Vehicle Detection Project

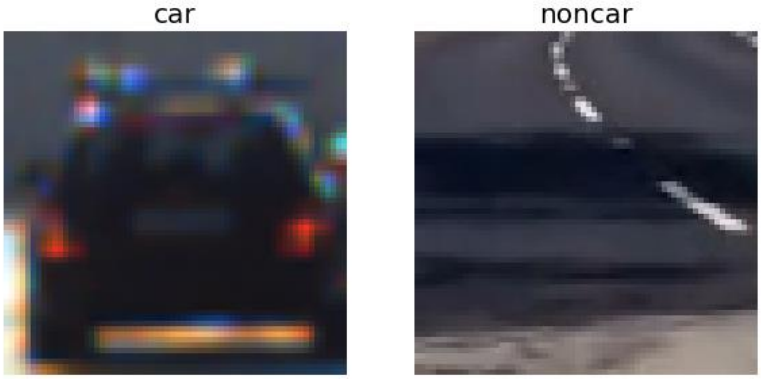
The goals / steps of this project are the following:

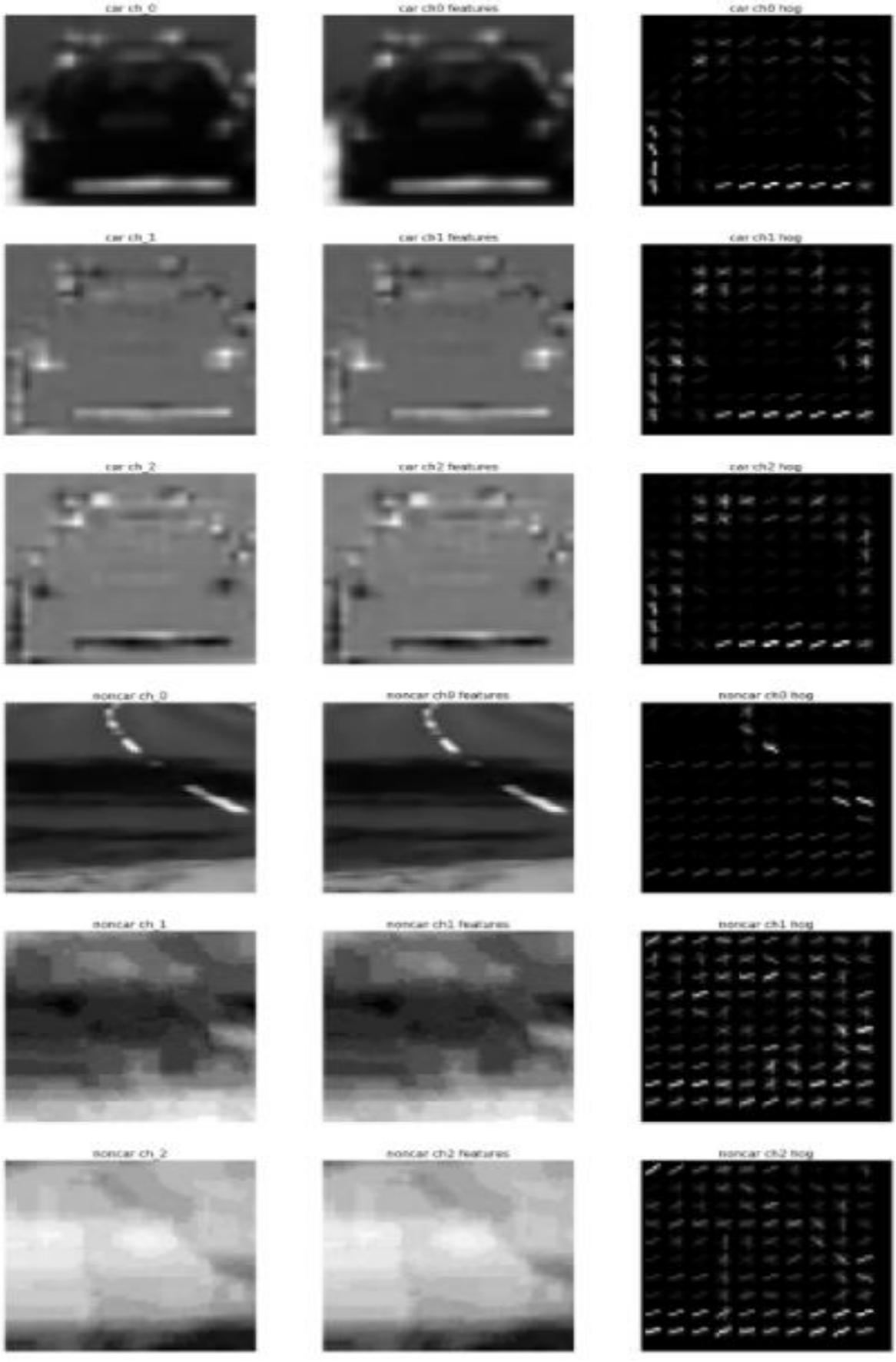
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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. Explain how you settled on your final choice of HOG parameters.**

   The code for the HOG feature extraction is in cell 4 of the IPython Notebook P5.ipynb. The hog features are extracted using the get_hog_features() function. The function takes in an image and the values for orientation limit, pixels per cell, cells per block to calculates the HOG features using the scikit-learn hog() function. Additionally, features are extracted from the image using spatial binning, color histogram using the bin_spatial(), color_hist functions respectively. The orientation, pixels per cell, cells per block were chosen to 9, 6 and 2 respectively as they gave the best results. The following image shows the HOG and spatial bining of each channel in the training images for car and non-car images. The YCrCb color space is chosen as it gives the best training and detection.




car ch_0
car ch0 features
car ch0 hog
car ch_1
car ch1 features
car ch1 hog
car ch_2
car ch2 features
car ch2 hog
noncar ch_0
noncar ch0 features
noncar ch0 hog
noncar ch_1
noncar ch1 features
noncar ch1 hog
noncar ch_2
noncar ch2 features
noncar ch2 hog

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).**

   The code for training a classifier is in cells 7, 8 of the IPython Notebook P5.ipynb. A linear support vector classifier was used to classify the training set using the LinearSVC() function. The feature set for each car and non-car image was generated by applying the bin_spatial(), color_hist() and get_hog_features() functions. All channel of the YCrCb color space were used to generate the HOG features as it gave the best results. The orientation, pixels per cell, cells per block, spatial size, histogram bin size were chosen to be 9, 6, 2, (64,64) and 32 respectively after numerous iterations. The features were split into training, validation sets and the classifier was trained with a test accuracy of 0.9865.

```
Using: 9 orientations 6 pixels per cell and 2 cells per block
Feature vector length: 21132
10.200656652450562 177.62916088104248
30.19 Seconds to train SVC...
Test Accuracy of SVC =  0.9865
```

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?**

   The code for implementing a sliding window search is in cell 10 of the IPython Notebook: P5.ipynb. The function findCars() performs the sliding window search on an input image. The input image is converted to YCrCb color space and the area where the sliding window search must be performed is cropped out. The image is then scaled by a desired factor. Multiple scale factors were used that changes the window size and helps detect different size cars. The scale factors used were [1.0, 1.2, 1.3, 1.4, 1.5, 2.0]. The pixels per cell and cells per block were chosen to be 8 and 2 in the beginning. However, the pixels per cell was changed to 6 as it improved the training and detection of vehicles.

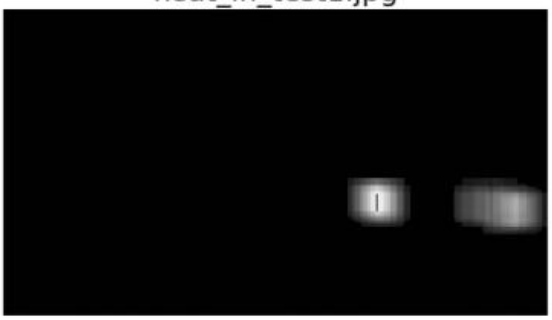
2. **Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?**

   The code for implementing the pipeline on the test images is show in cell 9, 11 of the Ipython Notebook: P5.ipynb. The classifier was trained on the YCrCb color space that gave the best results on the test images. Multiple scales of the image were used to improve the detection of different sized vehicles. A heat map of the image was generated by awarding 1 point to all pixels in each detected window. A threshold of 4 is applied to remove any false positives. Apart from this, hard negative mining of false positives is done for few frames to improve classifier performance. The functions addHeat(), apply_threshold() and vehicleTestImages are used to create heatmaps, apply the threshold and execute the pipeline on the test images.

test1.jpg

vehicles_in_test1.jpg

heat_in_test1.jpg



boundboxes_in_test1.jpg

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 link to my find video output is here.

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.**

   The pipeline for the video implements a similar technique as the pipeline for the test images. The false positives are filtered by generating a heatmap and applying a threshold to remove false positives. Additionally, for a video the heatmap is a summation of the heatmaps of the previous 5 frames and the threshold is applied on the sum of the heatmaps. This magnifies the correct detections but removes the spurious false positives. Multiple scale factors are used to improve detection. The draw_labeled_bboxes() is used to combine overlapping windows. The scipy label function is used to identify number of independent heat maps and assign labels. These labels are then used to classify the bounding boxes. The bounding box is then drawn using the minimum and maximum pixel locations in each bounding box category.

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?**

   The first issue faced was false positives in shadows which was solved by converting the images to YCrCb color space. The number of false positives is high in vehicles travelling on the opposite route. This was improved by adding heat thresholds over consecutive frames and hard negative mining of some false positives. However the pipeline still shows false positives in some locations. The pipeline also fails for a partial white vehicle. This could be improved by a robust selection of features, using multiple color spaces and increasing number of consecutive frames