Self-driving car engineer nano-degree

Project 5: Vehicle detection

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, apply a color transform and append binned color features, as well as histograms of color, to HOG feature vector.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the pipeline on a video stream 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.

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

In order for the steps to be successful I first import all the required libraries that I will make use of. Then I define a function *get_hog_features,* which takes in:

- An image, on which HOG features will be extracted .
- Orient, the number of orientation bins that the gradient information will be split up into in the histogram.

- pix_per_cell, which specifies the cell size over which each gradient histogram is computed.
- cell_per_block, the local area over which the histogram counts in a given cell will be normalized.
- vis which is a Boolean in order to determine whether or not the function should return the hog image along with the features
- Feature_vec, a Boolean to automatically unroll the features

Next I define a function called *bin_spatial* which computes the binned color features given an image and a new image size.

I then define a function called *color_hist* which computes the color histogram features on each color channel separately and combines the histograms into a single feature vector.

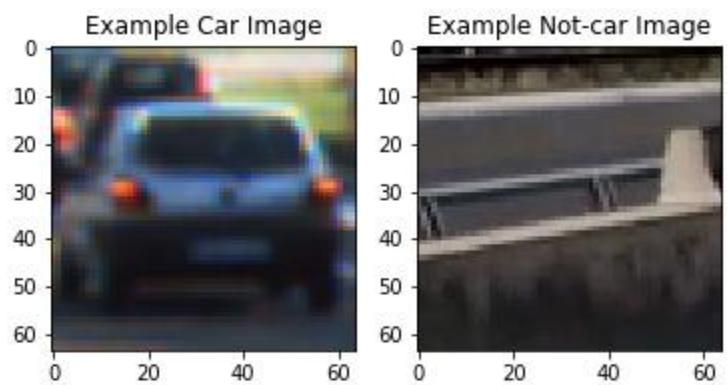
The three previously mentioned functions is called and utilized through the *extract_features* function which is used to extract features from a list of images. This function takes in:

- A list of images
- The desired color space
- The spatial size required for the *bin_spatial* function
- The number of histogram bins
- Orient
- pix_per_cell
- cell_per_block
- hog channel
- spatial_feat, Used to determine whether or not to use the bin_spacial function.

➢ hist_feat, used to determine whether or not to use the color_hist function.
➢ hog_feat, used to determine whether or not to use the get_hog_features function.

This function then returns a list of feature vectors.

Next I load all the images required for training the classifier into the cars and notcars arrays. And then display a randomly selected image from each array:

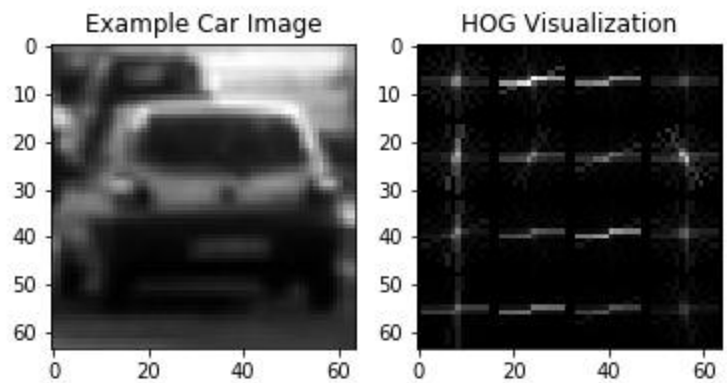


After displaying an image from each array, I then define the parameters that I have chosen for the *extract_features* function:

- color_space: YCrCb
- orient: 11
- pix_per_cell: 16
- cell_per_block: 2
- hog_channel: ALL
- special_size: (32, 32)
- hist_bins: 32

- ❖ spatial_feat: True
- ❖ hist_feat: True
- ❖ hog_feat: True
- ❖ y_start_stop: [380, 656], The minimum and maximum in y to search in slide window.

After defining the parameters I use them to extract features from the cars and notcars arrays into car_features and notcar_features. I then convert a randomly selected car image to grayscale and extract hog features from the image and then display the original along with the HOG features:



After the feature extraction I train an SVC classifier. In order to achieve this I first create an array stack of feature vectors and then fit them to the **StandardScaler** function.

I then apply the scaler to the array stack feature vectors and define the labels for the vectors.

I the split the data into randomized training and test sets with a 80-20% training-validation split. I then fit the newly split and randomized data on to the svc classifier and print/display the test accuracy obtained from the trained classifier:

```
Test Accuracy of SVC =  0.999
```

Next I define the *single_img_features* function which is similar to the *extreact_features* function except that the new function only takes in a single image.

Then I defined a *slide_window* function which returns a list of windows.

Next the *draw_boxes* function iterates through bounding boxes and draws a rectangle to on the given coordinates, and then returns the image containing the newly drawn rectangles.

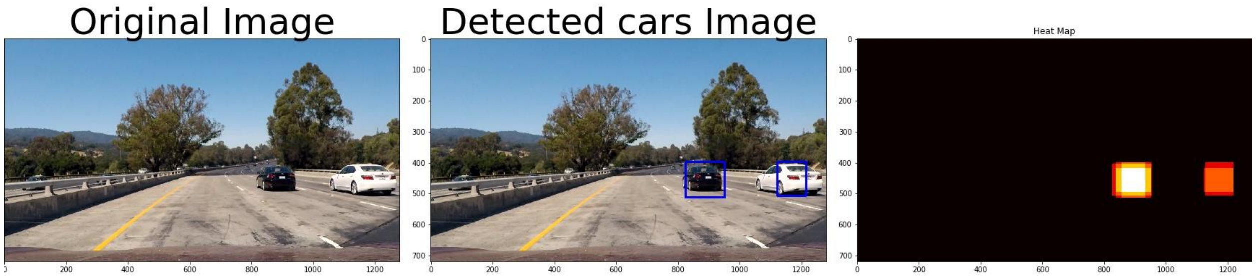
The *search_windows* function searches through the windows provided by the *slide_window* function utilizing the single_img_features function. The extracted features is then fed into the classifier and performs a prediction on the given features within the windows. Then if the prediction yield a result of “1” the specified window is then added to the on_windows array which then contains all the windows predicted to be cars.

Next I loop though all the images contained within the test_images folder to determine whether the pipeline up to this point is capable of detecting cars within each image that a car is present:

After testing the pipeline on the test images I found that it detects the cars relatively well except that a in a few places it detects cars where there are none. In order to address this problem I created three new function:

- add_heat:
  - This function iterates through a list of bounding boxes, and adds "1" for all pixels inside the bounding box. Then returns the updated heat map.
- apply_threshold:
  - This function zeros out all pixels below the selected threshold and returns a thresholded map.
- draw_labled_bboxes:
  - This function iterates through all the detected cars from the previous functions and draws a rectangle determined through the labels value.

Next I define the *vid_to_img* function which utilizes the previously created pipeline. Here once again I use all the images stored in the test_images folder to determine whether the newly created functions improve the vehicle detection pipeline:



The resulting images shows that the new functions did indeed remove the previously false detected windows.

I then proceeded to call the *vid_to_img* function on the project_video.mp4 which is saved in the output_video folder.

Conclusion

As seen by viewing the project_video.mp4 in the output_video folder the pipeline is completely capable of detecting cars within a video stream. However the pipeline does seem to struggle detecting the white cars for a couple of seconds possibly due to the car being out of range due to the y_start_stop value. Other possible improvements to the pipeline could be to increase the speed at which the pipeline performs on the video stream as it is currently taking very long to process.