Face Detection

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 cva Namespace Reference

Namespaces

- fd

4.2 cva::fd Namespace Reference

Classes

- class Face

  An interface class for a face.
- class FaceDetector

  The main class which provides functionality of detecting faces on the given sequence of images.
- class Version

  The class is used to represent the version number for the library.

Enumerations

- enum BackendType { BackendType::CPU, BackendType::GPU, BackendType::VPU }

  The type of backend to use.

Functions

- Version version ()

  Return version of the current library.

4.2.1 Enumeration Type Documentation

4.2.1.1 enum cva::fd::BackendType [strong]

The type of backend to use.

Enumerator

CPU Use CPU backend.

GPU Use GPU backend.

VPU Use VPU backend.

4.2.2 Function Documentation

4.2.2.1 Version cva::fd::version ( )

Return version of the current library.

Chapter 5

Class Documentation

5.1 cva::fd::Face Class Reference

An interface class for a face.

`#include <fd.hpp>`

Public Member Functions

- virtual cv::Rect boundingBox () const =0
  Get the bounding box of the face.
- virtual float confidence () const =0
  Get the confidence of the face.
- virtual ~Face ()=default

5.1.1 Detailed Description

An interface class for a face.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 virtual cva::fd::Face::~Face ( ) [virtual], [default]

A virtual destructor for the abstract class.

5.1.3 Member Function Documentation

5.1.3.1 virtual cv::Rect cva::fd::Face::boundingBox ( ) const [pure virtual]

Get the bounding box of the face.

Could be outside of the frame.

Returns

Rectangle that is a bounding box of the face.

5.1.3.2 virtual float cva::fd::Face::confidence ( ) const [pure virtual]

Get the confidence of the face.

Returns

Float confidence value of the face.

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

- fd.hpp

5.2 cva::fd::FaceDetector Class Reference

The main class which provides functionality of detecting faces on the given sequence of images.

`#include <fd.hpp>`

Public Member Functions

- virtual std::vector< std::unique_ptr< Face > > process (const cv::Mat &frame)=0
  Process the frame and detect faces.
- virtual void setConfidence (float detection_confidence=0.)=0
  Set confidence threshold to reject detections with weak score.
- virtual float confidence () const =0
  Get confidence threshold to reject detections with weak score.
- virtual ~FaceDetector ()=default

Static Public Member Functions

- static std::unique_ptr< FaceDetector > create (const std::string &model_path, const std::string &weights_↩ path, BackendType backend_type=BackendType::CPU, const std::string &custom_layers_xml_path="")
  A factory for FaceDetector.

5.2.1 Detailed Description

The main class which provides functionality of detecting faces on the given sequence of images.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 virtual cva::fd::FaceDetector::~FaceDetector ( ) [virtual], [default]

A virtual destructor for the abstract class.

5.2.3 Member Function Documentation

5.2.3.1 virtual float cva::fd::FaceDetector::confidence ( ) const [pure virtual]

Get confidence threshold to reject detections with weak score.

Returns

floating point minimal confidence value for the detected objects.

5.2.3.2 static std::unique_ptr<FaceDetector> cva::fd::FaceDetector::create (const std::string & *model_path*, const std::string & *weights_path*, BackendType *backend_type* = BackendType::CPU, const std::string & *custom_layers_xml_path* = " ") [static]

A factory for FaceDetector.

Parameters

in	*model_path*	The path to the file with the model (xml file for DLDT).
in	*weights_path*	The path to the file with the weights (bin file for DLDT).
in	*backend_type*	The type of DLDT backend to use.
in	*custom_layers_xml_path*	The path to XML file describing DLDT custom layers. Applicable for GPU target only. Should be empty if custom DLDT layers are not used.

Returns

An instance of a face detector implementing FaceDetector interface.

Examples:

main.cpp.

5.2.3.3 virtual std::vector<std::unique_ptr<Face> > cva::fd::FaceDetector::process (const cv::Mat & *frame*) [pure virtual]

Process the frame and detect faces.

The call is blocking. The method does not modify the input data.

Parameters

in	*frame*	the input frame If the frame is an empty cv::Mat, the method returns empty vector.

Returns

Vector of OpenCV smart pointers to Face structures, describing the faces found on the input frame.

5.2.3.4 virtual void cva::fd::FaceDetector::setConfidence (float *detection_confidence* = 0.) [pure virtual]

Set confidence threshold to reject detections with weak score.

Parameters

in	*detection_confidence*	floating point minimal confidence value for the detected objects. (may be negative or positive) In case NaN value the method throws an exception – an instance of cv::Exception with the field code equal to cv::Error::StsBadArg.

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

- fd.hpp

5.3 cva::fd::Version Class Reference

The class is used to represent the version number for the library.

```
#include <fd.hpp>
```

Public Member Functions

- Version (std::uint32_t major=0, std::uint32_t minor=0, std::uint32_t patch=0)
- std::uint32_t major () const

  Get major number of the version.
- std::uint32_t minor () const

  Get minor number of the version.
- std::uint32_t patch () const

  Get patch number of the version.
- std::string toString () const

  Get version string: major.minor.revision.

5.3.1 Detailed Description

The class is used to represent the version number for the library.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 cva::fd::Version::Version (std::uint32_t *major* = 0, std::uint32_t *minor* = 0, std::uint32_t *patch* = 0) [inline], [explicit]

5.3.3 Member Function Documentation

5.3.3.1 std::uint32_t cva::fd::Version::major ( ) const [inline]

Get major number of the version.

5.3.3.2 std::uint32_t cva::fd::Version::minor ( ) const [inline]

Get minor number of the version.

5.3.3.3 std::uint32_t cva::fd::Version::patch ( ) const [inline]

Get patch number of the version.

5.3.3.4 std::string cva::fd::Version::toString ( ) const [inline]

Get version string: major.minor.revision.

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

- fd.hpp

Chapter 6

File Documentation

6.1 example.dox File Reference

6.2 fd.hpp File Reference

```
#include <opencv2/core/core.hpp>
#include <cstdint>
#include <memory>
#include <string>
#include <vector>
```

Classes

- class cva::fd::Version

  The class is used to represent the version number for the library.
- class cva::fd::Face

  An interface class for a face.
- class cva::fd::FaceDetector

  The main class which provides functionality of detecting faces on the given sequence of images.

Namespaces

- cva
- cva::fd

Macros

- #define CVA_FD_DLLEXPORT

Enumerations

- enum cva::fd::BackendType { cva::fd::BackendType::CPU, cva::fd::BackendType::GPU, cva::fd::Backend↩ Type::VPU }

  The type of backend to use.

Functions

- Version cva::fd::version ()

  Return version of the current library.

6.2.1 Macro Definition Documentation

6.2.1.1 #define CVA_FD_DLLEXPORT

Chapter 7

Example Documentation

7.1 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 "cva/fd/fd.hpp"
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/core/utility.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <type_traits>
#include <utility>

using namespace std;

int main(int argc, char* argv[])
{
    const cv::String callOptions = "{help h usage ? |       | print this message                }"
                                   "{image i        |       | path to the image to find faces on }"
                                   "{model m        |       | path to the model file            }"
                                   "{weights w      |       | path to the weights file          }"
                                   "{show-gui s     | false | whether to show gui               }"
                                   ;

    cv::CommandLineParser argumentParser(argc, argv, callOptions);
    argumentParser.about("Face detection demo application");

    if (argumentParser.has("help"))
    {
        argumentParser.printMessage();
        return 0;
    }

    string image_path = argumentParser.get<string>("image");
    cout << "Reading image '" << image_path << "'" << endl;
    cv::Mat image = cv::imread(image_path);
    if (image.empty())
    {
        cout << "Cannot read the image" << endl;
        exit(1);
    }

    std::string modelPath;
    std::string weightsPath;
    if (! argumentParser.has("model"))
```

```
    {
        std::cout << "Error: command line parameter '--model' is required" << std::endl;
        exit(1);
    }
    modelPath = argumentParser.get<string>("model");

    if (! argumentParser.has("weights"))
    {
        std::cout << "Error: command line parameter '--weights' is required" << std::endl;
        exit(1);
    }
    weightsPath = argumentParser.get<string>("weights");

    auto detector = cva::fd::FaceDetector::create(modelPath, weightsPath);
    if (!detector)
    {
        cout << "Cannot create detector" << endl;
        exit(1);
    }

    auto result_faces = detector->process(image);

    cv::Scalar green_color(0, 255, 0);
    for (const auto& face : result_faces)
    {
        cv::Rect r = face->boundingBox();
        cv::rectangle(image, r, green_color);
    }
    if (argumentParser.get<bool>("show-gui"))
    {
        cv::imshow("Face Detection Example", image);
        char key = 0;
        const char ESC_KEY = 27;
        while (key != ESC_KEY)
        {
            key = cv::waitKey();
        }
    }
    else
    {
        cv::imwrite("face_detection_example.png", image);
    }

    return 0;
}
```