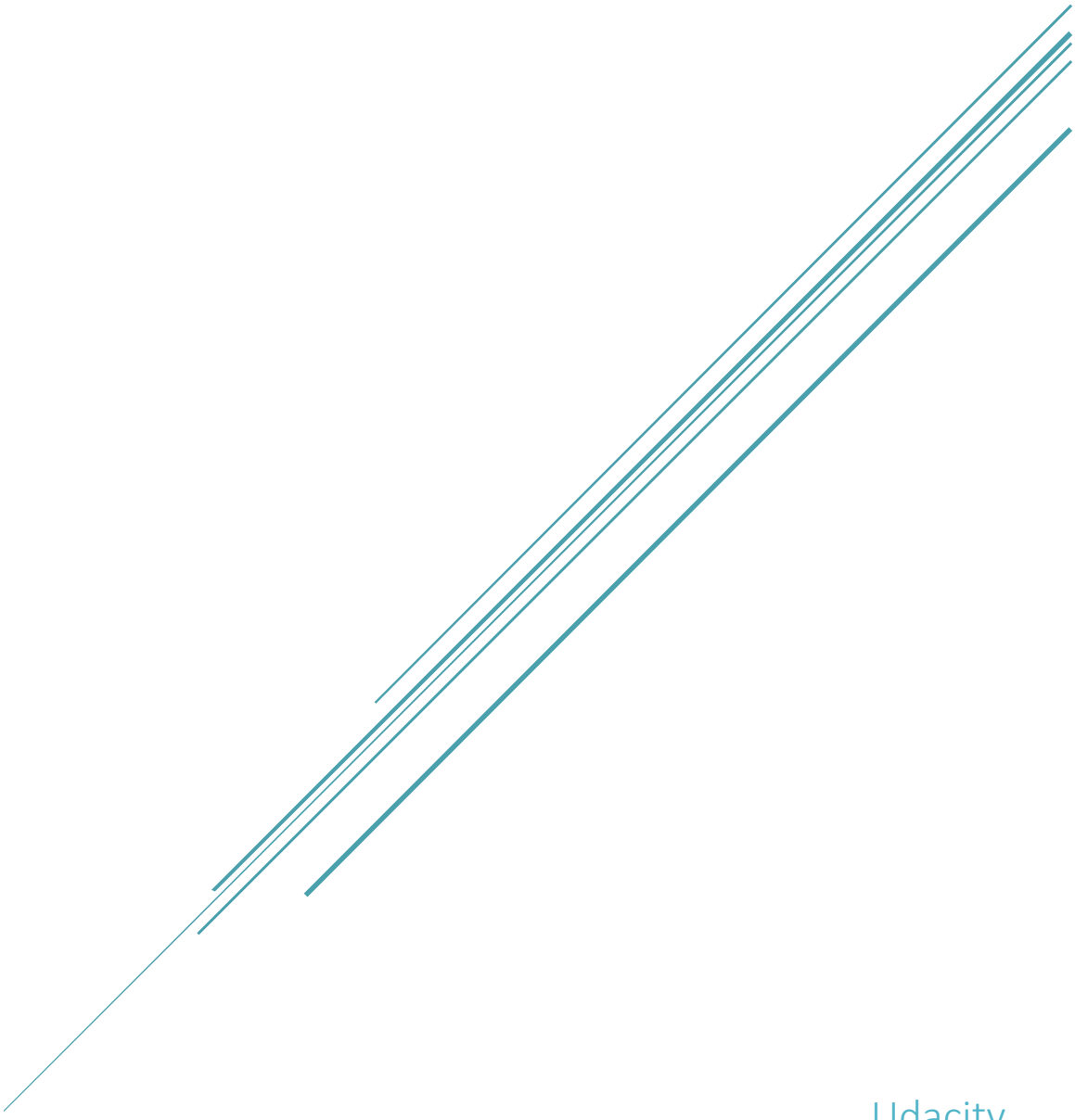
VEHICLE DETECTION PROJECT

Faisal Waris



Udacity
Self-Driving Nano Degree

Introduction

The goal of this project is to identify and mark vehicles in a camera image:

The basic steps are as follows:

1. Feature extraction using Histogram of Gradient (HOG) method from labelled images of vehicles and non-vehicles
2. Train a classifier to distinguish between car and non-car images after normalization of extracted features and cross-fold validation method
3. Implement method for multi-scale, sliding window search for identifying vehicles and marking their location in an image, which uses the trained classifier
4. Implement a pipeline to process a video file and create an annotated video file

The steps outline above are explained in detail in the subsequent sections of this report. The final section is the discussion of the results.

Training Data and HOG Feature Extraction

Data

Two training sets were used:

a) Training set provided in the project repo
b) And the additional dataset provided by Udacity, annotated by Autti

For Autti, the positive cases where extracted using the annotation CSV file and negative cases where extracted by randomly selecting 64x64 of the road, tree and sky sections.

Approx. 16K positive and 41K negative images were used for training the classifier.

The data set was split 85%/15% for training and final validation, however 5-fold cross validation was used with the 85% training set to create a robust classifier.

HOG Feature Extraction and Normalization

The final model uses HOG features from 3-color RGB values. Other color spaces such as Grayscale, HSV, HSL were also tried but did not produce as good results. The OpenCV class HOGDescriptor was used for feature extraction. This class was also later used to detect vehicles using multi-scale windows with a trained SVM classifier. This method restricts feature space but is fast and efficient. The default L2Hys normalization with a threshold of 0.2 was used with the extracted features.

Other HOG parameters used are as follows (some of them are different from the Python API):

- BlockSize: 16, 16
- BlockStride: 8 ,8
- CellSize: 8, 8
- WinSigma: 3 (smoothing parameter)
- GammaCorrection: true
- No. of Bins: 9
- Levels: 64

The parameters where selected based on the values allowed by the HOGDescriptor class and through experimentation by operating the pipeline on the test video.

Training the Classifier

After some early research into different methods, only the OpenCV SVM classifier was considered because of its compatibility with the OpenCV HOGDescriptor class.

Feature array: The HOG features were extracted for the positive and negative training images and organized into a jagged array of floats. Each element of the outer array is an array of floats and contains the HOG features from the corresponding image.

Label array: An int array was created for labels with values +1 / -1, aligned with the feature array

The feature and label arrays where shuffled together and then organized into an OpenCV TrainData structure. TrainData is compatible with the OpenCV SVM class for the "TrainAuto" method. TrainAuto allows for cross fold validation and auto tuning of the SVM hyper parameters (e.g. C, Gamma, etc.) for optimal training.

Following OpenCV sample for object detection, I chose the SVM.Types.EpsSvr which is a regression rather than classification SVM. Unfortunately, this was not trainable by TrainAuto for some error in OpenCV. I therefore used the basic Train method instead. The SVM parameters were tuned manually after trial and error. I found that using 30% of the available data gave better results and faster training times.

The overall accuracy on the out-of-sample validation dataset was 92%.

The trained SVM classifier is provided in the accompanying YAML file **veh_detect.yml**.

Vehicle Detection

OpenCV HOGDescriptor allows for a trained SVM to be plugged in for multi-scale object detection. An instance of the HOGDescriptor was created using the same parameter values as for training and then the 'detector' SVM was set via the descriptor's setSVMDetector method. The instance of the SVM was created from the YAML file saved after training.

Once configured with a detector SVM, the HOGDescriptor's DetectMultiScale method can be used to perform a sliding windows search – at different levels of a pyramid scale – to detect and locate objects in the input image. Used in this way, the HOGDescriptor class internally computes the features for the image and uses them efficiently for object detection.

There are several parameters that control the search the final values are in [brackets]:

- hitThreshold [0.1]– a configurable value that filters out weaker detections and thus can be used to tune object detection after training to reduce false positives
- winStride [8x8] – the stride of the search window, specified as Size, e.g. 16 x 16
- padding [4x4] – specified as Size; affects feature values around the edges of the tracking window
- scale [1.1] – a value that affects the pyramid scaling. This value should be in the range 1.05 to 1.5. A lower value means that more levels will be explored, at the cost reduced performance.
- groupThreshold [0] – an integer that specifies how to treat nested rectangles for grouping

The HOGDescriptor configured with the SVM detector, was first tested on a small sample of images from the training set. Figure 1 shows some example results. The classification is correct except for cases where a large part of the vehicle is cut off.

Figure 1: Object detection performed on sample positive and negative training set images; green boundary rectangle indicates successful vehicle detection

The test video was used to optimize the various parameters (e.g. HOG feature extraction, SVM training, and multi-scale detection) over many train runs.

Video Processing

For the video, each frame is processed as follows:

1) normalize the frame for feature extraction
2) Use the HOGDescriptor configured with the detector SVM to find regions of interest using the striding, multi-scale search
3) The above step returns a set of rectangles along with their strength or weight of detection. A filter is used to eliminate weak detections and false positives.
4) The resulting rectangles are further filtered for location, too high or too low ones are eliminated
5) The rectangles are then consolidated using a heat map like approach. The heat map is converted to an image the OpenCV FindContours method is used to find regions in the image.
6) Bounding boxes are determined for the regions using OpenCV BoundingRect function.
7) The resulting bounding boxes are drawn on the output video frame

The main code for the project is in the VehicleDetecHog.fsx script file. The function `testVideoDetect` implements the controlling logic for creating an output video from an input video.

Discussion

The final results shows that in the project video the white and black cars are tracked reasonably well but there are still many false positives.

The results can be improved with better tracking of the detected vehicles. Tracking was not implemented.