Vehicle Detection Project

Ximing Chen

I. Goals and Steps

The goals / steps of this project are the following:

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.
- 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.

II. Loading Images

As a first step of the project, we load training images to the workspace. Notice that there are 8792 images with car in the image and 8968 images without cars. Thus, the size of positive examples and negative examples are roughly balanced, this is important as it can avoid biasing in learning. Here are examples from the vehicle and non-vehicle class, see Figure 1.

Notice that the training dataset provided for this project (i.e., vehicle and non-vehicle images) are in the .png format, whereas the example test images are in .jpg format. When loading images using matplotlib, the reader reads png images in a scale of 0 to 1 while it reads jpg images in a scale of 0 to 255. Therefore, when extracting features, one has to properly scale the images to the same range. To avoid this issue, we use imread function

(a) (b)

Figure 1: In (a), we show the an example image from the vehicle class, i.e., the set of images containing cars inside. In (b), an example image from the non-vehicle class.

from the OpenCV library and manually convert them from BGR channel to RGB channel. This step makes sure that all the inputs are consistent in color space representations.

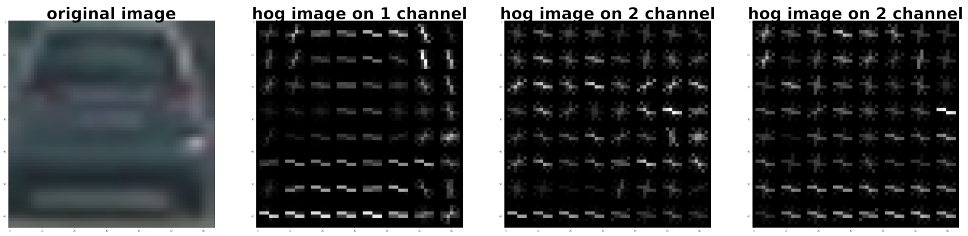
III. Feature Extraction

we aim to extract features given an image of size 64x64. To achieve this goal, we consider obtaining the histogram of oriented gradients.

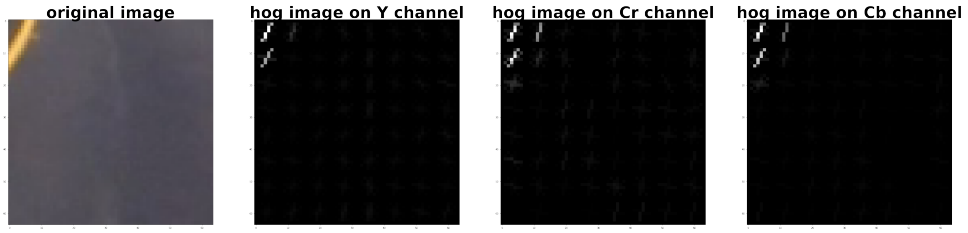
We apply the function `hog` from `skimage.feature` library to obtain the HOG feature. Basically, the algorithm of HOG follows five steps: (1) (optional) global image normalisation, (2) computing the gradient, (3) computing gradient histograms in each cell, (4) normalising across blocks and (5) flattening into a feature vector. The code for this step is contained in the third code cell of the IPython notebook.

Then, I explored different color spaces and different 'skimage.hog()' parameters ('orientations', 'pixels per cell', and 'cells per block'). I obtained 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 'YCrCb' color space and HOG parameters of `orientations` $= 8$, `pixelspercell` $= (8, 8)$ and '`cellsperblock` $= (2, 2)$:



(a)



(b)

Figure 2: In (a), we show HOG along side with the original image in the vehicle class. In (b), an example image and its HOG representations in the non-vehicle class.

We will discuss how to settle the final choice of HOG parameters in the next section.

IV. Model Learning

After features are extracted from the images, we began to use train a classifier using SVM. We follow the following steps:

First, we perform a normalization on the data. Then, we split all the data into training set and test set with split ratio equals to 0.2. Furthermore, the training and test sets are randomly shuffled. Finally, we use sklearn to fit the linear SVM using our training data. The code for these steps are shown in the 4th and 6th code block.

We have implemented the following combinations of model parameters and selected features:

We observed that adding spatially binned color and histograms of color in the feature vector improved the test accuracy. Consequently, we select the one with highest test accuracy for our model parameter setting.

Color Channel	*Orientation/Cell per block/Pixels per cell*	*Spatial feature size*	*Histogram bins*	*Testing accuracy*
HLS	9/2/8	32	32	0.9825
YUV	9/2/8	NA	NA	0.9729
YUV	11/2/16	NA	NA	0.96
YCrCb	8/2/8	16	32	0.9927

Table I: This table shows the comparison between test accuracy against different parameter settings.

V. Window Searching

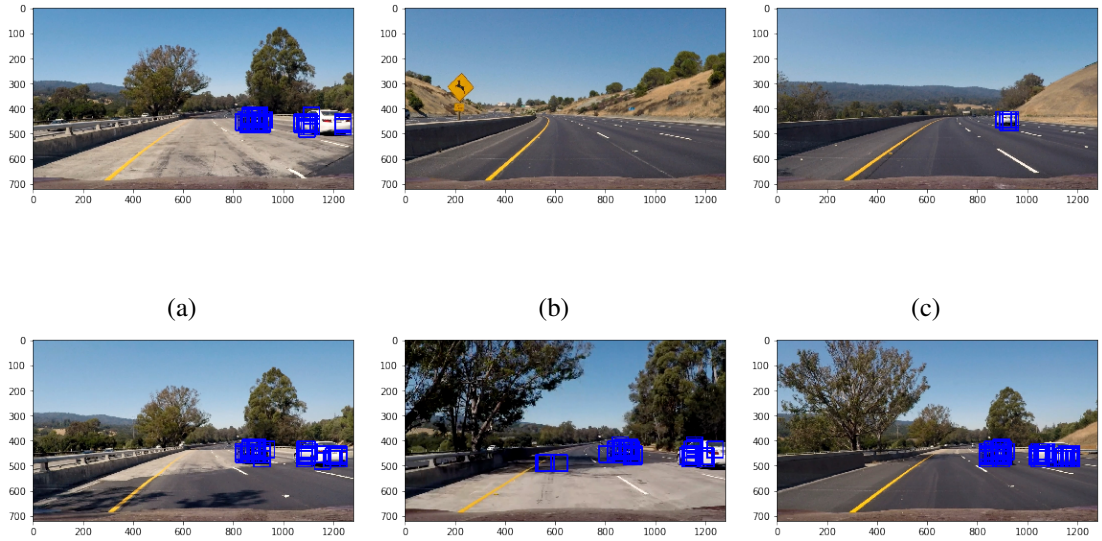
I have provided two methods for window searching. The first method is adapted from slide window method provided in the lesson material. Basically, we first select a region specified by $[xstart, xend, ystart, yend]$ to be searched. The reason to consider this selection of region is due to the cars appear only in the bottom half of the image in most of the cases. In our case, we set the region to be searched as:$(0, 350)$ (top left corner) to $(sizeofimage, 680)$. The window size is set of be 64 and there are 85 percent overlapping between two adjacent windows. Therefore, given a window, i.e., a subimage of the test image, we extract the features of the window and use our trained SVM to determine whether there is car within the window. After performing the above step for all the windows, we obtain a list of windows that detects cars.

However, the above step may need to recompute feature extraction step over and over again. To address this issue, we adapted the method find cars from the lesson materials. In this method, instead of performing feature extraction on each window individually, the HOG features are extracted once on the entire searching area and the features sub-sampled/windowed according to the size of the window. The rest of the pipeline follows from the first method.

We show a few examples on our sliding window search method.

VI. Improvements

To combine overlapping boxes, we use the heapmap. Essentially, given a list of boxes that are labelled with 1, i.e., car identified, we set all the pixels within the box to be one. Consequently, when two boxes are overlapping, the overlapped region will have higher count than the others. Then, we apply a threshold to the heap map image. In this case, in the heap map image, a large portion of the pixel values equal to zero while there are multiple



(a) (b) (c)

(d) (e) (f)

Figure 3: The detected windows.

connected regions with positive pixel value, these positive regions are identified as the region representing the cars.

The function ‘scipy.ndimage.measurements.label()‘ is used to obtain the labels for the connected regions. Finally, we constructed bounding boxes to cover the area of each blob detected.

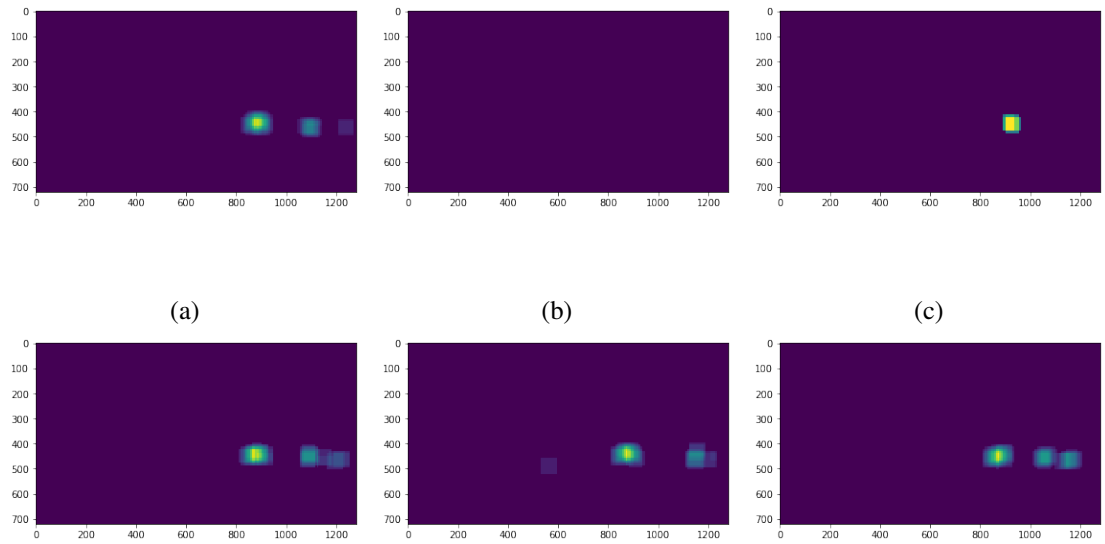
Here’s an example result showing the heatmap from a series of frames of video, the result of ‘scipy.ndimage.measurements.label()‘ and the bounding boxes then overlaid on the last frame of video, see Figure 4 and 5.

VII. Video

Please see the zip file for video.

VIII. Discussion

The constraint of current approach is the speed for detection. Although it takes around 0.006 seconds to extract the feature of a single image, the time becomes significantly larger due to the large number of sliding windows. Essentially, if we have a larger number of sliding



(a) (b) (c)

(d) (e) (f)

Figure 4: The heat images.

(a) (b) (c)

(d) (e) (f)

Figure 5: The vehicle images after using heatmap technique.

windows, it is easier to detect vehicles, however, this may lead to increase in computation time. Consequently, more sophisticated method must be applied to address this issue.

Another issue related to this project lies in the classification method, we have seen in the

table that the test accuracy across different parameter settings are very similar. However, in practice, most of the parameter settings leads to a large detection error, i.e., they either fail to detect the vehicle or they result in false positives. This being said, it probably means that the model overfits. More importantly, it may not be generalized well to arbitrary road conditions.