Self-Driving Car Nanodegree

P5 – Vehicle Detection

Raymond Ngiam
December 11th, 2017

I. Definition

Problem Statement

The goal of this project is to create a software pipeline to identify vehicles from video stream of a front-facing camera on a car. The steps of this project include the following:

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
2. Apply a color transform and append histograms of color, as well as binned color features, to the HOG feature vector.
3. Normalize the features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
5. 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.
6. Estimate a bounding box for vehicles detected.

II. Methodology

Feature Extraction

Histogram of Oriented Gradient (HOG) [1] will be used for extracting feature to use as input for the classifier.

The HOG approach first computes the gradient magnitude and direction for each pixel in the image. Then, these outputs are grouped into small cells, e.g. 8x8 pixel cells. Within these cells, a histogram of the gradient orientation is computed. Note that the histogram is not strictly a count of the

number of samples in each orientation. Instead gradient magnitude of each sample is summed up, so that stronger gradients contribute more weight to their orientation bin. A concept of block normalization is also introduced to improve the robustness to intensity variations in the image.

In essence, HOG features serve to provide a descriptor or signature for a given shape or structure in an image which is robust to small variations in the shape. Figure 1 below shows the HOG feature visualization for one of the vehicle images (of size 64 x 64 pixel) using the Luminance channel from LUV color space. The HOG features are extracted using cell size of 8 x 8 pixel, block size of 2 x 2 cell, and orientation bin of 12. Thus a 1D feature vector of length 7 x 7 x 2 x 2 x 12 = 2352 is produced.

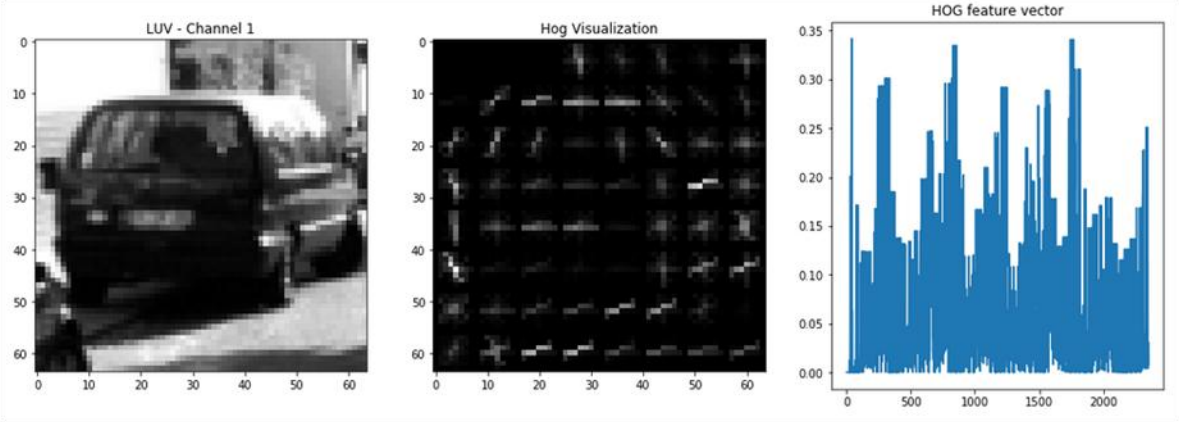

Figure 1 – HOG features.

In addition to HOG features, two more image features, namely histograms of color and binned color features are used as well. These two features serve to provide spatial and color information to complement the HOG feature.

As its name implies, histograms of color features are generated by concatenating intensity histograms of the three color channels for a particular color space. This image feature captures only the color information in the image as spatial information are lost during binning of the histogram and only the count of the intensity values are preserved. The related parameters for this image feature are the color space to be used and the bin size of the histograms. Figure 2 shows the histograms of color features for the vehicle image using the LUV color space and bin size of 32.

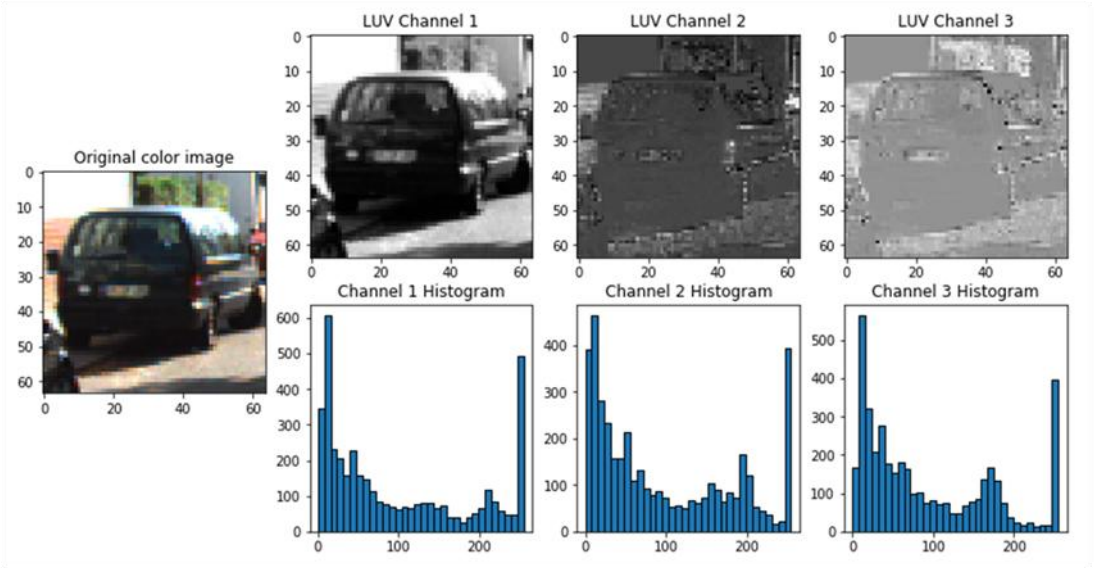

Figure 2 – Histograms of color features.

Binned color features are produced by concatenating the flattened 1D intensity arrays of the three color channel images. In contrast to histograms of color, binned color features preserve both color and spatial information in the image. However, a 64 x 64 pixel image would generate 1D feature of length 64 x 64 x 3 = 12,288. It is thus more practical to resize the image to smaller dimension prior to creating the binned color features. Figure 3 below shows the binned color features extracted for a 24 x 24 pixel resized image in LUV color space.

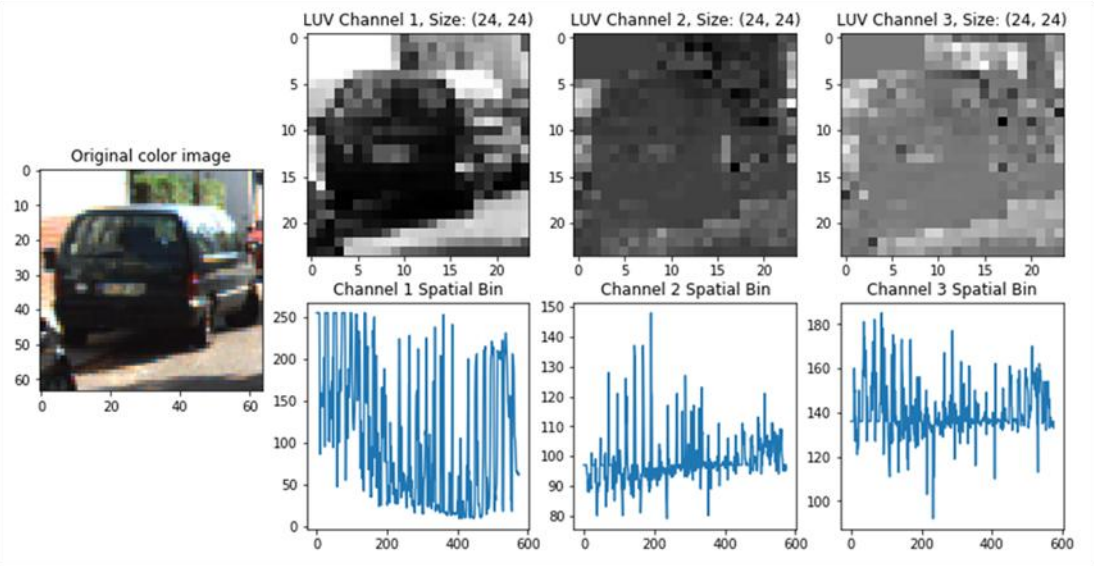

Figure 3 – Binned color features.

The parameters associated with the feature extraction stage and their possible values are as listed below:

- Color space of image to be used (RGB, HSV, HLS, LAB, LUV, YUV, YCrCb)
- Channel of color image to be used for HOG feature (1, 2, 3, All)
- Number of orientation bins (6, 9, 12)
- Pixel per cell ((4x4), (8x8), (16x16))
- Cell per block ((2x2), (4x4))
- Binned color features image size ((16x16), (24x24))
- Color histogram bin count (32, 64, 128, 256)

Different combinations of parameters are tested and visualized for some randomly selected images of "vehicle" and "non-vehicle" class. The parameter combination that produces features with a significant difference between the two classes is selected. The final selected parameters are as depicted below:

- Color space of image to be used: LUV
- Channel of color image to be used for HOG feature: 1
- Number of orientation bins: 12
- Pixel per cell: (8x8)
- Cell per block: (2x2)
- Binned color features image size: (24x24)
- Color histogram bin count: 128

Data Normalization

The feature extraction pipeline as mention above is then executed on all the "vehicle" and "non-vehicle" class images to obtain the 1D feature vectors for classification.

However, since we are using three different types of image feature, we will have different ranges of value in the feature vectors. From the example above, the HOG feature has values of range (0, 0.35), the histograms of color feature has values of range (0, 600), and the binned color feature has values of range (0, 255).

It is thus important that we normalize the feature vectors so that effect of small valued features (like the block normalized HOG features) are not overwhelmed and dominated by large valued features (e.g. histograms of color features and binned color features).

Figure 4 below shows the normalized feature vectors for samples in the "vehicle" and "non-vehicle" class.

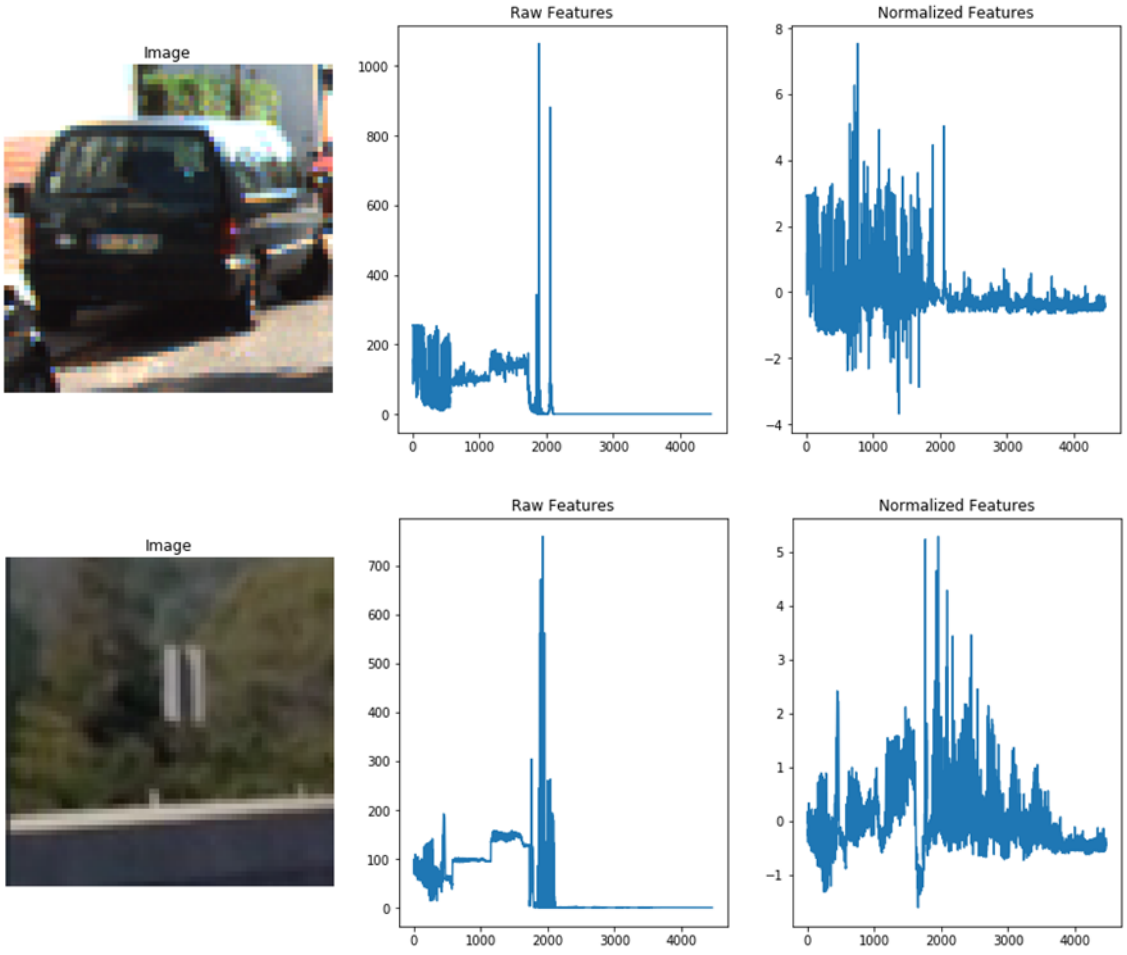

Figure 4 – Normalized feature vectors.

Train Classifier

The dataset provided for this project consists of 8792 samples for "vehicle" class and 8968 samples for "non-vehicle" class. Additional 3066 hard negative samples are generated and added to the "non-vehicle" class. (See Discussion section for details)

We will use the linear SVM classifier in scikit-learn for classification of "vehicle" versus "non-vehicle" objects in the video frames. Stratified shuffle split function in scikit-learn is used to split the dataset into training set (80%) and test set (20%), while maintaining the ratio of the "vehicle" class to "non-vehicle" class within both training and test set.

The linear SVM classifier is then trained using the training set with regularization parameter, C=0.1 while keeping other parameters at default values. The C parameter is chosen based on grid search optimization in scikit-learn.

The trained linear SVM classifier achieved accuracy of 98.30% in the test set.

Multiscale Sliding Window Search

The classifier takes in image patches of size 64 x 64 pixel and classifies whether the patches belong to a vehicle. However, due to the perspective nature of the camera, vehicles in the video may vary in size depending on their distance from the camera. We will be implementing sliding window search in the image at two different scales, i.e. 1.0 and 2.0.

Instead of using window of two different sizes, we will be using the same 64 x 64 pixel window for sliding window search of both scales.

For the sliding window search at scale 1.0, we will use the 64 x 64 pixel sliding window in the original video frame. For the sliding window search at scale 2.0, we will use the 64 x 64 pixel sliding window in the video frame scaled by 0.5 in both width and height.

We can restrict the sliding window search of scale 1.0 near the horizon of the image as we know that vehicles appear smaller further down the horizon of the image.

The green rectangle in Figure 5 represents the region where the sliding window search of scale 1.0 took place. The cyan rectangles are the 64x64 pixel windows with 0% overlap to illustrate the size of the window without confusion. (Note: in actual implementation, we use overlap of 75%, hence, there will be more overlapping windows that it appears here.)

Vehicles closer to the camera will appear bigger. Hence, for sliding window search of scale 2.0, we will place the region of interest on both sides of image, near to the bottom of the image.

The green rectangles in Figure 6 below represent the region where the sliding window search of scale 2.0 took place. Similarly, the cyan rectangles are the sliding windows with 0% overlap for illustration of search window size only. We also use 75% overlap of search windows for sliding window search of scale 2.0.

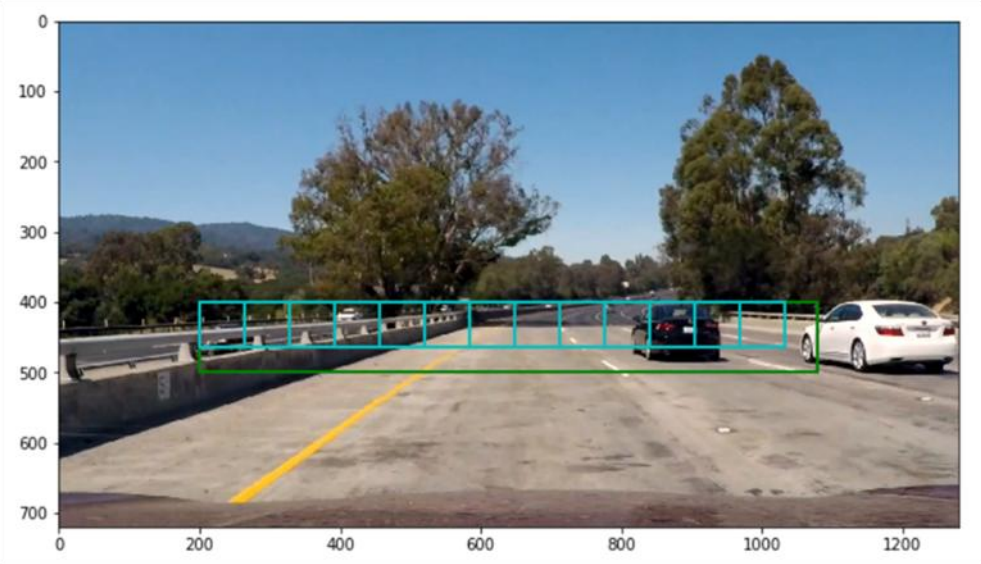

Figure 5 – Sliding window search at scale 1.0. Region of interest for the search is highlighted in green, and search windows with 0% overlap are shown in cyan within the region of interest.

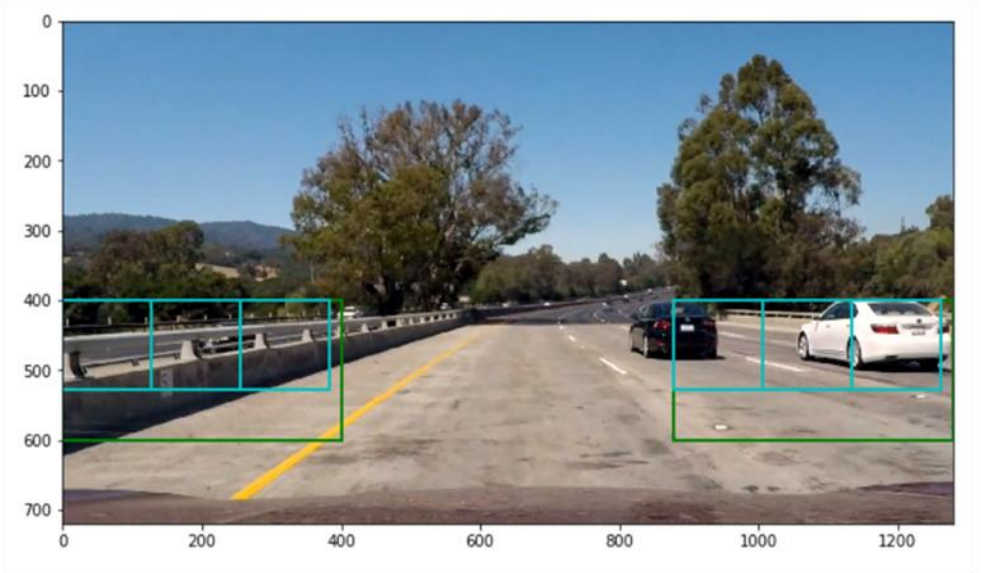

Figure 6 – Sliding window search at scale 2.0. Regions of interest for the search are highlighted in green, and search windows with 0% overlap are shown in cyan within the regions of interest.

Figure 7 below shows the classification output of the trained linear SVM classifier after the multiscale sliding window search.

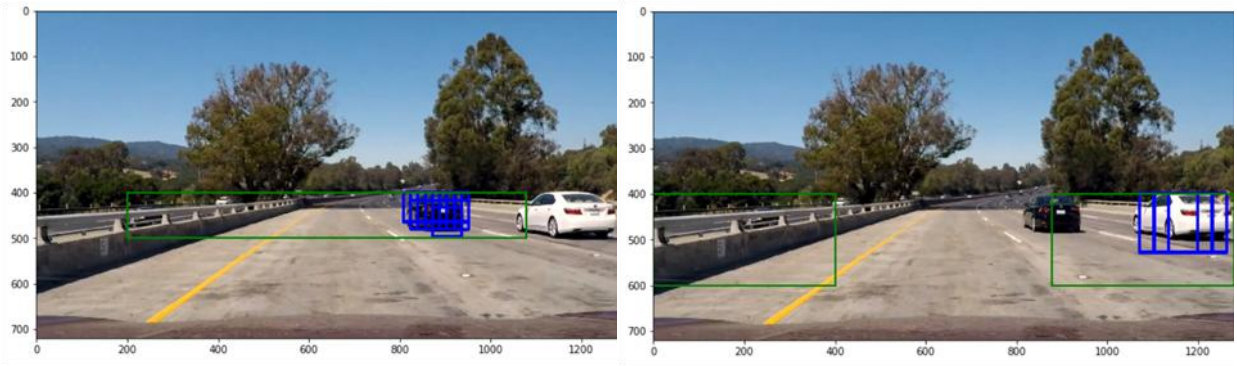

Figure 7 – Classifier detections for sliding window search of scale 1.0 and 2.0.

Combine Overlapping Detections and Remove False Positives

We use the concept of a heat map for combining overlapping detections and remove false positives. From the list of bounding boxes returned by the classifier, we add "heat" (+=1) for all pixels within each bounding box.

The "hot" parts (high valued) of the heat map are where we have high confidence the vehicles are located. By imposing a threshold, we can reject false positive detections that are less probable, or have less "heat".

Once we have a thresholded heat map, we will use the scipy.ndimage.measurements.label() function to determine connected components in the heat map. For each labeled region, a bounding box is returned to represent the location of the detected car.

Figure 8 shows the process flow for combining multiple detection and removing false positives using heat map thresholding.

Video Implementation

The project video is a video file with frame rate of 25 fps. However, for our purpose of real-time vehicle detection, it is not necessary to have such fine resolution in time since human capability in recognizing vehicles during driving also cannot reach this level.

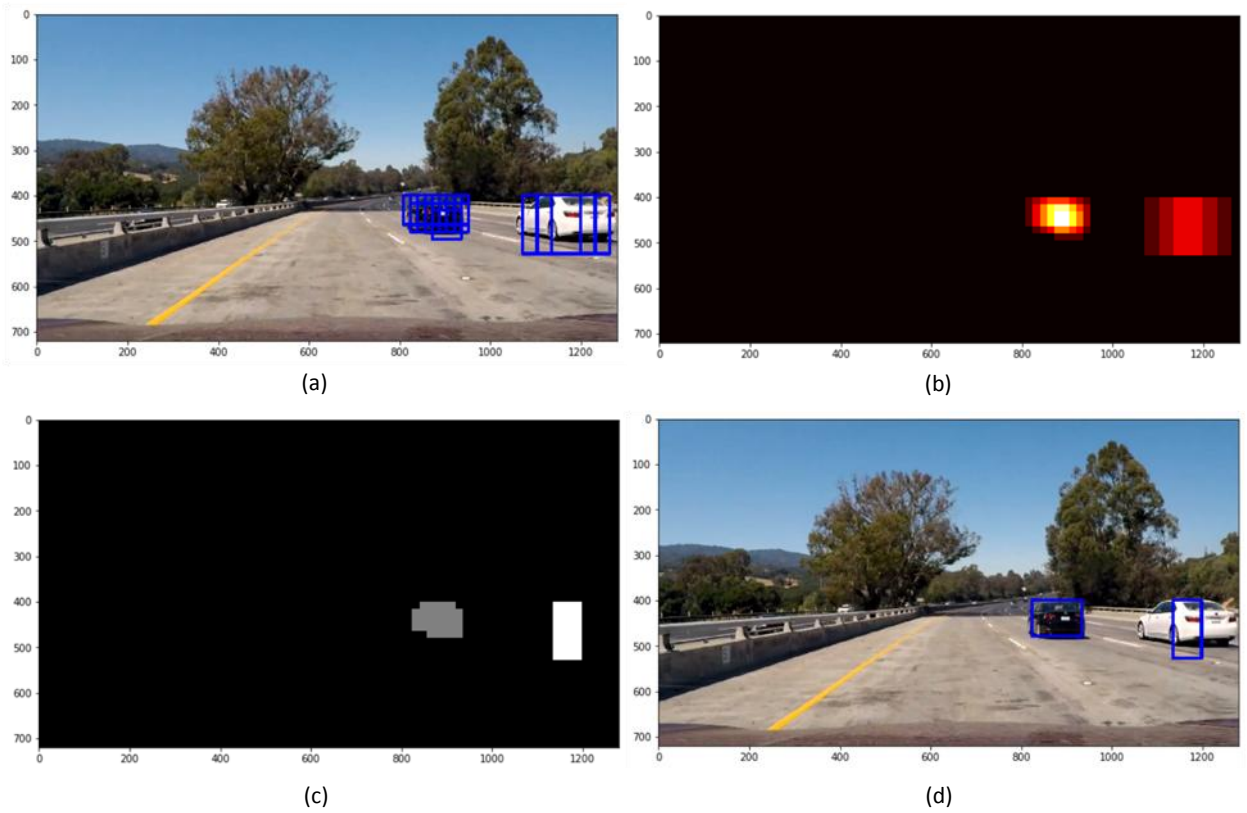

Figure 8 – (a) Raw positive detections from the linear SVM classifier. (b) Heat map generated from the positive detections. (c) Thresholded heat map using threshold value of 2.0. (d) Final bounding boxes returned.

For practical reasons, we will only process the video at frame rate of 5 fps by taking every fifth frame of the video stream. For intermediate frames, we will visualize them with the result from the last processed frame.

In addition, to further reduce the false positives, smoothing of detected heat map across multiple frames is implemented. Heat map from the last 8 processed frames are stored and an average heat map is computed. The averaged heat map is then thresholded at value of 1.3.

The inter-frame heat map smoothing contributed significantly in reducing false positives for the video being tested. The vehicle detection pipeline up to this phase is robust and consistent throughout the entire video. (See the output video file "project_video_out.mp4")

IV. Discussion

During earlier implementation of the project, the false positive rate of the linear SVM classifier is rather high and fine tuning of available parameters does not show significant improvement. One apparent reason for this issue is that the dataset we used does not contain the appropriate examples for the classifier to learn the correct decision boundaries.

To overcome this issue, we performed hard negative mining using the project video in order to increase the number of negative samples in the training dataset. The hard negative samples are created from misclassifications of the linear SVM classifier trained with the original dataset.

73 such samples are identified manually, and then data augmentation, including 4 variations in image smoothing and 5 variations in extent of white noise added, are introduced (73 x 4 x 5 = 1460 variations). 73 + 1460 = 1533 hard negative samples are generated in this way. These hard negative samples are further augmented by flipping the image around the vertical axis, thus adding a total of 2 x 1533 = 3066 hard negative samples to the training dataset. The linear SVM classifier is then retrained with the aggregated dataset.

This hard negative mining step plays a significant role in reducing the false positives of the vehicle detection pipeline.

The current pipeline made an assumption that horizon is always at the center of the image, and the ROIs for the sliding windows are hard coded based on that assumption. The pipeline might encounter failures when the car is driving on roads with significantly different inclination, where the sliding windows' ROIs are no longer valid.

The vehicle detection pipeline can be further improved by implementing Kalman filter to track detected bounding boxes across video frames. Apart from that, deep learning object detection approaches like YOLO 9000 [2] can be implemented for higher recognition performance.

V. References

[1] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

[2] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint arXiv:1612.08242 (2016).