Vatche Donikian

Project 5: Vehicle Detection and Tracking

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.

Histogram of Oriented Gradients (HOG)

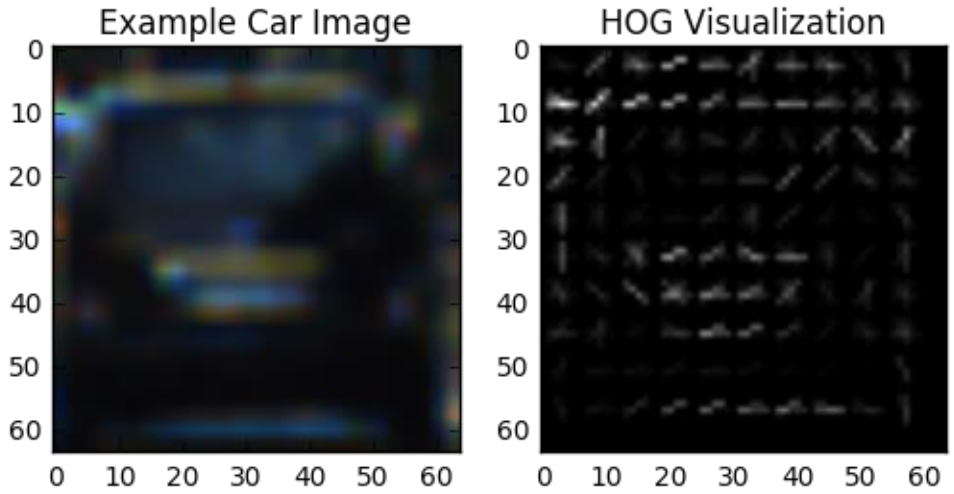
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the second code cell of the Jupyter Python notebook. The first cell is used to import all libraries and functions needed for the project. I read in images from the vehicle data provided, and labeled the vector ‘cars’. I also read in images from the non-vehicle data provided and labeled the vector ‘notcars’. The data I used for cars was from the ‘KITTI’ dataset, while the data I used for not cars was from the ‘Extras’ dataset. I also attempted to use the GTI data but I did not see an improvement so I left it out in the final run. I used the ‘glob’ function to import in all .png files in those folders. Below is an example of one of each of the car and non-car images:

Next, I performed the Histogram of Oriented Gradient (HOG) analysis as we did in the lesson. This consisted of defining a function to get the HOG features, and then defining a function to extract those features. Obtaining the HOG features was done as so:

features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell), cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=False, visualise=vis, feature_vector=feature_vec)

The different parameters that I adjusted were the number of orientations, the pixels per cell, and the cells per block. I also investigated the use of performing the HOG transform on different color spaces. Once this was set up, I ran this for the set of cars and not cars, extracted the features, and augmented the features with each other. I have provided an image below and its corresponding HOG feature extraction:



I also worked with adding a combination of features from spatial binning and histogram of colors. This feature extraction was completed in the 3rd jupyter notebook cell. I separated this from the HOG feature extraction so that I could attempt to pull features from a different color space.

Once I had the features extracted from the different methods, I combined them all, and then scaled and normalized them, in order to not have the different features interfere with each other in classification and training.

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

After lots of trial and error, I settled on the following parameters:

colorspace for HOG = 'YCrCb'
orient = 10
pix_per_cell = 6
cell_per_block = 2
hog_channel = "ALL"

I then chose the following parameters for the spatial binning and histogram of color feature extraction:

Colorspace for Spatial Bins and histogram of color: ‘HLS’
spatial_size=(64, 64)
hist_bins=32

I ended up at these values via trial and error. I found that the larger number of orientations, the more features I was extracting. However, using 20 orientations made my model overfit. Similarly, the smaller number of pixels per cell, the more features I obtained as well. If this number was too small, it would make the model take very long to train. Cells per block had a similar relationship. As for the color space, I found that YCrCb was best for feature extraction from HOG, and HLS was best for feature extraction from color histograms. The parameters I chose gave my classifier the most consistent set of vehicle detections throughout the test images.

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).

I trained a linear SVM using the sklearn commands provided in the lessons. This consisted of defining the classifier, fitting the classifier to the training data, and predicting labels then testing accuracy. This is all done in the 4th jupyter notebook cell. The classifier was trained using the following simple list of commands:

```
svc = LinearSVC()

svc.fit(X_train, y_train)
```

where X_train and y_train were training data obtained from a train_test_split of the original car i mages and labels. I set aside 20% of the data for testing the trained model, which seemed to be a n adequate amount. After fitting the data, I received a 99.86% test accuracy from my test set. Thi s was sufficient for me to move on to my next section.

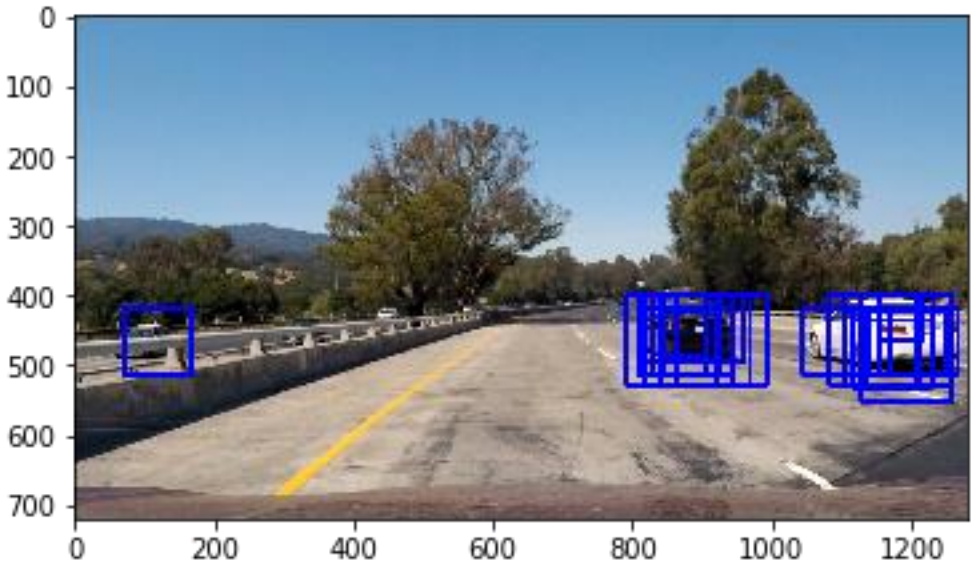
Sliding Window Search

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?

To detect the vehicles, I had to use a sliding window search. It does not make sense to search the whole image, as there should be no cars in the sky. Thus, I used starting and ending points in the y-direction for where a car could be found. I found that having a more overlap in the windows helped in creating many boxes wherever a vehicle is, distinguishing it from a false positive; however, too much overlap would slow the video production down. I chose to shift over two cells when sliding; this slowed down the program but helped make the model more robust. I also used three different scaling factors, in order to detect vehicles at all different perspective ranges. The scales I used were 1,1.5, and 2. This helped my detections be more accurate throughout the video.

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?

After much trial and error, I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the HLS color space, which provided a nice result. Here is an example of a raw sliding window detection from my pipeline:



As you can see, the black and white cars are detected many times, in different sizes as well. I have one detection on the left side, which is a false positive but I believe it may be from the car on the other side of the road. These false positives were filtered out with the heat map described below.

Video Implementation

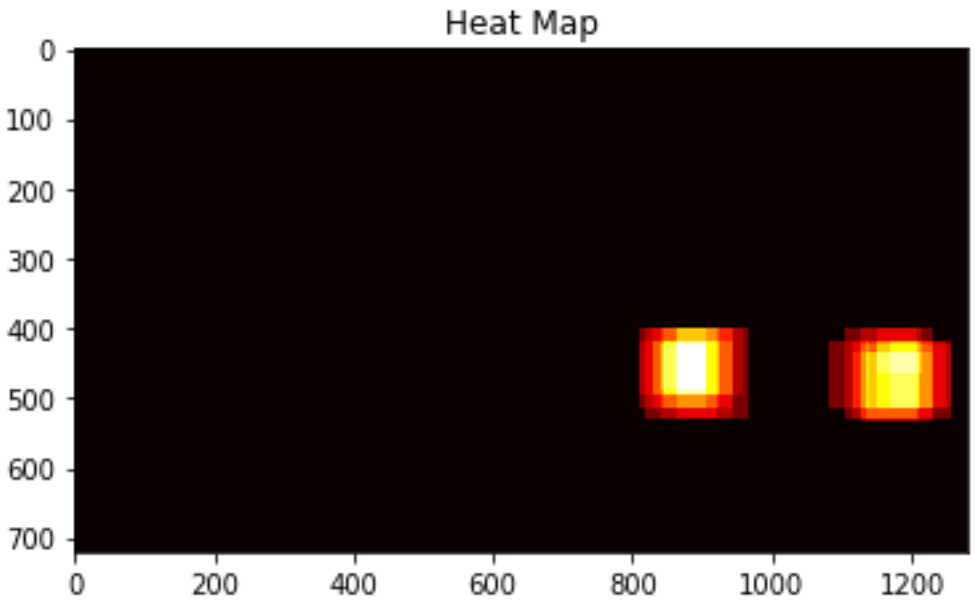
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

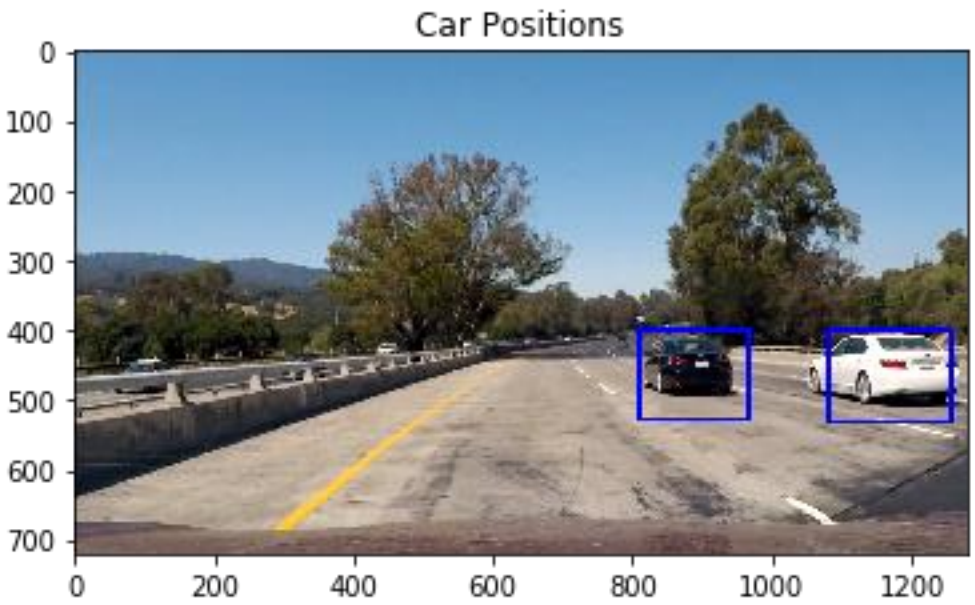
https://www.youtube.com/watch?v=6ndk9brZvx4&feature=youtu.be

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 vehicle detections in each frame of the video, using a bounding box array. From the positive detections I created a heatmap to identify vehicle positions. I applied a threshold so that only objects detected multiple times were chosen as vehicles, while the rest was brushed off as false positives. Then, I labeled the 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. These bounding boxes are where my estimate of each car is in each frame of the video. Then, when running the video, I saved the heat maps from the previous 12 frames and summed them up. Thus, my video was less jumpy (the vehicle detection was smoother).

Below is an example of the heat map corresponding to a test image, and the bounding box it created:





As you can see, the heat map helps filter out false positives, and create a nice general bounding box for the vehicles 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?

At first I had some problems with feature extraction. I was having difficulty detecting way too many false positives with just HOG extraction. So I decided to implement bins and histogram of color. It also took some time to implement this, as I wanted to separate them from HOG so that I could use different color scales. Once I was able to split them and augment the features, it worked fine.

The pipeline is pretty robust, but it may fail in city driving when there are many cars, and in many directions. It may group the cars in one big bounding box. Or, the cars coming from the opposite direction may not get detected, since the images used in feature extraction were all from the back. This could be a problem in city driving, as when a car makes a left turn in front of you. To make it more robust, I could train with more data among other things. Also, a better smoothing can be done for it to be less jittery.