CS 682 FINAL PROJECT WRITE UP
VEHICLE DETECTION
BY
MITHILAESH JAYAKUMAR
(G01206238)

1. INTRODUCTION

Vehicle detection is a complex and safety-critical problem faced by self-driving vehicles, particularly given the various geometries and colors of vehicles present in the real-world. Adding to the challenge, vehicles can be occluded by other objects, including other vehicles. Moreover, it is critical that a detection system is robust to varied weather and lighting conditions.

2. RELATED WORK

Most researchers use a CNN or an algorithm such as HAAR or Histogram of Oriented Gradients (HOG) to extract features and then train a classifier such as a Support Vector Machine (SVM) or AdaBoost.

HAAR features introduced by Viola et al. [1] can be used to get a strong edge feature and it is very effective to remove the shadow of a vehicle on the road. It is used to detect the boundary of vehicles using features such as a shadow, intensity, and vertical edge[2].

The HOG feature, proposed by Dalal and Triggs[3], has gained wide recognition as a successful feature-based method, especially for real-time applications. The basic concept behind the HOG descriptor is that local object appearance and shape within an image, can be characterized by the distribution of intensity gradient magnitudes and directions. The HOG divides an image into small spatial connected regions and the local 1-D histogram of gradient directions of each pixel within each region is accumulated. These histograms are then concatenated to get the descriptors.

Modern Neural Network approach uses a CNN for object detection. [4]YOLO is an object detection pipeline based on the Neural Network. YOLO considers object detection as a regression problem by creating spatially separated bounding boxes and predicting associated class probabilities.It divides the image into an SxS grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. The output is a 1470 vector, containing probability, confidence and box coordinates.It has already been trained to identify 20 classes of objects. Compared with other object recognition methods, such as Fast R-CNN, YOLO is fast with preferably high accuracy, nearly 45 fps in base version and up to 155 fps in higher versions making it quite favourable for real-time applications.

3. THEORY

In this project, we focus on detecting the vehicles (specifically cars) present in the images or videos captured. We approach this problem with 2 methods. One is the HOG + SVM approach and another is the YOLO approach.

In the HOG + SVM approach, we identify a car using hog feature and color feature. To search the position of the car, we implement a sliding window search.

In the YOLO approach, we apply a single neural network using the darknet's weight file of the pretrained YOLO_TINY model.

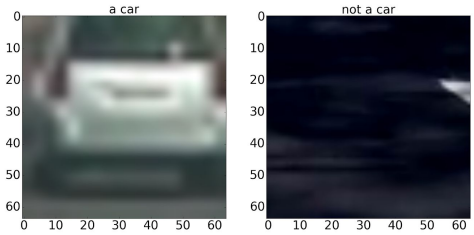
The first approach uses region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, we refine the bounding boxes to eliminate duplicate

detections making these approaches slow and hard to optimize.

In the second approach a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. The performance of both our approaches is evaluated in terms of the number of correctly classified images i.e., Vehicle or Non-Vehicle.

4. DATASET

We gather our images in each category namely “Vehicles” and “Non-Vehicles” from the GTI vehicle image database[5] and the KITTI vision benchmark suite [6].In total there are 8792 images of vehicles and 9666 images of non vehicles. A sample image from both the categories are shown below.



5. IMPLEMENTATION

a. Traditional Approach

We use a combination of Histogram of color, Spatial Bins, and HOG-Histogram of Oriented Gradients for generating features from our training data to use those features in vehicle detection.

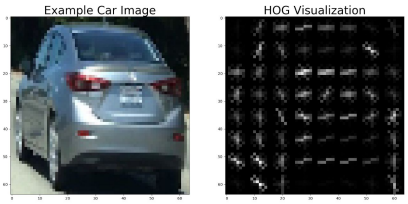
The first step is to perform a Histogram of Oriented Gradients (HOG) feature extraction on a training set of images.Then we apply a color transform and append binned color features and histograms of color to our HOG feature vector.

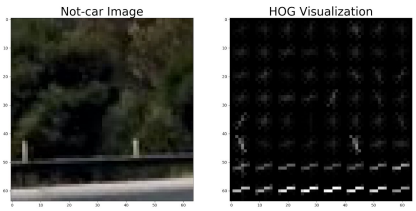
We explore different color spaces and different parameters such as orientations, number of pixels per cell, and number of cells per block from which we choose the final combination. For HLS color space the L-channel appears to be most important, followed by the S channel. We discard RGB color space, for its undesirable properties under changing light conditions. YUV and YCrCb also provided good results. There was relatively little variation in the final accuracy when running the SVM on different color spaces.

Color Space	Accuracy	Training Time (CPU)
YUV	96.75%	80 s
YCrCb	97.11%	70 s
LUV	96.23%	65s
HLS	97%	68 s
HSV	96.8%	120 s

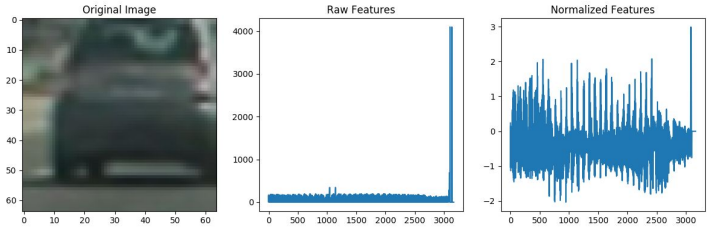
We finally chose the YCrCb as it has the highest accuracy. The following are the parameter settings and the final feature vector has a length of 6173 elements most of which are HOG features.

- YCrCb color space
- orient = 9 # HOG orientations
- pix_per_cell = 8 # HOG pixels per cell
- cell_per_block = 2 # HOG cells per block
- hog_channel = "ALL"
- spatial_size = (32, 32) # Spatial binning dimensions
- hist_bins = 32 # Number of histogram bins
- spatial_feat = True # Spatial features on or off
- hist_feat = True # Histogram features on or off
- hog_feat = True # HOG features on or off

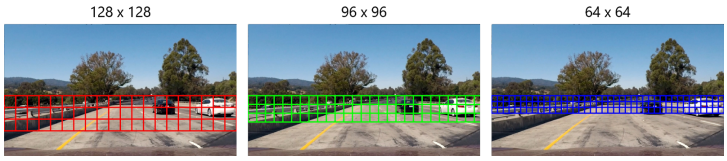




The next step is to normalize the final feature vector or else the classifier may have some bias toward the features with higher weights.

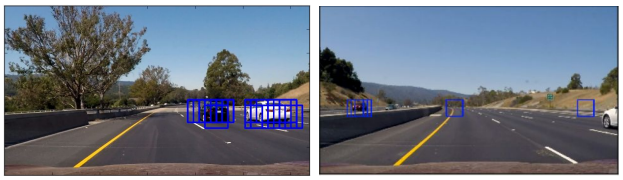


We then implement a sliding-window technique and use an SVM classifier to search for vehicles in images in each window. We use window sizes of 128x128, 96x96, 64x64 pixels size and each adjacent sliding window has an overlap of 75%. Using even slightly less than 75% overlap resulted in an unacceptably large number of false negatives. The upper half of the image and lower bonnet's position are ignored to search vehicles.

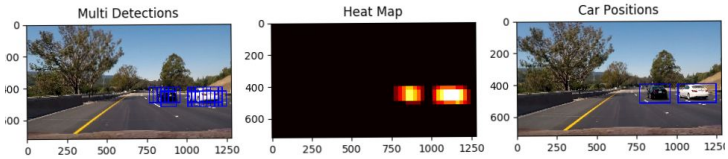


For every window, the SVM classifier is used to predict whether it contains a car nor not. If yes, save this window. In the end, a list of windows containing detected cars are obtained.

During sliding window search, there are many false positives and Multiple Detections. False positives occur on the side of the road and also simple lane lines get detected as cars from time to time. Cars driving in the opposite direction also get detected in so far as a significant portion is visible.



To resolve this problem we use heatmap. We store all the identified bounding boxes by our classifier in a list. We use a threshold to filter out the false positives and then generate a heatmap for all the windows remaining in our list. We identify individual blobs in the heatmap as each blob corresponds to a vehicle. Finally we reconstruct bounding boxes to cover the area of each blob detected.



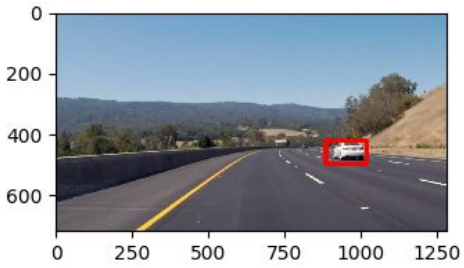
We also extend this implementation to detect vehicles in a video stream. We accumulate the heatmap for N previous frame. We then apply the same threshold which we used for the image pipeline to identify individual blobs in the heatmap.

b. Modern Approach

For comparison, we use the original YOLO implementation from the darknet website. The weights of YOLO, trained on the Common Objects in Context dataset (COCO) are also available at the darknet website[7].

The YOLO has been already trained to identify the objects like "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "dining table", "dog", "horse", "motorbike",

"person", "potted plant", "sheep", "sofa", "train", "tv monitor". We construct a convolutional neural network architecture based on Keras, containing 9 convolution layers and 3 full connected layers. We then load the pre-trained weights and detect bounding boxes for class “car” in the images.



RESULTS

For the SVM based approach, the accuracy was 96.24 when tested on 400 images containing a mix of both “Vehicle” and “Non-Vehicle” data. The total time taken to predict the class labels of 400 images was around 3 seconds (approx) whereas the YOLO gave an accuracy of 99.87 for the same test dataset. The time taken by YOLO was around 0.1 seconds which is 20 times less compared to the SVM.

A linear SVM was used due to its fast evaluation. Nonlinear kernels such as “rbf” will take not only longer time to train, but also much longer to evaluate.

Due to heatmap threshold the bounding box appears unstable. The processing time is more due to sliding window search making it not favorable for real-time application. We need a lot of engineering work to make it running real-time. As a sliding window is used, if a car is climbing a hill or if it is present a little far then it might not be detected properly.

YOLO is more preferable due to its strength against SVM's shortcomings as it has better detection and more stable bounding box with real-time processing speed. YOLO approach might fail on images with low light conditions like during night time or heavy rain or foggy day.

REFERENCES

1. Viola P., Jones M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of Computer Vision and Pattern Recognition, pp. 511–518. IEEE, Kauai (2001).
2. Han, S., Han, Y., Hahn, H.: Vehicle detection method using Haar-like features on a real time system. World Acad. Sci. Eng. Technol. 59(35),455–459 (2009).
3. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 886–893. IEEE, San Diego (2005).
4. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
5. GTI vehicle image database [http://www.gti.ssr.upm.es/data/Vehicle_database.html]
6. KITTI vision benchmark suite [http://www.cvlibs.net/datasets/kitti/]
7. TensorFlow Implementation of YOLO [https://github.com/gliese581gg/YOLO_tensorflow]