Project 5: Vehicle detection

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

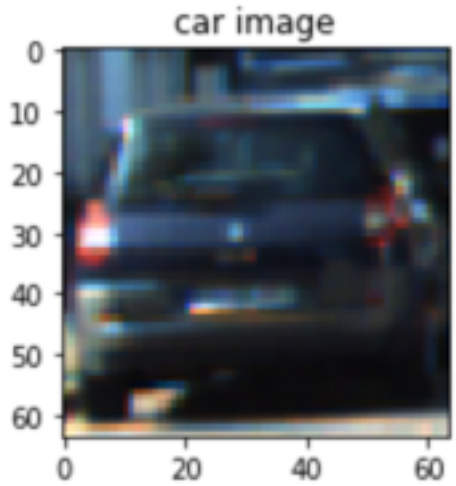
You're reading it!

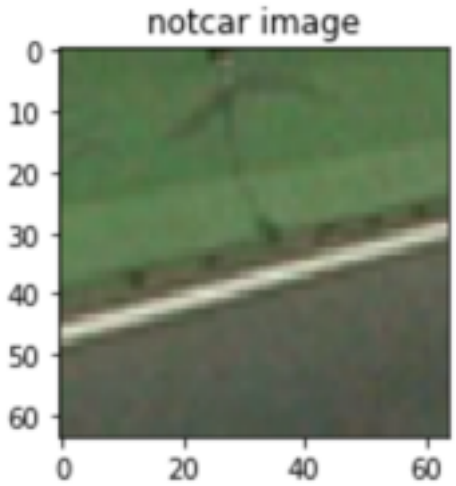
Histogram of Oriented Gradients (HOG)

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

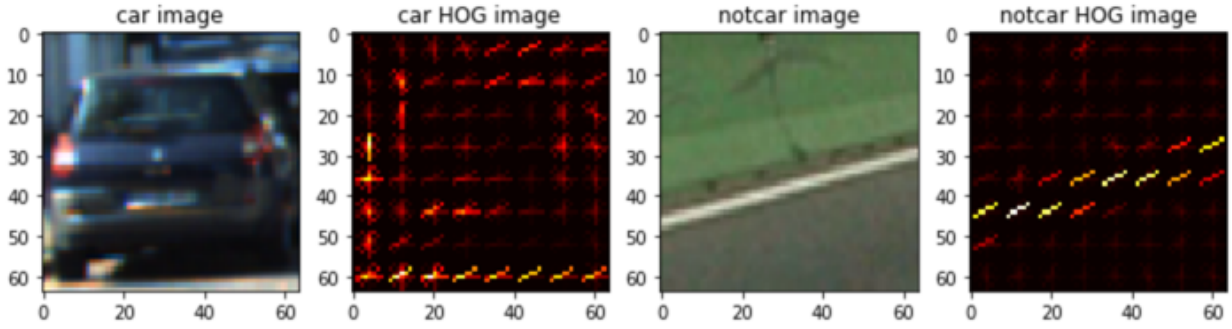
In image-detection.ipynb file, step 3: Extract HOG (Histogram of Gradients) Features contains the code(the get_hog_features function) to extract HOG features.

First, I reaad randomly a car and a not_car image. You can find one example image for each below.





Then I have set the different parameters and color spaces and inspect the results.



These images are generated with following parameters:

```
color_space = 'RGB' #can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 6
pix_per_cell = 8
cell_per_block = 2
hog_channel = 0 # can be 0,1,2 or 'ALL'
spatial_size = (16,16) #spatial binning dimensions
hist_bins = 16 #number of histogram bins
spatial_feat = True # Spatial features on or of
hist_feat = True # Histogram features on or of
hog_feat = True #HOG features on or of
```

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

I have tried several parameters and the following parameters give the best results for me.

```
Parameters

color_space = 'YCrCb' #can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL' # can be 0,1,2 or 'ALL'
spatial_size = (16,16) #spatial binning dimensions
hist_bins = 16 #number of histogram bins
spatial_feat = True # Spatial features on or of
hist_feat = True # Histogram features on or of
hog_feat = True #HOG features on or of
```

Call get_hog_features with the given parameters. We compute hog features for each individual channel. Also, I set feature_vec=False in order to calculate whole region of interest.

```
ch1 = ctrans_tosearch[:,:,0]
ch2 = ctrans_tosearch[:,:,1]
ch3 = ctrans_tosearch[:,:,2]

# compute individual channel HOG features for the entire images
hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block, feature_vec=False)
hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_vec=False)
hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_vec=False)
```

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).

First, I have extracted the features for car and notcar images. Then I stack both feature set together as one feature set. Then, I have split the data into two part as train and test and apply LinearSVC to the trian dataset. I trained the model. Finally, I used test data set to calculate the model accuracy. The accuracy of the model is close to 99% (Test accuracy of SVC = 0.9887)

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?

Now, we can classify an object whether it is a car or notcar. But, we need a method to search through an image. We can use sliding window for searching image. We we search through an image with a grid pattern and extract the same features that we trained our classifier on each window. Then, we can run the classifier to give a prediction at each step. Then, it will tell us which windows contains a car and it will draw a rectangle on the car.

While you are searching it is important to tune the parameter. You can find some important aspect of tuning parameters for sliding windows technique.

1- on an image, a car might be on some certain areas. You dont need to search for car in all image, you can just search the possible areas. It will decrease the search time and improve the performance. Since we would apply this to a video and we would like to have a solution which can be used in real car in real time. The performance is very important. For instance we can discard all pixels above and below some certain threshold.

2- All image has a perspective, closed objects seems bigger and far objects seems smaller. So we need to adjust our window size according to its position. We could make window size bigger, when it is coming closer to the bottom of the image.

3- It is important to have overlap between each window. an object still might be bigger than the window size. We can detect the whole object while sliding with overlaps.

4- while we are extrcting the hog features of the image, it is not necessary to extract image for each windows each time. Instead, we can extract the features for whole image at once and we can use this information for each window are during the sliding window search. It is very important for the performance of the system.

While we are deciding to scale of an window. There are several factors to be considered. First, What is the expected size of the image. Also, what is the desired information level. For example We probable don’t want to search pixel by pixel. According to our use case, there should be a size

which is meaningful and support our decision making. Also, size of the image is a important factor. If the image and search are is huge, it might be a good idea to keep windows size big enough. Otherwise, it might cause some performance issue since we search object in too many windows. For overlapping the windows similar consideration should be taken into consideration.

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?

You can find an output of the pipeline for the test images. I have used some techniques to optimize the performance of the classifier, since I would like to use this classifier on videos (possibly realtime.)

First, I cropped certain areas in y axis in order to decrease the search space of the image.

Second, I use the overlap size as 0.5. There might be more overlap but I decrease the overlap ratio in order to decrease window size and increase the performance of the system.

I also increased windows size. I had made many experimentation and I had found a lot of window size that has similar accuracy. And I used the bigger windows size with same accuracy in order to have both good performance and good accuracy.

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.)

There is nearly zero false positive and most of the time the cars could be identified. You can find the video in this link: https://vimeo.com/264688827

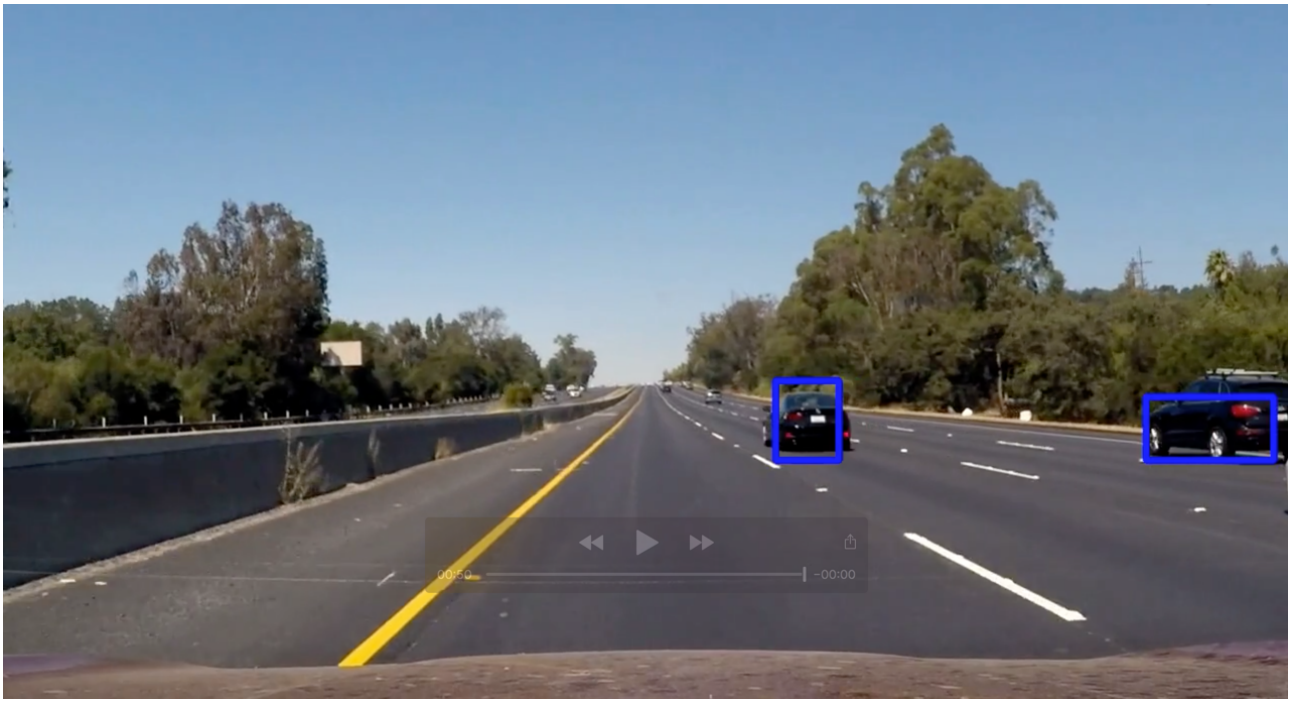
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I used a threshold (apply_threshold function) and discard if it is less than certain values. So, Most of the false positive values are disappeared. Also, I used series of frames(5 frames) and I used heatmap for a series of images. So, If a noise is pop up on a single frame, we can discard it as well. I discarded overlapping detections of the vehicles. You can find the code below.

```
from scipy.ndimage.measurements import label

from collections import deque
history = deque(maxlen=5)

def apply_threshold(heatmap, threshold):
    #zero out pixels below the threshold
    heatmap[heatmap <= threshold] = 0
    #return thresholded map
    return heatmap

def draw_labeled_bboxes(img, labels):
    #iterate through all detected cars
    for car_number in range(1, labels[1]+1):
        #Find pixels with each car_number label value
        nonzero = (labels[0] == car_number).nonzero()
        #identify x and y values of those pixels
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])

        #define a bounding box based on min/max x and y
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))

        #draw the box on the image
        cv2.rectangle(img, bbox[0], bbox[1], (0, 0, 255), 6)

    #return the image
    return img
```

```
def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

    # Return updated heatmap
    return heatmap# Iterate through list of bboxes



def process_image(img):
    out_img, heat = find_cars(img, scale)
    labels = label(heat)

    #multiple detections and false positives
    ########################################################
    bboxes = get_bboxes(labels)

    # Add heat to each box in box list
    heat = add_heat(heat,bboxes)

    #take heatmap for N consecutive frame
    history.append(heat)

    # Apply threshold to help remove false positives
    heat = apply_threshold(heat,1)

    heatmap = np.clip(heat, 0, 255)
    labels = label(heatmap)
    ########################################################

    #draw bounding boxes on a copy of the image
    draw_img = draw_labeled_bboxes(np.copy(img), labels)
    return draw_img
```

You can find an example that boxes are drawn and corresponding heatmaps are also shown.

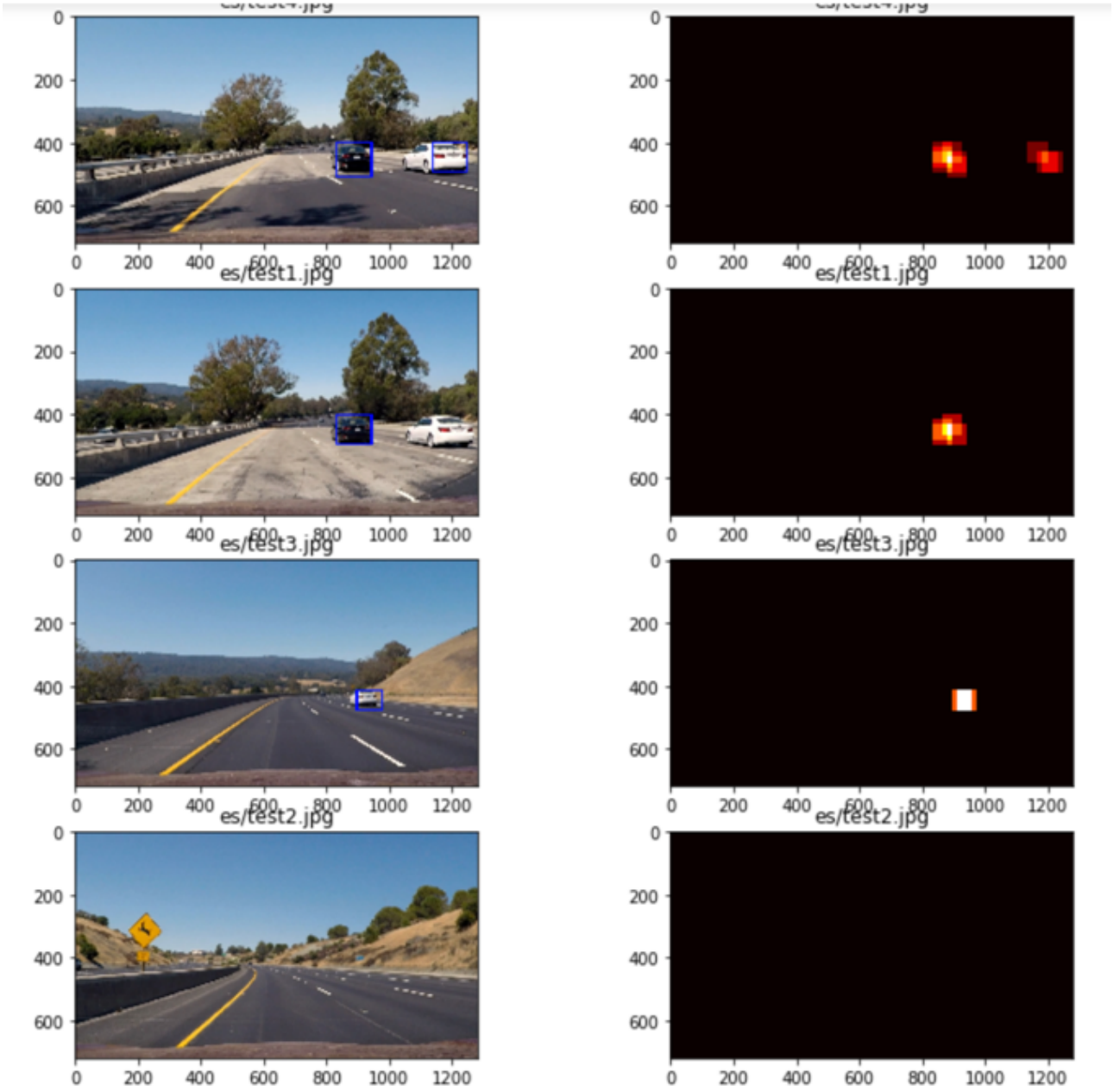


And, here you can find the last frame of the video.

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?

I believe the pipeline is quite good and we get satisfactory results. But, It could be improved further. First, we are analyzing each frame separately and it might cause some noisy result. We can take an average of the consecutive frames in order to improve the stability. Also, we can use lane line detection and understand current lane and road. So, we can just search for the road and can



understand whether a car at the same line or at another line. This information might me very helpful during decision making.