Vehicle Detection Project

The goals / steps of this project are the following:

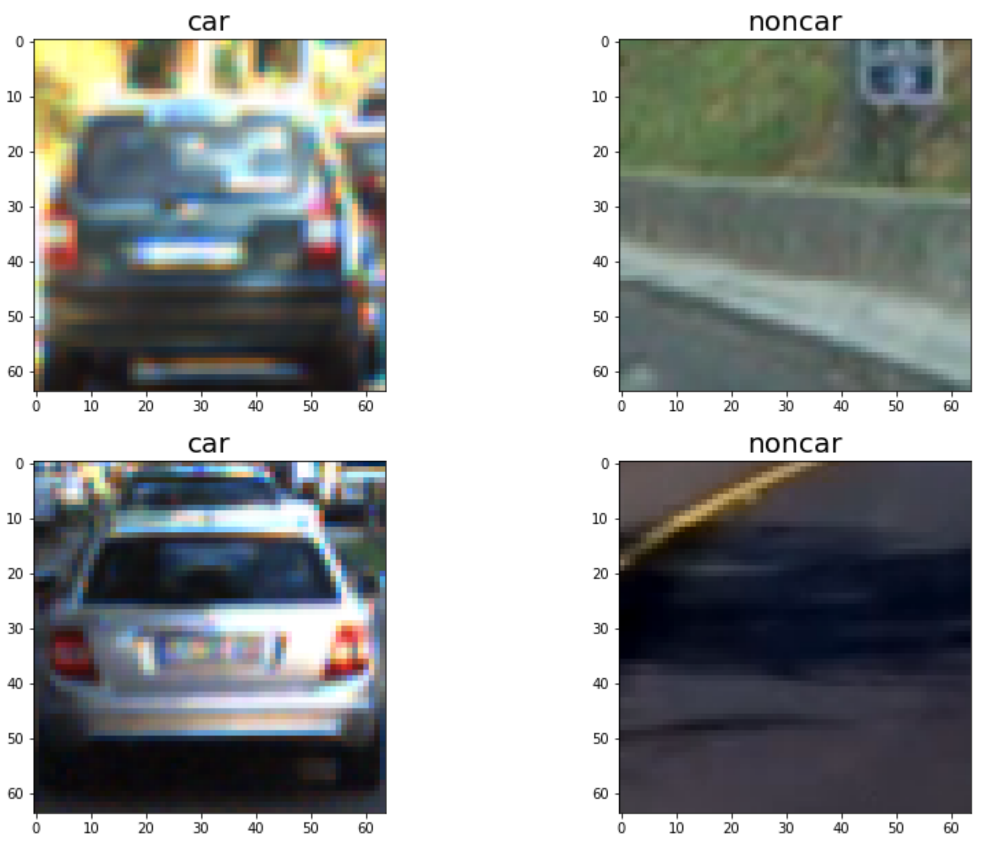
- •Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- •Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- •Note: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

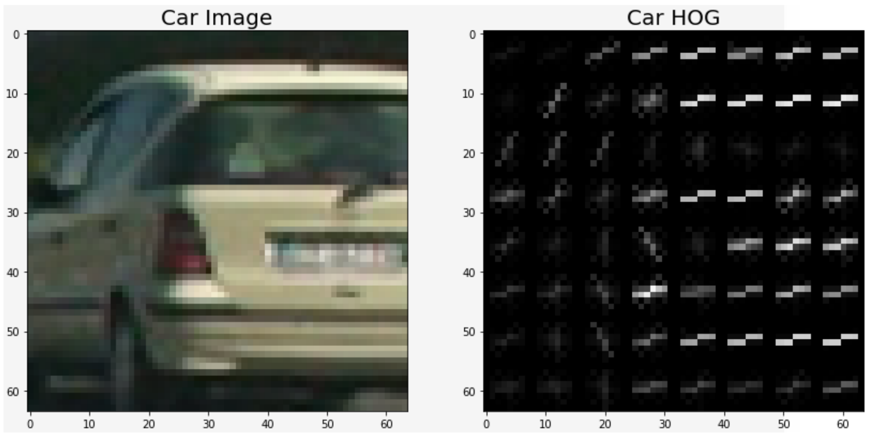
Histogram of Oriented Gradients (HOG)

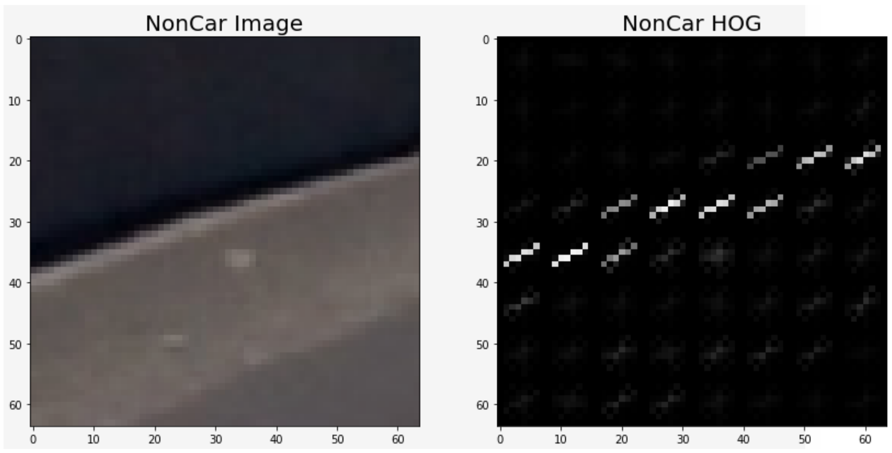
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first code cell of the IPython notebook (or in lines # through # of the file called `some_file.py`).
I started by assigning all the png images of vehicles and non-vehicles into cars and noncars respectively. The number of images in both cars and non cars are printed.
Number of images in cars = ***8792***
Number of images in noncars = ***8968***.
Few random examples of cars and noncars are plotted to confirm the classification is right. The images are read using cv2.imread and then converted from BGR to RGB format



The scikit-image hog() function takes in a single color channel or grayscaled image as input, as well as various parameters. These parameters include orientations, pixels_per_cell and cells_per_block.
get_hog_features function is defined to return HOF features and visualization in the code cell **4.** I grabbed random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.
Here is an example using the RGB color space with blue channel and HOG parameters of orientations=9, pixels_per_cell=(8, 8)and cells_per_block=(2, 2):





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

I tried various combinations of color spaces and HOG parameters with responses as training time and accuracy of the test set. The number of features played a key role with trade-off between training time / classification time and accuracy of the test set. The default HOG parameters discussed in the lecture were good enough (though not optimal) to give high accuracy and reasonable time to classify images. The time to classify is a key metric when we use window search on an actual image and eventually the video.

orientations=9, pixels_per_cell=(8, 8)and cells_per_block=(2, 2)

I tried different color spaces,one at a time, and the intention was to use ALL channels in determining the features for the images. I settled on YCrCb based on Udacity's suggestion, though I have got similar test set accuracies with HSV and YUV.

The parameters that I used are:

Parameters

cspace = 'YCrCb'

spatial_size = (32, 32)

hist_bins = 32

hist_range = (0, 256)

orient = 9

pix_per_cell = 8

cell_per_block = 2

hog_channel = "ALL"

In addition to the HOG features, I have used binned color features and color histogram features as defined in code cell **6**

Complete feature extraction is done using the function extract_features in code cell **7**, where all the features are combined to give **feature length 8460** for each image. In the following code cell, all the features of images belonging to classes: car and noncars are determined. Since mpimg is used to read png images here (0,1); later on all the test jpeg images and video pipeline frames are normalized by 255.

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 normalized features and defining the labels vector for thus image classification. It took 54.1 seconds to train SVC. The test accuracy is 98.8% with 0.00186 seconds to label an image. The relevant code is in cell 9.

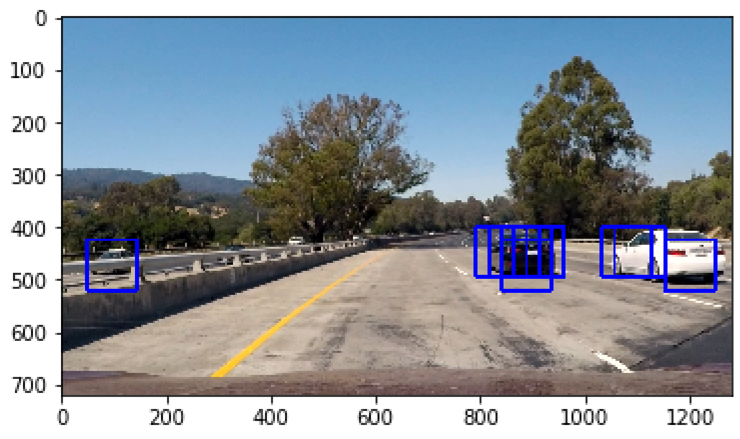
Sliding Window Search

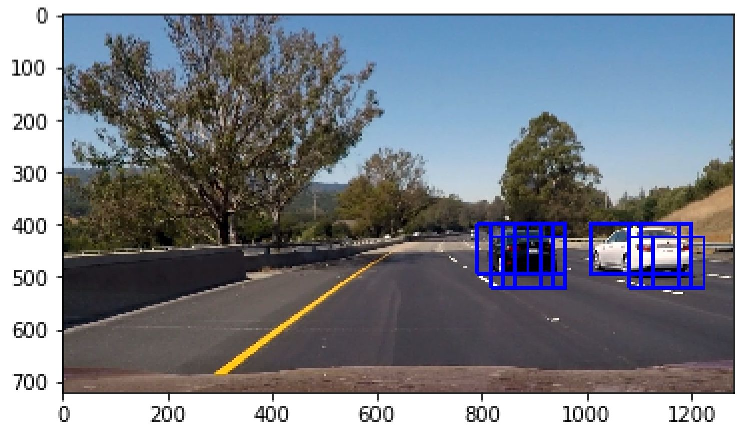
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

Here, I implemented in code cell 10, as discussed in the lecture “find_cars”, an efficient method for doing the sliding window approach, one that allows us to only have to extract the Hog features once. The find_cars only has to extract hog features once and then can be sub-sampled to get all of its overlaying windows. The image below shows the first attempt at using find_cars on one of the test images, using a single window size. I haven’t explored a lot on different scales and overlap. The search area in the image where a car can be present (400 to 656 along the y-axis), window size and overlap where the same as those used in the lesson since they worked well for detecting vehicles in test images.

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?

Ultimately I searched on a single scale using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:



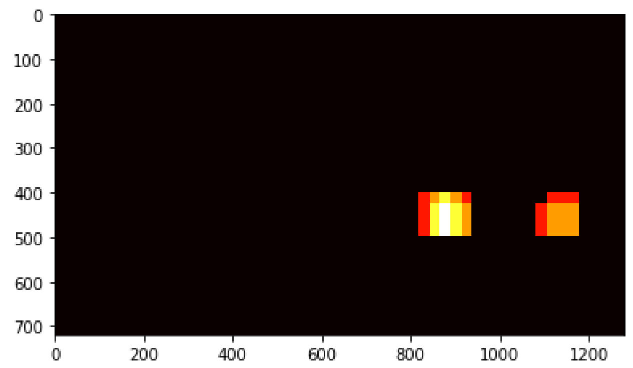
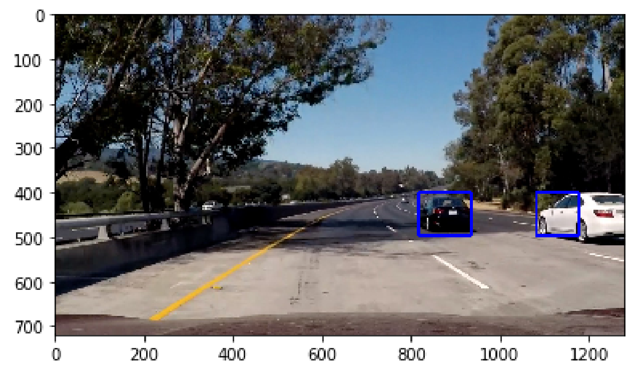
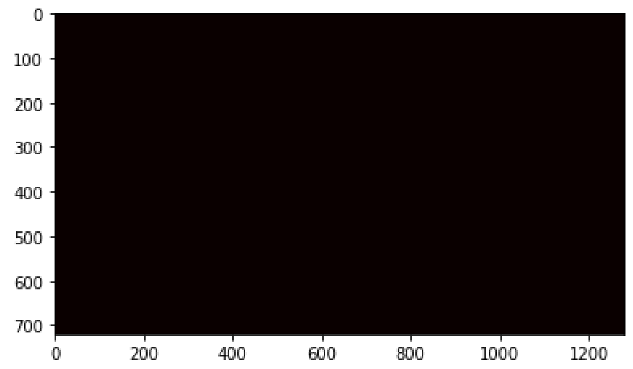


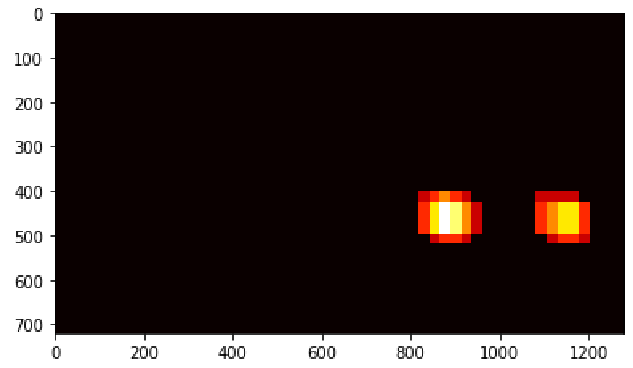
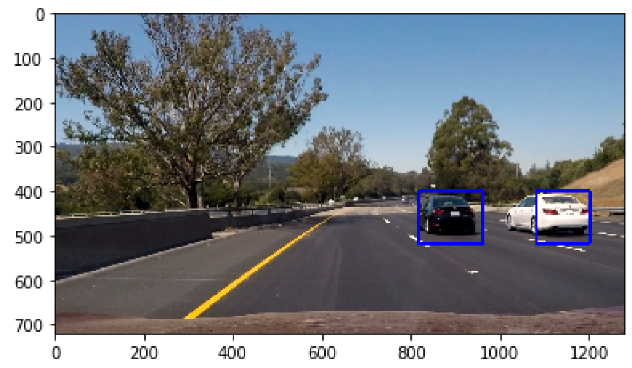
There are some false positives on the left side of the lane, as in one of the images above. A combined heatmap and threshold is used to differentiate the true and false positives. This works better when we use different size windows as the true positive is accompanied by several positive detections. In my case, I have used only one window size. I set an arbitrary threshold of 1 to remove false positives. Because a true positive is typically accompanied by several positive detections, while false positives are typically accompanied by only one or two detections, a The add_heat function increments the pixel value (referred to as "heat") of an all-black image the size of the original image at the location of each detection rectangle. Areas encompassed by more overlapping rectangles are assigned higher levels of heat. The following image is the resulting heatmap from the detections in the image above:

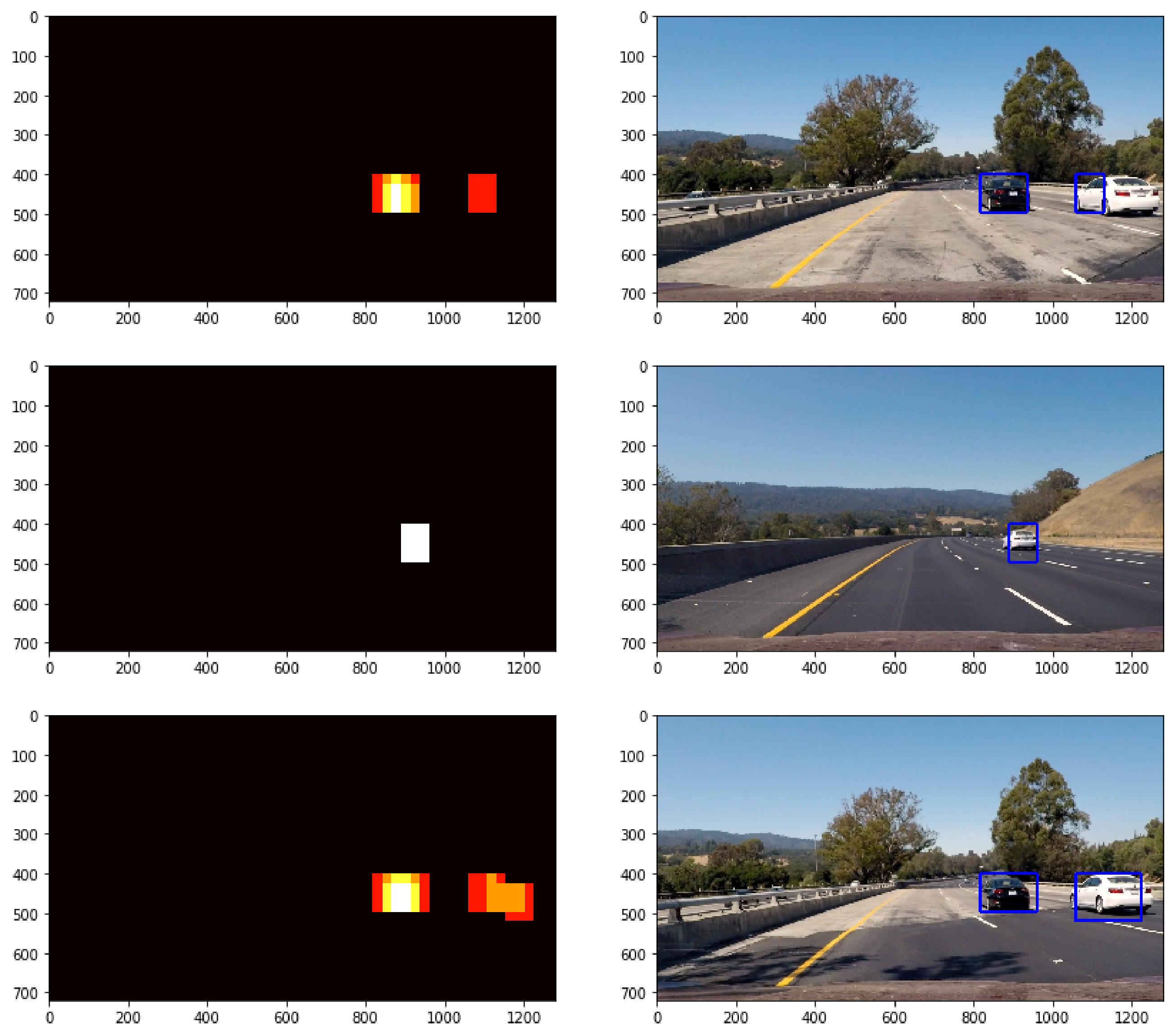
Video Implementation

1. 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 then created a heatmap (**code cell 12**) that takes in the bounding boxes per image, and for those regions in a new blank image, adds a constant value for every box. This results in identifying where the classifier predicts the highest probability of a car being present in the image. However, there are still some false positives with this heatmap. As a result, a thresholding operation is carried out to reduce those. Following is the result of this process -I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used scipy.ndimage.measurements.label() to identify 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.
Here are the results of the test images:


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



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

Video is attached

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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further

The approach that I have taken works, but I believe that there are several places where it could be improved:
The first part is image classification. I haven’t done extensive feature engineering to see what are the key features so one can work with fewer features with higher/matched accuracy. Rather, I used brute force of including a high number of features that resulted in higher test accuracy but long training time which correlates to classification time.

In general, image classification is a highly non-linear problem. Using neural network with multiple hidden layers would able to capture complex/richer features beyond histogram and gradient features.

The second part is sliding windows. I have used a fixed window size to classify cars on an image. Using variable window sizes and overlap will help to identify more bounding boxes and therefore heat maps to separate true positives from false positives (more robust). This also adds time to classify, and in a video with more than few fps will highlight issues. For example: In my pipeline, the time to find a car/label/box is longer, as it is evident in the last second of the video when a new car comes on the screen. My code wasn’t quick enough to catch that.

Also using classes to identify vehicles track their relative speed and distances would be a great addition.

In situations, where there is no contrast between a vehicle and the background. This methodology of detecting vehicles (gradients and histograms) would fail.