Aggregated Channel Features Detector Offline Training Guide

ABSTRACT:
This is the offline training User Guide for the Aggregated Channel Features detector.
KEYWORDS:
User Guide
APPROVED:

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
1.0	Sep. 18, 2018	Le Ngoc Linh	Initial version
1.1	Nov. 01, 2018	Dao Anh Xuyen	Update information

Table of Contents

1 Introduction

The Aggregated Channel Features (AggCF) object detector is a fast and effective sliding window detector. It is an evolution of the Viola & Jones (VJ) detector with a ~1000 fold decrease in false positives (at the same detection rate) [1]. The AggCF detection algorithm is provided as a part of an open source Matlab toolbox (hereinafter "AggCF toolbox"), developed by Piotr Dollar et al. APEX-CV Pro has an implementation of this algorithm [2]. This document describes the procedure to use the AggCF toolbox to train the AggCF detector, before deploying the detector coefficients into applications using APEX-CV Pro's AggCFDetector objects.

1.1 Purpose

The APEX-CV AggCFDetector objects perform two tasks: 1) calculate aggregated channels features from input images, 2) apply object detection based on offline pre-trained model which include pre-calculated parameters of decision trees and thresholds. The APEX-CV library uses settings of aggregated channels features generation and model which are optimal in terms of both performance and accuracy for execution on the APEX cores, including octaves/scales number, decision tree number and depth. To achieve better detection accuracy during test/deployment, it is recommended to train the model with dataset of characteristic similar to actual test/deployment scenario. An AggCF training package is provided to facilitate end-user to apply training process on customized dataset, verify the accuracy of trained model and export model for quick deployment in applications using APEX-CV AggCFDetector without having to understand the algorithm and the implementation of the code in details. The purpose of this document is to describe the steps needed for retrain the AggCF detector model using the provided AggCF training package.

1.2 Audience Description

This document is intended for external use by end-users who wish to retrain the AggCF model.

1.3 Document location

This document is located in Vision SDK at: vsdk\s32v234_sdk\docs\apex

AggCF offline training package: vsdk\s32v234_sdk\libs\APEX-CV_pro\aggcf_pd\offline

1.4 References

Id	*Title*	*Location*
1	*Piotr's Image Toolbox document*	*https://pdollar.github.io/toolbox/*
2	*The Apex-CV Pro Library*	*Vsdk/s32v234_sdk/docs/apex/apex_cv/UG-10328-02-12_APEX-CV_Pro_Library.pdf*

Table 1: References

1.5 Definitions, Acronyms and Abbreviations

Term/Acronym	*Description*
AggCF	*Aggregated Channel Features*

Table 2: Acronyms

2 Prerequisites

2.1 Matlab and C++ compiler

AggCF toolbox requires Matlab 2011b or later.

Part of AggCF toolbox is implemented in C++ code. Matlab compiles the code into .mex libraries and call them as Matlab functions. A C++ compiler is required. Currently only Visual C++ and MinGW GCC are tested to compile AggCF toolbox.

MinGW GCC can be installed by using Matlab Add-on Explorer or Download and install:

- Use Matlab Add-on Explorer:
  - On Menu bar: Please click Add-Ons/Get Add-Ons -> An Add-On Explorer window will be opened.
  - Search "MinGW GCC" on Add-On Explorer window and Choose MinGW-w64 in result list.
  - Click Install. Please look the below figure more detail.

Figure 1 Install MinGW-w64 on Add-On Explorer

- Download and install:
  - Download MinGW GCC from http://tdm-gcc.tdragon.net
  - Install to C:\ TDM-GCC-64 (default)

  NOTE: openmp must be enabled during installation:

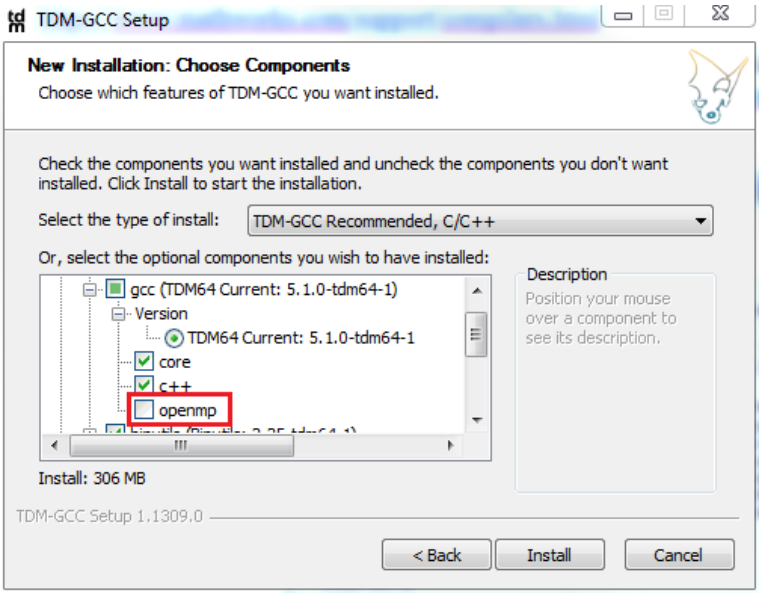

Figure 2 TDM-GCC-64 installation

- Add following environment variables:
  - Variable name: `MW_MINGW64_LOC`
  - Variable value: `C:\TDM-GCC-64`
- In Matlab:
  ```
  >> mex -setup
  ```
  And choose MinGW64 Compiler (C++) from list of detected compilers

The instructions in this document has been verified with the following environment:

- OS: Windows 7/10
- Matlab: R2017a
- C++ compiler: TDM-GCC-64 5.1.0

2.2 Matlab's Image Processing Toolbox

As stated in AggCF documents, AggCF toolbox is meant to be a complement, not a replacement for Matlab's Image Processing Toolbox. In fact, it requires Matlab Image Processing Toolbox to be installed.

2.3 Octave

In case Matlab is not accessible, a developer can use Octave as an open source alternative.

The instructions in this document has been verified with following environment:

- OS: Windows 7
- Octave: 4.2.0
- C++ compiler: x86_64-w64-mingw32 (included with Octave)
- Image Package 2.6.1 (included with octave)

The instructions in this document are applied to both Matlab/Octave environment unless explicitly declared otherwise.

2.4 AggCF Toolbox

The latest version of the toolbox can be found at

https://github.com/pdollar/toolbox

We recommend using version 3.5 which was used for this document.

Installing:

- Unzip the toolbox
- Add all directories to Matlab path:
- `>> addpath(genpath('path/to/toolbox/')); savepath;`
- Copy and extract `aggcf_toolbox_patch.zip` from directory "s32v234_sdk\libs\apexcv_pro\aggcf_pd\offline\train\aggcf_offline_trainer" to AggCF toolbox directory. The patch contains some modification in the training parameters according to the APEX-CV implementation limitations, as well as some modification for Octave compatibility.

Compile C++ libraries:

```
>> aggcfCompile
```

Note:

- Currently, only Matlab is capable of running openmp code with both MinGW64 and Visual C++. There is no support for openmp in Octave as of now and hence training detector in Octave will be significantly slower.
- Matlab 2017 changed mex compile default option to –largeArrayDims. That means compiling AggCF toolbox with default option will result in failure due to large array allocation. Option –compatibleArrayDims needs to be specified to avoid this issue. These checking are handled in aggcfCompile.m.

2.5 Training data

AggCF algorithm needs a large amount of data to train its detector in order to be effective. The training data comprises image data, along with ground truth information about the objects to be detected in each frame, i.e. object location, size, etc.

2.5.1 Caltech Pedestrian Dataset

This dataset is developed by Caltech University and has been used to benchmark the performance of various computer vision algorithms. The dataset consists of approximately 10 hours of 640x480 30Hz video taken from a vehicle driving through regular traffic in an urban environment. About 250,000 frames (in 137 approximately minute long segments) with a total of 350,000 bounding boxes and 2300 unique pedestrians were annotated. The annotation includes temporal correspondence between bounding boxes and detailed occlusion labels.

The full dataset can be downloaded at:

http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/

This address also contains additional datasets developed by other institutions. The main datasets is named data-USA to differ from others.

2.5.1.1 Extracting database

The dataset is provided with Matlab routines for extracting and evaluating. To install these routines, download the code from above address, unzip and add all paths to Matlab path:

```
>> addpath(genpath('path/to/code3.2.1/')); savepath;
```

The data-USA consists of:

- Image data: stored in .seq video format
- Annotations data: ground truth information in .vbb format
- The whole dataset is divided into 11 subsets. Set00-05 for training and set06-10for testing.

Extract dataset:

- Unzip image data and annotations data to appropriate location:
  - `./code3.2.1/data-USA/videos`
  - `./code3.2.1/data-USA/annotations`
- In Matlab:

```
>> cd code3.2.1
>> datadir = '../data/Caltech/' %database destination
>> dbInfo('usatest')
>> dbExtract([datadir 'test'],1,[])
>> dbInfo('usatrain')
>> dbExtract([datadir 'train'],1,4)
```

- Extracted database should contains .jpg image files and .txt annotation files in following structure:

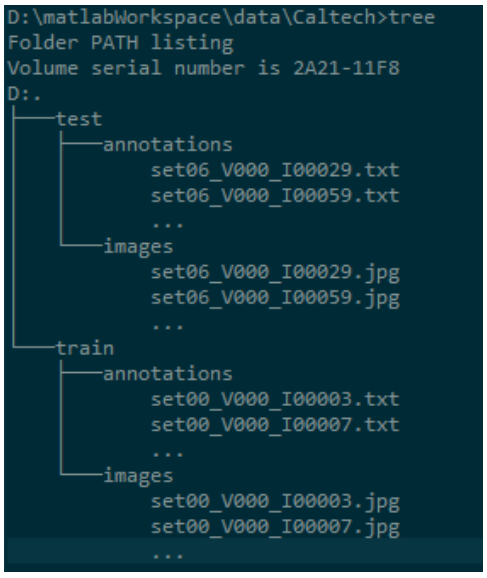

Figure 3 Extracted database

2.5.2Customized dataset

Each image file need to be accompanied with an annotation file with the same name, organized in the same above database structure.

In order to be understood by training scripts, data must be in the same format as of Caltech sets:

- Image data: .jpg, VGA resolution
- Annotation data: text file in following format:

  % bbGt version=3

  person 407 164 12 23 0 0 0 0 0 0 0

  person 435 166 7 15 0 0 0 0 0 0 0

  person 233 119 9 15 1 233 119 9 15 0 0

  Each object has following fields:

  - lbl - a string label describing object type (eg: 'pedestrian')
  - bb - [l t w h]: bb indicating predicted object extent; left/top of bounding box coordination and weight/height
  - occ - 0/1 value indicating if bb is occluded
  - bbv - [l t w h]: bb indicating visible region (can be [0 0 0 0])
  - ign - 0/1 value indicating bb was marked as ignore
  - ang - [0-360] orientation of bb in degrees (not used)

3 Training detector

3.1 Training parameters

APEX-CV implementation has some limitations that need to be reflected in Matlab scripts:

- Input size: 640 x 480 RGB
- Binary trees: 2048 decision trees with variable depth (max depth = 4)
- Octave setting:
    - ‘nPerOct’,4 : number of scales per octave (1 real octave and 3 approximate scales)
    - ‘nOctUp’,0: 0 up sampled scale
- Opts.modelDs=[64 32]; opts.modelDsPad=[64 32] : window size height x width = 64x32

Please see acfDemoCal.m for more information.

3.2 Train the detector

Following figure describe the overall procedure

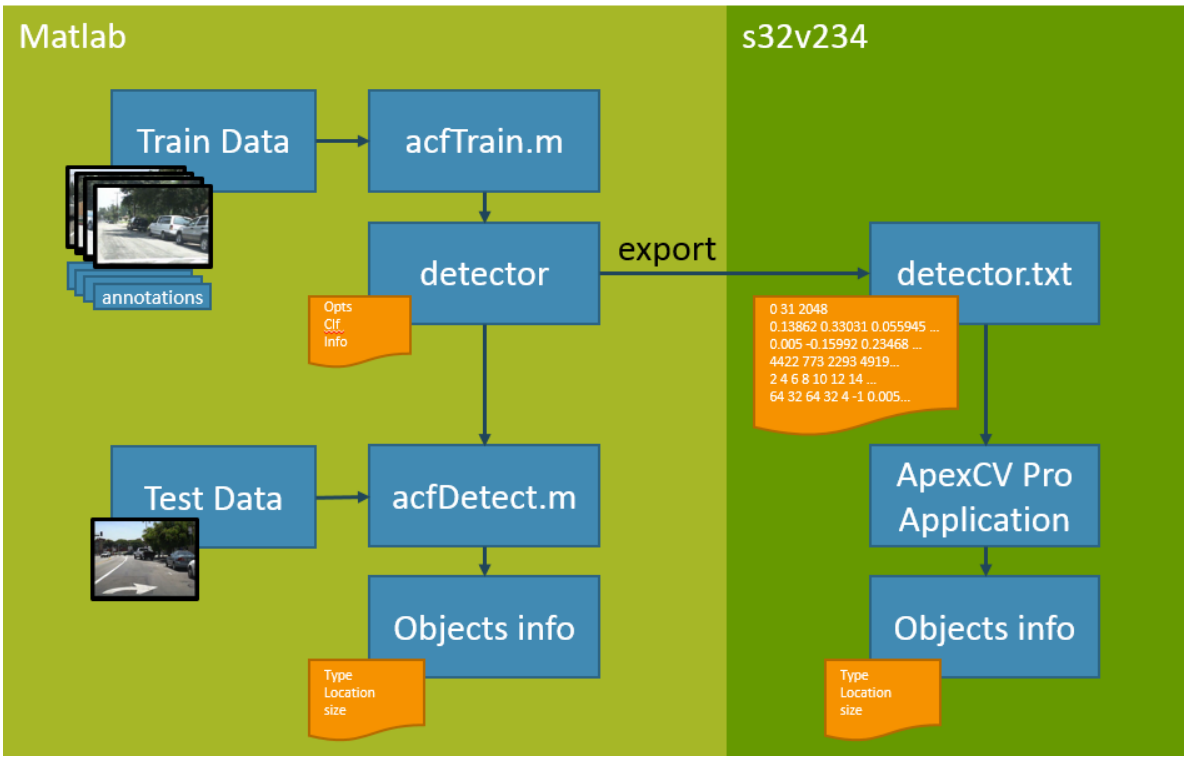

Figure 4 Overall procedure

- If training is performed on Octave, image package needs to be loaded, and turn off pager to display training progress log:

```
>> pkg load image
>> more off
```

- Modify acfDemoCal.m to specify the location of extracted database (.jpg/.txt)

```
dataDir='/path/to/data'
```

- Execute the training scripts:

```
>> acfDemoCal
```

Detector training is a time consuming task, especially when using Octave since the mex libraries in Octave were compiled without openmp option.
After finishing training detector, a ROC curve showing the average miss rate should looks like:

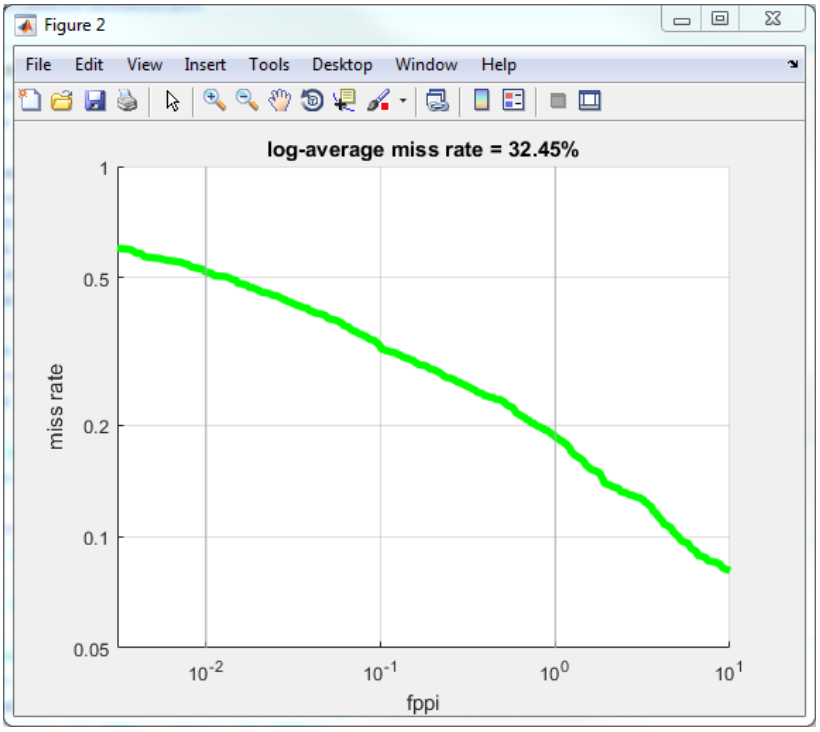

Figure 5 Training log-average miss rate

- Testing the detector:
  After finishing the training process, a user can test the trained detector against a VGA image to verify its functionality. For example:

```
>> I=imread('D:\matlabWorkspace\data\Caltech\test\images\set0
7_V001_I00479.jpg'); bbs=acfDetect(I,detector);
>> figure(2); im(I); bbApply('draw',bbs);
```

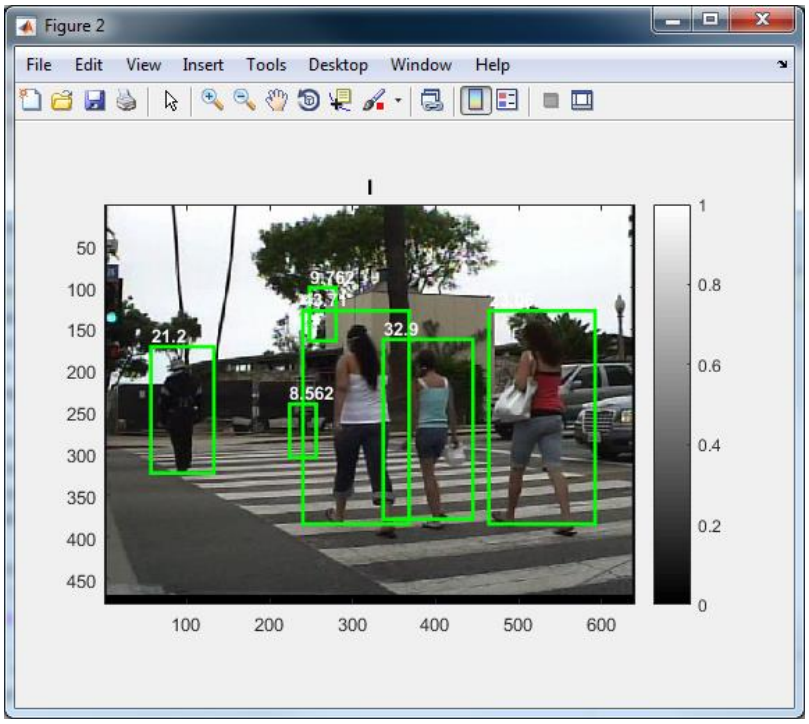

Figure 6 Testing trained detector

- Export detector:

  The final output of training process is a struct named `'detector'`. After detector functionality has been verified on Matlab, the `'detector'` struct can be exported for use into APEX-CV applications.

  ```
  >> convertDetector('detector.txt', detector)
  ```

  For example, the detector trained with current configuration in `aggcf_patch.zip` can be used with APEX-CV Pedestrian Detection AggCF Demo:

- Application under:
  vsdk\s32v234_sdk\demos\apps\pedestrian_detection_aggcf
- Data/config under:
  vsdk\s32v234_sdk\demos\data\apps\pedestrian_detection_aggcf\gdc_pd.ini
  [DATA]
  Filename=detector.txt