HOG parameters, I tried various combinations of block size and orientation bins and overlapping steps within the region of interest.

In the end I choose to convert the images to the HLS colorspace and combine the HOG features with spatial and color binning. I used a block size of 4 cells with 8x8 cells. 9 hog orientation buckets.

The parameters are listed below.

```
INPUT_CHANNELS = ['hls_all']
HOG_CELLS_PER_BLOCK = (4, 4)
HOG_PIXELS_PER_CELL = (8, 8)
HOG_ORIENTATIONS = 9
USE_SPATIAL = True
USE_COLOR_HIST = True
```

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 that encapsulates the classifier is in src/classifier.py

This code will either load the cached model from a previous training session, but in the train method loads the various images splits into training and test set. Then trains a Linear SVM model using GridSearchCV in order to find the best hyperparameters for the model. Once the model has trained the model is pickled to a local file so it can be quickly loaded and used without having to be trained again.

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?

My sliding window search is implemented first by slicing and scaling the area of the image where vehicles are most likely to appear. This is first implemented in preprocess.py in the extract_regions_of_interest. Then

For each one of these slices, a SlidingWindowHog instance is created (src//sliding_window_hog.py). This class calculates the hog feature blocks for the entire slice and then extracts a set of feature windows that represents a 64x64 window. The VehicleDetector class then uses this info and adds the color and spatial histogram features before passing to the classifier for prediction.

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?

This is good example of sliding window in action.

I was able to improve the performance of the search

1. Precomputing the hog features for the entire scaled subregion.

2. by reducing the number of predictions the classifier had to make by adjusting
    a. how many blocks to move the window when searching from left to right.
    b. How many scaled slices to break the roadway image into.

i. I was able to adjust the number of regions also through a configuration flag. In the image above it's set to 5. This could possibly be reduced further, or even better might be to run each slice in parallel using multiple cores to reduce the image processing latency.

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.)

Here's a link to my video result

Here's a link to my video result with extra diagnostics in frame.

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.

For each video frame I built a binary result per pixel for each box the classifier returned a positive result for. Then I built an integration heat map from multiple frames, calculating the fraction of positive results per pixel over the number of frames. This heat map was then filtered with a relatively high fraction (0.9). This allows a pretty effective filtering as long as the classifier has recall higher than .9 this should keep a pretty stable box on the detected vehicles and filter out any false positives - as long as there aren't too many.

Once a minimum number of frames have been processed I used scipy.ndimage.measurements.label to extract the bounding boxes for the detections.

This is an example taken from the diagnostics images produced when processing the video. It shows the raw heatmap at the top right of the image, and the filtered heatmap below. Note also in this image that there are significant number of false positives but our vehicle is also detected.

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 ran into a number of issues with this project.

The most difficult and time consuming issue, was dealing with false positives. The first challenge was to find a set of features that reduced the initial false positives on a per frame basis. One thing I learned was that it's actually ok to have quite a few false positives as long as the recall of the classifier is good enough to classify the vehicles reliably. If I were to try to make this even more robust, I would try to measure the precision and recall of each configuration more empirically and focus on high recall and reasonable precision and let the filtering take care of the rest.

Next I found that while the heatmap helped reduce the false positives, it would still break down in certain parts of the video when the color of the road changed, or the yellow line to the left would cause a false positive over a few frames. I found that simply integrating over frames wasn't enough.
I tried some other ideas, for example I used a maximum filter to locate the local maxima in the heatmap and assumed that these were the center of the vehicles.
I then experimented with various other techniques to extract the vehicle boundaries.
I tried using OpenCV findContours on a thresholded image, and used the bounding box of the returned contour, this definitely showed some promise, but it picked up other contours from the background.

I then captured a portion of the image around the center and ran the classifier on just that part of the image - but again this wasn't particularly robust. It seems to me that this is a case of noisy data and techniques might be required to help filter out false detections.
One idea I did have would be to use the lane lines detected from previous project to further reduce the image search space, both for performance and to reduce the false positives.

Some of the other issues I encountered.

I think there is room for improvement in the accuracy of the classifier. Either by training with more data, or using a neural network instead of a support vector machine. It's possible that even more combinations of features could be tested in order to find the best features for this problem.

The processing of each scaled portion of the video could be handled in parallel, utilizing multiple cores in order to improve the real time performance of the pipeline. I believe it might also be possible to use fewer features to improve the performance of the classifier. Lastly, I know that my heatmap implementation could also be way more

optimized. The classifier is run on 64x64 pixel blocks so there is no need to build a heat map of the original size of the image. Shrinking this down should help performance quite a bit.