Smartphone Based Road Vehicle Detection

Summer Scholar: Minghan Wei

Advisor: Christoph Mertz

Introduction

The Navlab group has developed a smartphone-based system for infrastructure inventory and assessment. The system collects images from a vehicle(see right) and analyzes the images to find road distress and other infrastructure problems. One issue with the collected images is that they often contain vehicles. These vehicles not only obstruct the view, but there are also privacy concerns. It is desirable to either discard images with vehicles or to black out the corresponding areas in the images.

◆In this work we want to implement and train a vehicle detector using standard computer vision tools. The detector should be reasonably fast and perform comparable to the state-of-the-art. We also want to find out how many training examples we need to use.

Method

There are several well-designed algorithms to detect vehicles with high performance. As a part of a larger project, we want to balance the performance:

(1) detect most of the vehicles (high recall) with reasonable precision.

(2) The detector should take one second or less per image.

◆**We use OpenCV as our computer vision tool box.**

OpenCV is an open source computer vision and machine learning software library.

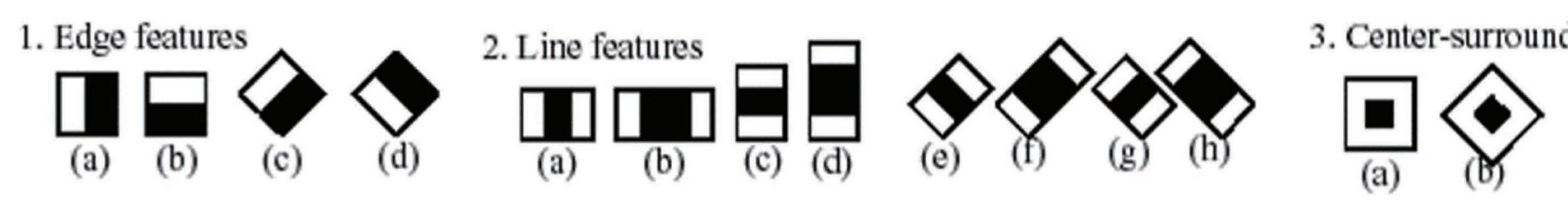
There are two steps to go:

(1) Training a classifier:

OpenCV can train a classifier using **haar** features combined with **adaboost** learning algoritm.

Haar features:

the difference of the sums of adjacent regions. (as shown below)



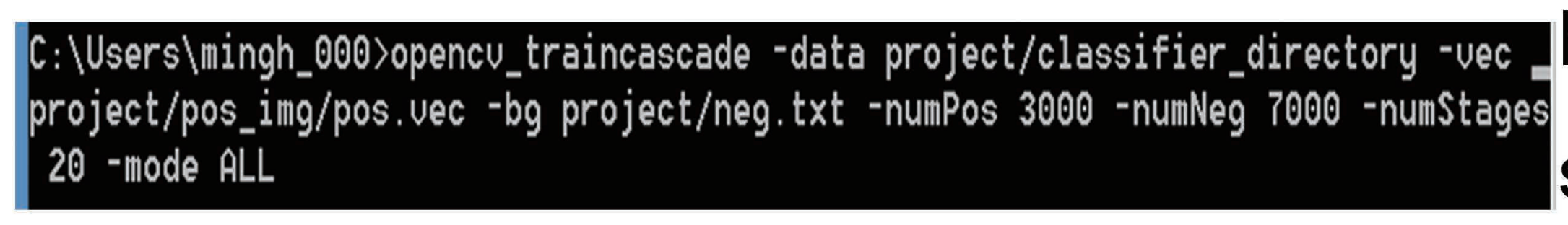
Compute haar features in differnt parts of samples and use that to represent a certain class.

Adaboost:

An adaboost classifier consists of a series of weak classifiers. Training samples which are classified wrongly in previously weak classifiers will gain more weight in the following stages.

How to run the training program:

Type in the command line as the following:



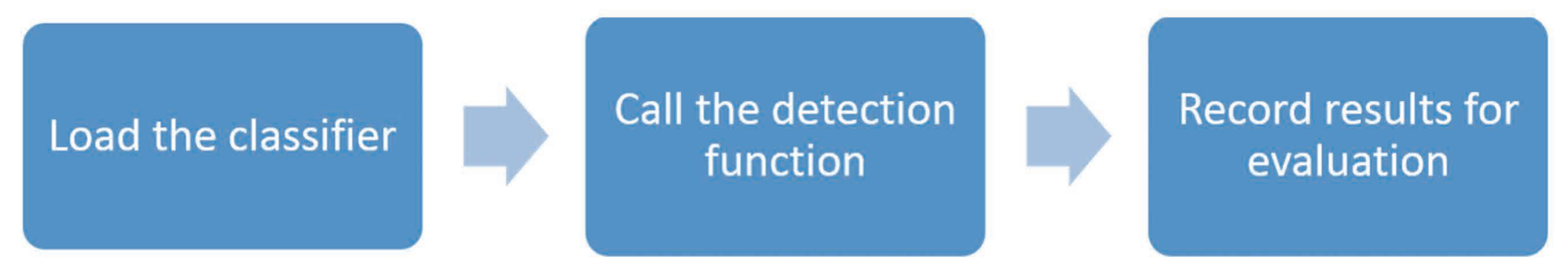
```
C:\Users\mingh_000>opencv_traincascade -data project/classifier_directory -vec
project/pos_img/pos.vec -bg project/neg.txt -numPos 3000 -numNeg 7000 -numStages
 20 -mode ALL
```

It takes about 24 hours to train when parameters are set as above on a common laptop.

See http://docs.opencv.org/doc/user_guide/ug_traincascade.html for more details of this function.

Even if someone knows little about haar or adaboost, he can get a classifier by setting proper parameters for this function.

(2) Detecting vehicles using the classifier in C codes.



We implemented this detector using OpenCV functions.

Evaluation

Visible results we get:

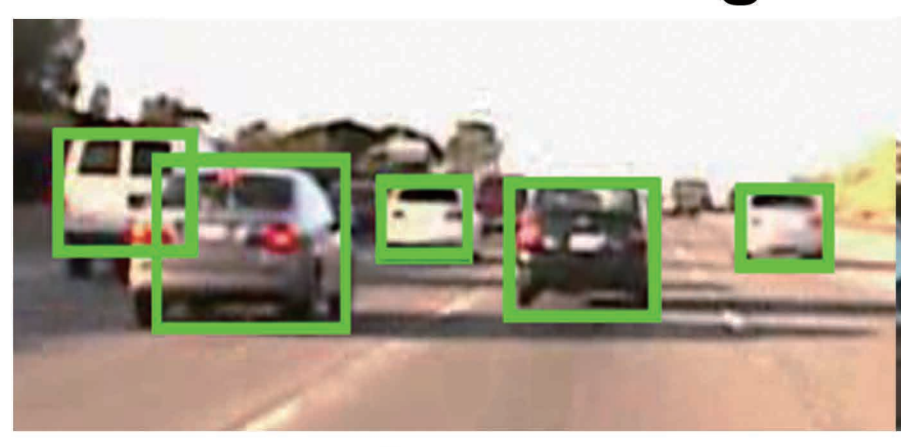


Figure a

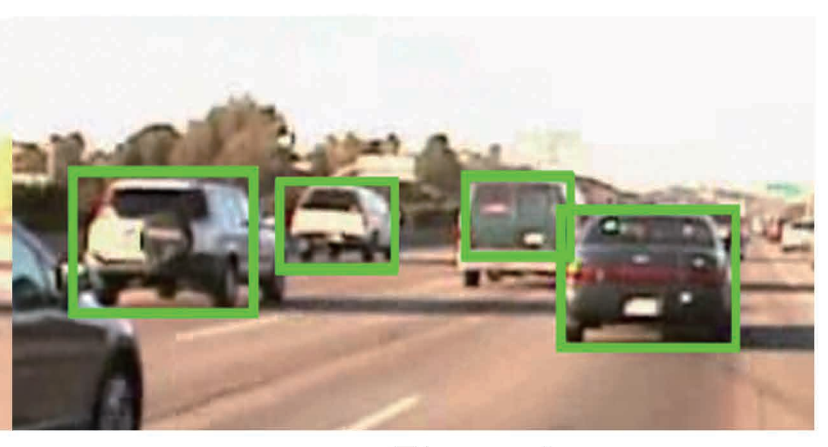


Figure b

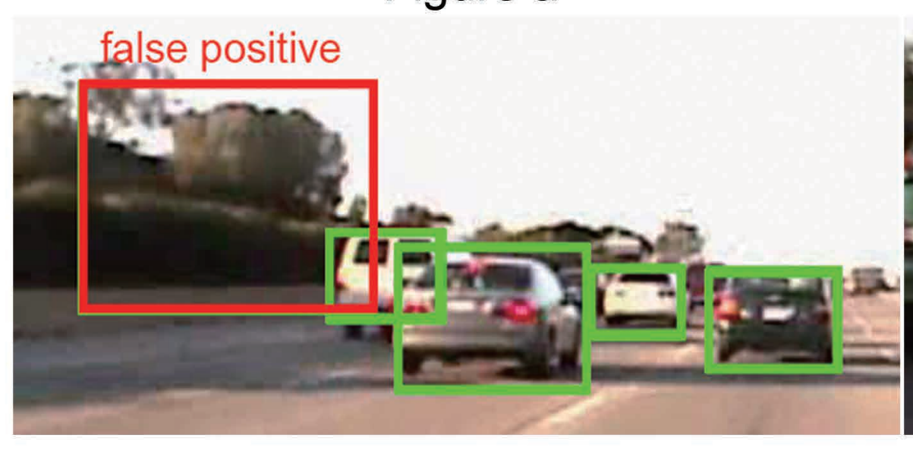


Figure c

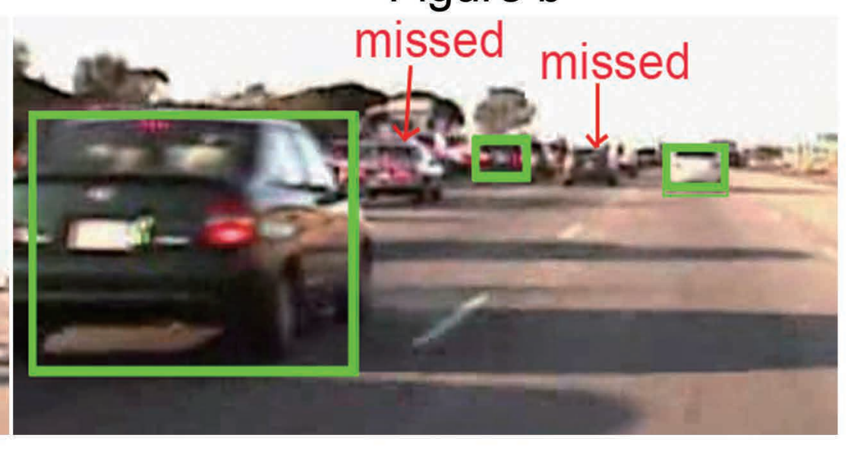


Figure d

It takes about 0.6s to process a 704*480 image on average.

Figure a, b show good detections.

In Figure c, false positive (red box) occurs due to vegetation. Other factors like shadows may also lead to this problem.

In Figure d, at least two cars are missed (see the arrows).

Figure e shows the performance of six classifiers we trained with different parameters.

Precision-Recall curves we get:

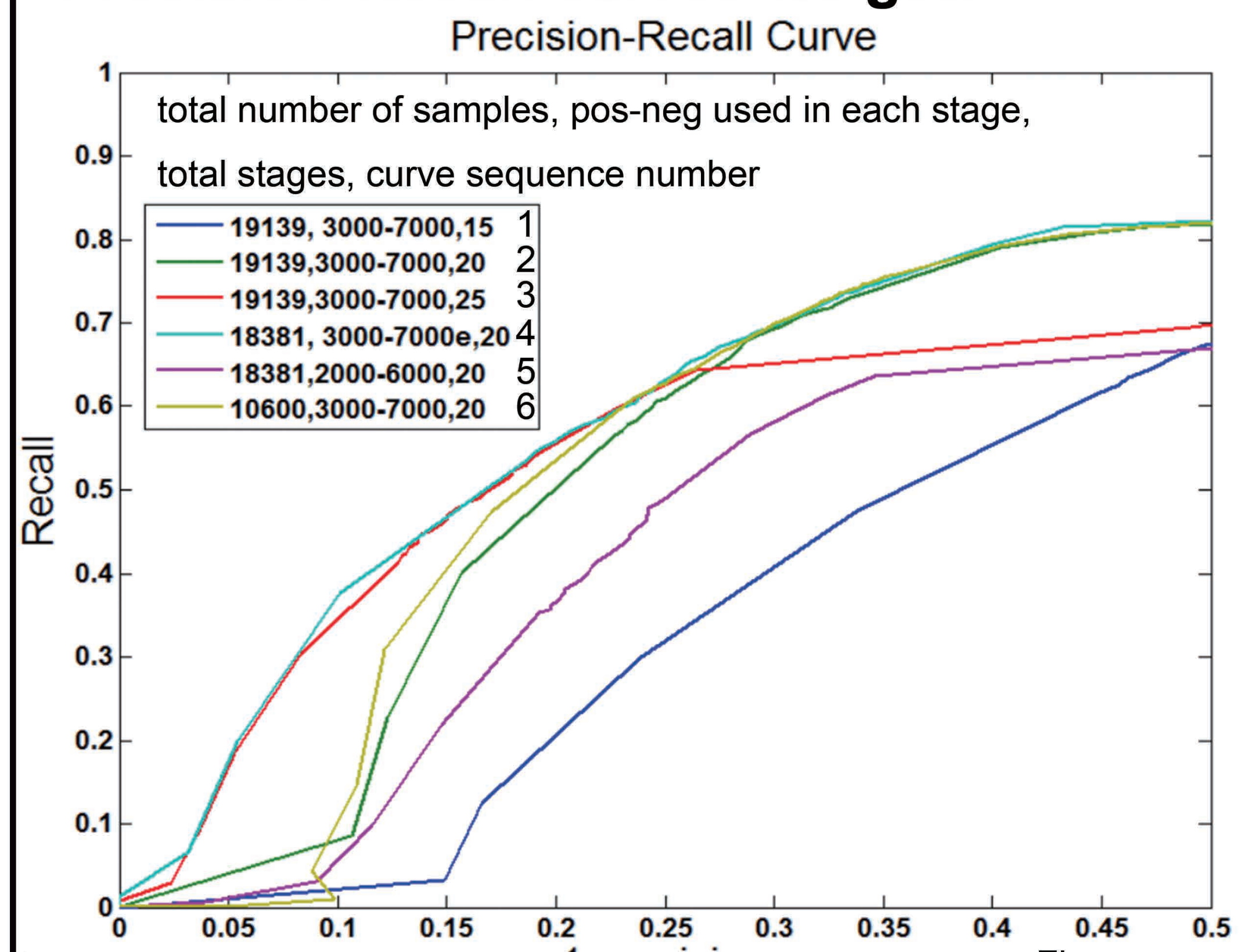


Figure e

We found:

1.More training samples in each stage can considerably improve performance (compare curve 4 and 5), while the total number of samples does not significantly affect results (compare curve 2 and 6).

2. More training stages do not guarantee a better performance. In our application, 20 stages is preferred due to its higher recall (compare curve 1,2,3).

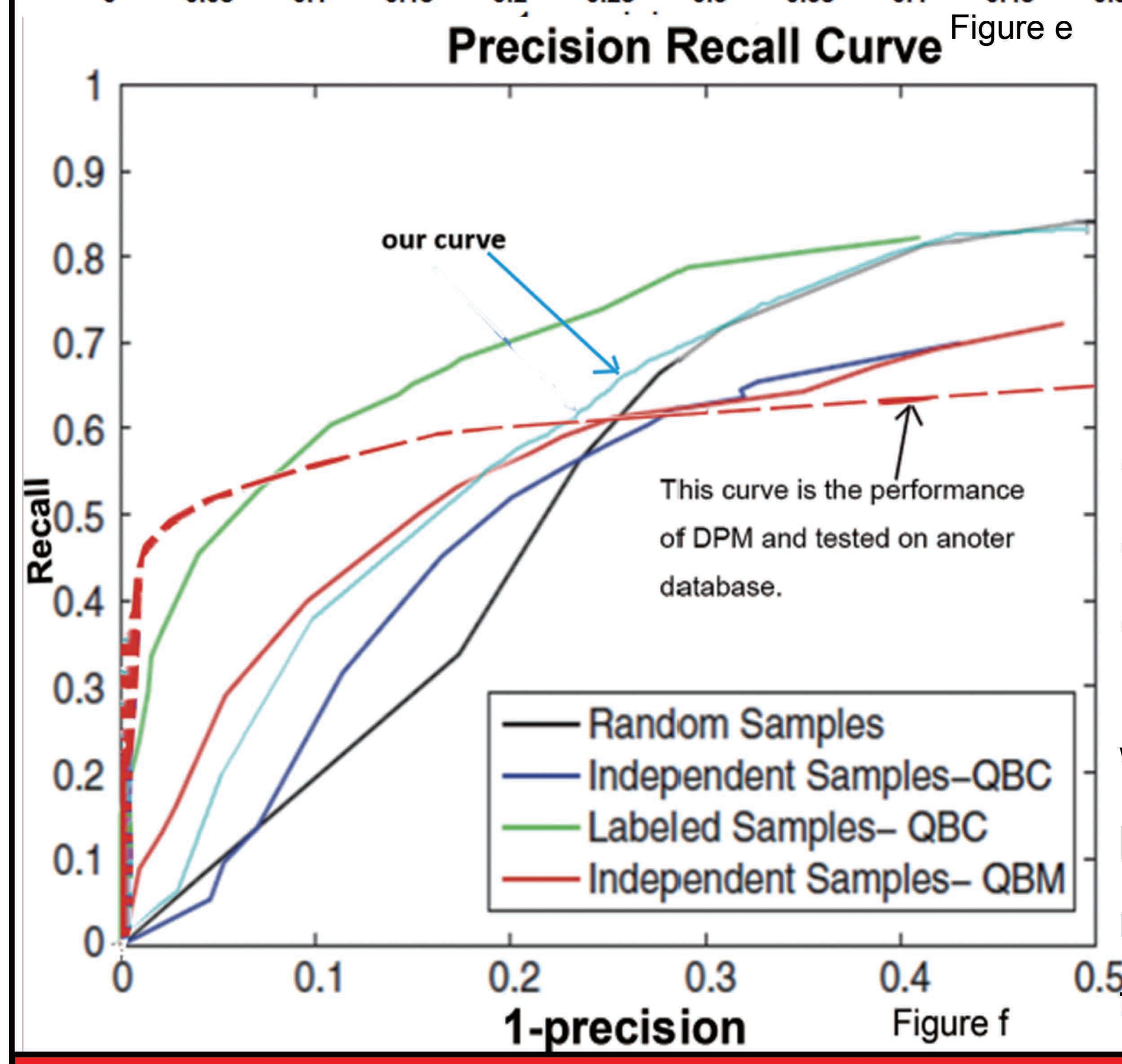


Figure f

To compare with statet-of-the-art haar + adaboost classifiers, in Figure f we put our best curve together with curves from the literature[1] tested on the same database.

QBC, QBM are different strategies for querying samples for retraining, while our classifier did not take advantage of retraining process.

We also include another curve from the leterature [3] using deformable parts models algorithm to train classifier and tested on another database.

Conclusion & Future Work

1. In this project we have implemented and train a detector using OpenCV. The performance is comparable to state-of-the-art according to Figure f.

2. Figure f also shows that by retraining the performance could be further improved. We can make our classifier better by retraining in following work.

3. Literatures [2],[3] show that some other methods which are not implemented in OpenCV have the potential to gain higher performance, eg. ICF[2], DPM[3]. Another computer vision library, libccv (see libccv.org), have implemented them. We will try to prove that.

References

[1].S. Sivaraman and M. M. Trivedi , Active learning for on-road vehicle detection: A comparative study, Mach. Vis. Appl, vol. 16, pp.1 -13 2011.

[2].P. Dollar, Z. Tu, P. Perona, and S. Belongie, 'Integral Channel Features, BMVC 2009.

[3].P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. IEEE Trans. on Pattern Analysis and Machine Intelligence.

Contact: minghanwei19@gmail.com