Vehicle Detection and Tracking Write-up

The overall detection pipeline consists of extracting features from images in 64x64 sections, and using a Linear Support Vector Machine to classify those features as either a vehicle or non-vehicle.

The code referenced is in Final_VehicleDetectionPipeline.py.

Feature Extraction:

I decided to use three features in order to robustly classify images as either vehicles or non-vehicles. The three features I decided to use were the Histogram of Oriented Gradients, Spatial Binning of Color, and the Histograms of Color.

Histograms of Color are robust to changes in appearance. The color distribution helps account for changes in an objects orientation, and normalizing the feature makes it size invariant, making it useful to detect objects if our training set varies in the color of vehicles.

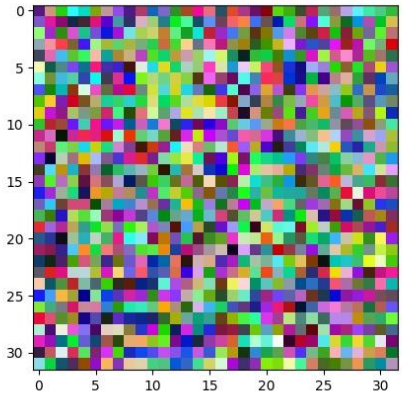

Figure 1: Spatial Binning of Color Example

Spatial Binning of Color are also quite useful as a feature vector even though it is less robust.

Histogram of Oriented Gradients (HOG) features are used to get shape information of objects in images.

Training the Model:

The first step of the project involved reading in all the file paths to the images used to train the classifier. I used 9701 car images, and 8968 non-car images. Here is an example of images in each class.

Figure 2: Car and Non-Car Image used as Training Data

All three features are extracted for every image that is a part of the training data in the extract_features function, and HOG features specifically in Final_VehicleDetectionPipeline.py Line: 109. The scikit image packages hog function used the following parameters:

Orient = 9, Pixels_per_cell = (8,8), and cells_per_block= (2,2)

These values were chosen because an orient of 9 gives a good balance between performance and size of the feature vector, our training data consists of 64x64 images and using cells that are 8x8 can gives good results. Lastly block normalization using blocks that consist of 2x2 cells helps normalize values locally while ensuring that the processing time is relatively quick.

When extracting these features, I decided to use the L channel of the LUV color space. I experimented with both the L channel, as well as the Y channel from the YCrCb color channel and found a better result from using the Lightness channel. I also found that using all 3 color channels provided a better classifier, but the processing time of extracting features would greatly increase.

After all the training data's features were extracted, the feature vectors were normalized so certain features like color values don't overwhelm the HOG feature (Final_VehicleDetectionPipeline.py Line: 161). After the feature vectors were normalized, I shuffled the training data and split off 20% as testing data. I then trained a Linear Support Vector Machine with the features and their labeled data (Final_VehicleDetectionPipeline.py Line: 182).

The trained SVM had an accuracy of 98.15% on the test data.

Sliding Window Search:

The trained SVM can now classify 64x64 images as either car or non-car images. In order to search larger images, a Sliding Window Search was implemented in the detect_vehicles function (Final_VehicleDetectionPipeline.py Line: 240-328). For computational efficiency, the HOG feature was extracted from the full image one time rather than getting the HOG for each window.

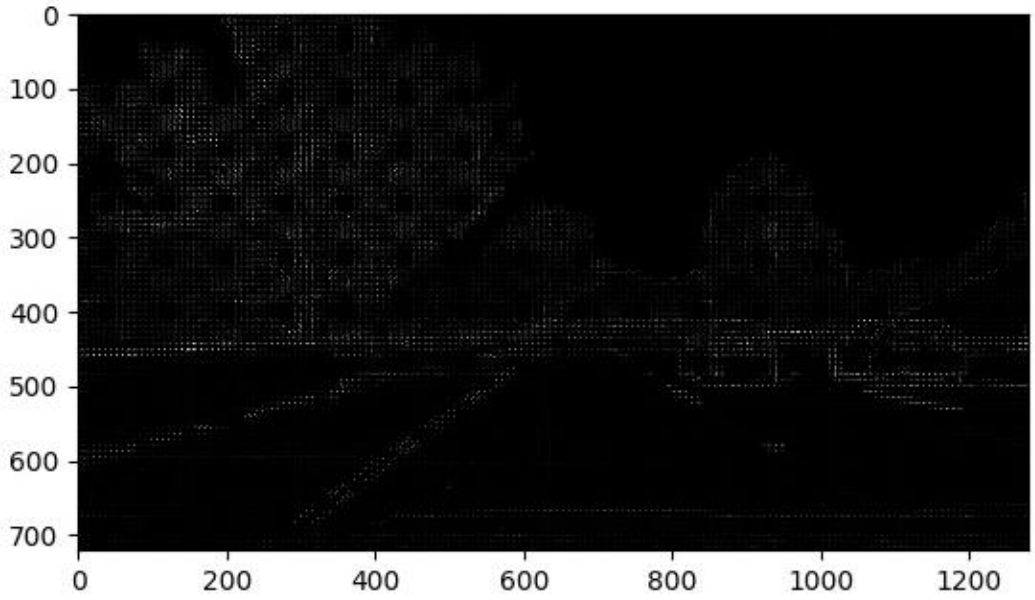

Figure 3: Histogram of Oriented Gradients for the full image

Afterwards the image was separated into its blocks, and each 64x64 section of the image was classified as either a vehicle or a non-vehicle. The detect_vehicles function returns the co-ordinates for each box that was classified as a car.

In order to effectively find cars that vary in size in a larger image, the scale of the windows searched needs to vary. I used a scale of 1, 1.5 and 2, meaning vehicles were effectively searched in 64x64, 96x96, and 128x128 windows. These scales were chosen because they detected the required vehicles in all the images well while also limiting the number of sliding windows searched. Because each window requires features to be extracted, the more windows you have, the longer it takes to detect vehicles.

To limit the number of windows searched, regions searched by each scale varied by size (Final_VehicleDetectionPipeline.py Line: 409-423), where the smaller windows would search smaller regions with some minor overlap.

I decided regions of overlap based on areas I found the SVM was having trouble classifying windows as cars and non-cars. The overlap provided a way to ensure multiple windows would need to properly detect a car when rejecting false positives later on.

Below is an image of all the windows the SVM classified as a car.

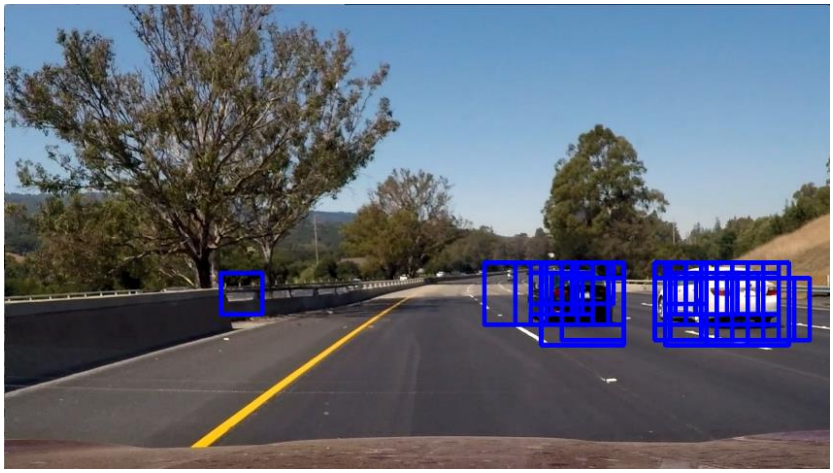

Figure 4: Windows Classified as Vehicles

Finding a balance between the number of windows searched with the time it takes to extract features from each image and classify it took a lot of testing, but I settled on a good balance with one color channel to extract features from, and limiting the area where windows search for vehicles.

Video Implementation:

The final output video is named project_videooutput.mp4. In order to remove false positives and combine overlapping bounding boxes, I created a heat map where each pixel that is enclosed in a box is given a value of 1. Pixels that are enclosed by more boxes will have higher values. The heat map values were then given a threshold to remove all false positives. (Final_VehicleDetectionPipeline.py Line: 356-399). In order to make the heat maps more robust in videos, I averaged the heat map over the last 20 frames and threshold the average.

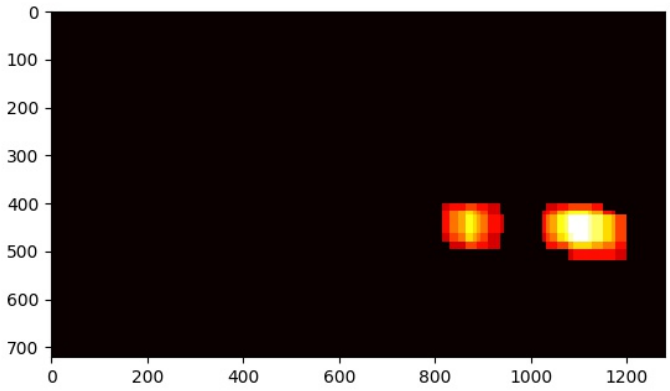

Figure 5: Heat Map of Detected Windows

I then used scipy.ndimage.measurements.label() to get the final bouding boxes for the vehicles and drew them on the output frame.

With the final bounding boxes, I also calculated its centroid to more accuratly detect where on the image the detected vehicle is. I also ensured that I average centroid values for the last 20 frames, and display them in the final video. These centroid values can be fit to a second order polynomial in order to predict future positions of each vehicle.

All tracking of data was handled with the VideoTracker class (Final_VideoDetectionPipeline.py Line: 498-625)

Figure 6: Final Output Frame of the Output Video