Vehicle Detection and Tracking*

Shanthan Suresh[1] †

March 12, 2017

1 Overview

In this project we employ Machine Learning techniques to detect vehicles on the road and track them. We use videos as input to demonstrate our solution.

2 Introduction

Vehicle detection falls in the category of object detection problem. This can be solved by neural networks based approach or conventional machine learning techniques. In this project, we employ the conventional machine learning approach.

This was a tedious exercise since we need to choose the parameters to suit the type of roads. This was done by an iterative method with a series of trial and errors.

3 Processing Pipeline

The pipeline proposed for this project is derived from the lecture videos from Udacity. The development can be split into two parts, namely, training and testing. The object detection problem is solved as an image classsification

*This project was supported by Udacity and Nvidia Graphics Pvt Ltd

†1Shanthan Suresh is with Nvidia Graphics Pvt Ltd, Bangalore, India shanthan.n2@gmail.com

problem at different positions of the image. We split up the image into several windows and check if a car is present at each window location using a SVM classifier.

- Training: Feature Extraction, Classifier
- Testing: Sliding window, Feature Extraction, Classifier

In the training phase we use a dataset that has images of cars and non-cars. This database has images of cars and non cars. Note that the training database is already processed by cropping bigger images to just get the cars and non car images. These images are 64x64 in dimension.

4 Feature Extraction

In this part of the pipeline, we extract relevant features that can be fed to the classifier. The features are items that distinguish a car from a non-car image. The features that we consider for this project are *color intensity*, *color histogram* and *HOG*.

4.1 Color Intensity Feature

The color intensity values in the image could be one set of features that distinguish a car from a non-car. For example, the car could have a certain color pattern versus the non-car which might have a random color pattern. We choose resize the image to 16x16 to get the color intensity features.

4.2 Histogram of color Feature

The color intensity feature is highly dependent on the orientation and shape of the car object. For example, if the car position changes, the color intensity pattern might change. So, we take the histogram which makes the color distribution independent of the orientation of the object, which gives us a more robust feature for classification.

4.3 Histogram of Gradients(HOG)

We also consider histogram of gradients which gives us the variation of the gradient along different directions. This will have a certain pattern for the cars, thus giving us a good feature.

5 Classifier

We use a Support Vector Machine(SVM) classifier for the classifying between cars and non-cars. We use a linear classifier and this gives us a very good result.

6 Sliding Window Search

In the testing phase we have an image that might contain car anywhere. But the classifier which we have trained is based on image which have car in the main focus. So, we cannot feed the test image directly to the classifier.

We employ a sliding window technique, wherein we move a window across the image (both horizontal and vertical directions) and crop that section of the image and apply the feature extraction, classifier on that portion of the image. If the classifier detects a car in that region of then we have a car detection at this position of the window in the image.

In this approach we slide a window over the image and at each location of the window on the image, we extract the features, run the classifier and detect if a car is present at that location. We then aggregate the decisions from all the windows to detect the presence or absence of a car in the image.

7 Video Pipeline

We employ the pipeline discussed above with some minor additions. We process the video frame by frame. In order to avoid false detection we employ a heat map, wherein we maintain a track of positions of cars detected in the previous frames. If this is consistent with the current position of the detected car, we declare it as a car. It was important to use the heat map across several images as we observed that this helps us to avoid false detections. Also this

gives some sort of smoothing the bounding boxes across images, leading to a more visually appealing results.

8 Discussion

This was an interesting project which helped us understand several aspects of object detection.

9 Rubric Criteria

In this section we address the points related to rubric criteria, for this project.

Are all required files submitted?

My project includes the following files:

* vehicle-detection.ipynb containing the code for the project.
* output_images folder containing output images of pipeline.
* extras folder containing output of different algorithms in pipeline.
* project_video_output.mp4 containing the output video.
* writeup_report.pdf summarizing the results.

Provide a Writeup / README

You are currently reading the Writeup. This has all the relevant figures and details of the solution.

9.1 Histogram of Oriented Gradients(HOG)

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.

We started off with RGB images. We extracted the HOG features on all channels. However, it was not giving that accurate a detection. We then converted the image from RGB to HSV format and experimented with the features. This showed much better results. As per the lessons we tried using just the V channel to see if it works good. However, it was not giving good results, compared to choosing the HOG from all the channels. We used the following parameters:

* Color space = HSV.
* pixels per cell = 8.
* cells per block = 2.
* orientations = 9.
* visualize = False.

The part of the code that does hog features extraction is *get_hog_features()*

Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

We used a *SVM* classifier to identify the presence of cars. We have used a linear SVM as it is quite fast. We had tried using the SVM classifier which outputs probability of a class being car. We compared this probability value with a threshold to make a decision. This turned out to be very slow in training and inference. We then switched to linear SVM which just outputs a label, given a feature vector. This is implemented in *train_classifier()*.

We split the cars,non-cars database into training and validation parts, in the ratio of 80:20. Once we extracted the features, we normalized the features to a scale of 0 to 1. This is done in *normalize_features()* We found that the classifier gives an accuracy of 99.2%.

9.2 Sliding Window Search

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?

We have used a sliding window approach to run the classifier on different portions of the image. We choose the windows at different scales and run through the image. As explained in the lessons, we do not run the window through out the image, but only the bottom half of the image. We choose square shaped windows. Initially we started with windows of size *72, 128, 192, 256*. Running on half the image size was taking lot of time. We then optimized the search by just choosing a height of 256, down from the image center. Also we observed that

192, 256 window sizes were not giving huge benefit. So, we discarded them and now have only two windows of *72, 128*. Please refer to *multiscale_search_window()* for more details. We observe that it just takes 0.4 seconds(on an average) to process each image, end to end.

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

With the classifier and seach technique described above, we ran the pipeline on an image and observed some false positives. We overcame this issue by adding heatmap with a threshold and got good results. This is done in *add_heat()* and *apply_threshold()* functions.

Fig. 1 shows the output image after running through the pipeline. It shows the cars detected correcty. We have stored this in *output_images* folder.

9.3 Pipeline(video)

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.)

We have generated the output video by running the pipeline on a video. The processing is happening frame by frame. This output video is named *project_video_output.mp4* and contains all the details that we described earlier for output image. The details include bounding boxes embedded in the output images, whenever the classifier has detected cars. This is implemented in *video_pipeline()* function.

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.

We used the heatmap across several frames to filter false positives in the images. This is done in *add_heat()* and *apply_threshold()* functions. We have a queue of size *30* to hold the bounding boxes from previous frames, to generate the heatmap for the particular frame.

10 Discussion

This project was an interesting one which involved careful choice of parameters to detect cars in videos.

We started work on this project by developing each module and doing an unit test, to make sure that the module functions as expected. Then we tied all the modules together to form the processing pipeline.

We faced a challenge initially as we failed to read a comment for reading the input images. The training set have images in *.png* format whereas the test images are in *.jpg* format. This let to cars not being detected in test images. We then normalized the intensity of the test images between 0 to 1 and this fixed the issue.

We also observed that the inference was taking a lot of time. We narrowed down this issue to the time taken in search windows function. We spent some time in analyzing this function and figured out that we need to scan only a fraction of the image portion to look out for cars. This change enabled us to process the images much faster.

Overall this turned out to be a good project to give us a feel of the challenges involved in computer vision based approaches.

10.1 Scope for further improvement

We see in our implementation that there are some sharp changes in the detected lane points towards the horizon. We feel that this could be improved further by smoothing out the filter coefficients. We so have a smoothing of the coefficients in the current code, but this could be improved.

10.2 Results

Fig. 1 to Fig. 3 depict the results of applying the pipeline on the road images. These images are placed in output_images directory of our submission.

11 Conclusions

In this project we used SVM classifier along with sliding window technique to detect the cars in an image. This was a fun project that helped us to learn

how SVM classifiers can be employed to do classification using features like HOG.

Figure 1: Output image-1 with the car detected.

Figure 2: Output image-2 with the no cars detected.

Figure 3: Output image-3 with the cars detected.