Crossroad Detector

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

Note

This library will load data files located in the directory located at `../share/cva/Crossroad↩ Detector/assets` relative to it. For correct operation, if the library is copied outside of the installation directory, the `assets` directory must be copied alongside it.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 cva Namespace Reference

Namespaces

- crd

5.2 cva::crd Namespace Reference

Classes

- class CrossroadDetector
- class DetectedObject

  Class which represents detected object.

Enumerations

- enum ObjectLabel { ObjectLabel::BACKGROUND, ObjectLabel::PERSON, ObjectLabel::VEHICLE, ObjectLabel::NON_VEHICLE }

  An identifier of the category to which an object belongs.

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ac::Version version ()

  Returns the version number of the library.

5.2.1 Enumeration Type Documentation

5.2.1.1 enum cva::crd::ObjectLabel [strong]

An identifier of the category to which an object belongs.

Enumerator

BACKGROUND

PERSON

VEHICLE

NON_VEHICLE

5.2.2 Function Documentation

5.2.2.1 CVA_AC_SHARED_LIBRARY_IMPORT ac::Version cva::crd::version ( )

Returns the version number of the library.

Chapter 6

Class Documentation

6.1 cva::crd::CrossroadDetector Class Reference

`#include <crd.hpp>`

Public Types

- enum { MAX_IMAGE_WIDTH = 4096, MAX_IMAGE_HEIGHT = 4096 }

Public Member Functions

- virtual ~CrossroadDetector ()=default

  Virtual destructor.
- virtual std::vector< std::unique_ptr< DetectedObject > > detect (const ac::ConstImageView &image)=0

  Detects objects in the image, returns vector of detected objects `image` *image where objects should be detected.*

Static Public Member Functions

- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< CrossroadDetector > createCpu ()

  Returns a pointer to a new object implementing ObjectDetectorCommon that uses the CPU for computations. CPU always uses FP32 model.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< CrossroadDetector > createGpu (const cva::ac::Precision precision=cva::ac::Precision::FP32)

  Returns a pointer to a new object implementing ObjectDetectorCommon that uses the GPU for computations. `precision` *model precision. FP32 or FP16 are supported.*

6.1.1 Member Enumeration Documentation

6.1.1.1 anonymous enum

Enumerator

MAX_IMAGE_WIDTH The maximal supported image width.

MAX_IMAGE_HEIGHT The maximal supported image height.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 virtual cva::crd::CrossroadDetector::~CrossroadDetector ( ) [virtual], [default]

Virtual destructor.

6.1.3 Member Function Documentation

6.1.3.1 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<CrossroadDetector> cva::crd::CrossroadDetector::createCpu ( ) [static]

Returns a pointer to a new object implementing ObjectDetectorCommon that uses the CPU for computations. CPU always uses FP32 model.

Examples:

main.cpp.

6.1.3.2 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<CrossroadDetector> cva::crd::CrossroadDetector::createGpu (const cva::ac::Precision *precision* = `cva::ac::Precision::FP32`) [static]

Returns a pointer to a new object implementing ObjectDetectorCommon that uses the GPU for computations. `precision` model precision. FP32 or FP16 are supported.

Examples:

main.cpp.

6.1.3.3 virtual std::vector<std::unique_ptr<DetectedObject> > cva::crd::CrossroadDetector::detect (const ac::ConstImageView & *image*) [pure virtual]

Detects objects in the image, returns vector of detected objects `image` image where objects should be detected.

The documentation for this class was generated from the following file:

- crd.hpp

6.2 cva::crd::DetectedObject Class Reference

Class which represents detected object.

`#include <crd.hpp>`

Public Member Functions

- virtual float confidence ()=0

  returns confidence of detected object in the range [0, 1]
- virtual ac::Rectangle< float > boundingBox ()=0

  returns bounding box of detected object
- virtual ObjectLabel label ()=0

  return label of detected object
- virtual ~DetectedObject ()=default

  virtual destructor

6.2.1 Detailed Description

Class which represents detected object.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 virtual cva::crd::DetectedObject::~DetectedObject ( ) [virtual], [default]

virtual destructor

6.2.3 Member Function Documentation

6.2.3.1 virtual ac::Rectangle<float> cva::crd::DetectedObject::boundingBox ( ) [pure virtual]

returns bounding box of detected object

6.2.3.2 virtual float cva::crd::DetectedObject::confidence ( ) [pure virtual]

returns confidence of detected object in the range [0, 1]

6.2.3.3 virtual ObjectLabel cva::crd::DetectedObject::label ( ) [pure virtual]

return label of detected object

The documentation for this class was generated from the following file:

- crd.hpp

Chapter 7

File Documentation

7.1 crd.hpp File Reference

```
#include <memory>
#include <string>
#include <vector>
#include <cva/ac/api.hpp>
#include <cva/ac/geometry.hpp>
#include <cva/ac/image_view.hpp>
#include <cva/ac/precision.hpp>
```

Classes

- class cva::crd::DetectedObject

  Class which represents detected object.
- class cva::crd::CrossroadDetector

Namespaces

- cva
- cva::crd

Macros

- #define CVA_CRD_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

Enumerations

- enum cva::crd::ObjectLabel { cva::crd::ObjectLabel::BACKGROUND, cva::crd::ObjectLabel::PERSON, cva::crd::ObjectLabel::VEHICLE, cva::crd::ObjectLabel::NON_VEHICLE }

  An identifier of the category to which an object belongs.

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ac::Version cva::crd::version ()

  Returns the version number of the library.

7.1.1 Macro Definition Documentation

7.1.1.1 #define CVA_CRD_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

7.2 example.dox File Reference

7.3 mainpage.dox File Reference

Chapter 8

Example Documentation

8.1 main.cpp

```
/*
    Copyright 2018 Intel Corporation.

    This software and the related documents are Intel copyrighted materials,
    and your use of them is governed by the express license under which they
    were provided to you (Intel Simplified Software License (Version April 2018))
    Unless the License provides otherwise, you may not use, modify,
    copy, publish, distribute, disclose or transmit this software or
    the related documents without Intel's prior written permission.

    This software and the related documents are provided as is, with no
    express or implied warranties, other than those that are expressly
    stated in the License.
*/

#include <cstdlib>
#include <iostream>
#include <numeric>
#include <iomanip>
#include <cmath>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>

#include <cva/ac/geometry.hpp>
#include <cva/ac/ocv/image_view.hpp>
#include <cva/crd/crd.hpp>

int main(int argc, char *argv[])
{
    // Parse the command line arguments using OpenCV.
    cv::CommandLineParser parser(argc, argv,
        "{ @image         |        | image path}"
        "{ help h         |        | print this message }"
        "{ impl           | CPU    | detector implementation to use, possible values are CPU, GPUFP32 or
       GPUFP16 }"
        "{ confidence_thr | 0.5    | confidence value to filter detected objects  }"
        "{ show           | false  | show detected objects }");

    if (parser.has("help"))
    {
        parser.printMessage();
        return EXIT_SUCCESS;
    }

    auto impl = parser.get<cv::String>("impl");
    if (impl != "CPU" && impl != "GPUFP32" && impl != "GPUFP16")
    {
        std::cerr << argv[0] << ": --impl must be either \"CPU\", \"GPUFP32\" or \"GPUFP16\".\n";
        return EXIT_FAILURE;
    }

    if (!parser.check())
    {
        parser.printErrors();
```

```
        return EXIT_FAILURE;
    }

    auto image_path = parser.get<cv::String>("@image");
    cv::Mat image = cv::imread(image_path);
    if (image.empty())
    {
        std::cerr << "failed to read image: " << image_path << std::endl;
        return EXIT_FAILURE;
    }

    auto confidence_thr = parser.get<float>("confidence_thr");
    if (confidence_thr <= 0 || confidence_thr >= 1)
    {
        std::cerr << "confidence threshold value must be greater than 0 and less than 1" << std::endl;
        return EXIT_FAILURE;
    }

    auto detector = impl == "CPU"     ? cva::crd::CrossroadDetector::createCpu
      () :
                    impl == "GPUFP32" ? cva::crd::CrossroadDetector::createGpu
      (cva::ac::Precision::FP32) :
                                        cva::crd::CrossroadDetector::createGpu
      (cva::ac::Precision::FP16);

    auto objects = detector->detect(cva::ac::ocv::toImageView(cva::ac::ImageFormat::BGR_8, image));

    struct LabelProperties {
        const char *description;
        cv::Scalar color;
    } const label_properties[] = {
        {"background",  {0, 255, 0}},
        {"person",      {255, 255, 0}},
        {"vehicle",     {0, 255, 255}},
        {"non-vehicle", {255, 0, 0}},
    };

    objects.erase(std::remove_if(objects.begin(), objects.end(),
                                 [confidence_thr](std::unique_ptr<cva::crd::DetectedObject> const& obj)
                                 {
                                    return obj->confidence() < confidence_thr;
                                 }),
                  objects.end());

    std::cout << "Objects: " << objects.size() << std::endl;

    for (std::size_t i = 0; i < objects.size(); ++i)
    {
        auto label = unsigned(objects[i]->label());
        auto confidence = objects[i]->confidence();
        auto bbox = objects[i]->boundingBox();
        std::cout << i << ": " << confidence << " " << label << " "
                  << label_properties[label].description << " "
                  << bbox.startX() << " " << bbox.startY() << " " << bbox.endX() << " " << bbox.endY()
                  << std::endl;
    }


    auto show = parser.get<bool>("show");
    if (show)
    {
        for (std::size_t i = 0; i < objects.size(); ++i){
            auto bbox = cv::Rect(objects[i]->boundingBox().startX(),
                                 objects[i]->boundingBox().startY(),
                                 objects[i]->boundingBox().endX() - objects[i]->boundingBox().startX(),
                                 objects[i]->boundingBox().endY() - objects[i]->boundingBox().startY());
            auto color = label_properties[unsigned(objects[i]->label())].color;
            cv::rectangle(image, bbox, color, 5, 8, 0);
        }
        cv::imwrite("result.jpeg", image);
    }


    return EXIT_SUCCESS;
}
```