License Plate Recognition

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

- lpr

4.2 cva::lpr Namespace Reference

Classes

- class LicensePlateDecoder

  Abstract class for DL-based license plate decoder.
- class Version

  A version of the library.

Enumerations

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

  Possible hardware backends that can be used for computation.
- enum WaitResult { WaitResult::COMPLETED, WaitResult::TIMED_OUT }

  Result of an asynchronous wait function.

Functions

- Version version ()

  Returns the version of the library.

4.2.1 Enumeration Type Documentation

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

Possible hardware backends that can be used for computation.

Enumerator

CPU

GPU

4.2.1.2 enum cva::lpr::WaitResult [strong]

Result of an asynchronous wait function.

Enumerator

COMPLETED

TIMED_OUT

4.2.2 Function Documentation

4.2.2.1 Version cva::lpr::version ( )

Returns the version of the library.

Chapter 5

Class Documentation

5.1 cva::lpr::LicensePlateDecoder Class Reference

Abstract class for DL-based license plate decoder.

`#include <lpr.hpp>`

Public Member Functions

- virtual std::string decode (const cv::Mat &plate)=0

  Decode the image containing only license plate.
- virtual void createAsyncRequests (size_t nrequests)=0

  Creates defined number of asynchronous requests.
- virtual void startAsyncDecode (const cv::Mat &plate)=0

  Starts the image decoding asynchronously.
- virtual WaitResult waitAsync (int64_t millis_timeout)=0

  Waits for decoding to end. Blocks until specified millis_timeout has elapsed or the decoding ends, whichever comes first.
- virtual std::string fetchAsyncDecodeResult ()=0

  Gets decoding result.
- virtual ~LicensePlateDecoder ()=default

  A virtual destructor for the abstract class.

Static Public Member Functions

- static std::unique_ptr< LicensePlateDecoder > create (const std::string &model_file, const std::string &weights_file, const std::string &dict_file, BackendType backend_type=BackendType::CPU)

  Create an instance of the license plate decoder.

5.1.1 Detailed Description

Abstract class for DL-based license plate decoder.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 virtual cva::lpr::LicensePlateDecoder::~LicensePlateDecoder ( ) [virtual], [default]

A virtual destructor for the abstract class.

5.1.3 Member Function Documentation

5.1.3.1 static std::unique_ptr<LicensePlateDecoder> cva::lpr::LicensePlateDecoder::create (const std::string & *model_file*, const std::string & *weights_file*, const std::string & *dict_file*, BackendType *backend_type* = BackendType::CPU) [static]

Create an instance of the license plate decoder.

Parameters

in	*model_file*	File with network topology in XML format.
in	*weights_file*	Binary file with network weights.
in	*dict_file*	Dictionary containing outputs of network mapping to alphanumeric characters. Each line of this file must contain class ID and the corresponding character separated by space.
in	*backend_type*	Device to compute classification on

Returns

Pointer to new object.

Examples:

main.cpp.

5.1.3.2 virtual void cva::lpr::LicensePlateDecoder::createAsyncRequests (size_t *nrequests*) [pure virtual]

Creates defined number of asynchronous requests.

Parameters

in	*nrequests*	number of supported asynchronous requests

5.1.3.3 virtual std::string cva::lpr::LicensePlateDecoder::decode (const cv::Mat & *plate*) [pure virtual]

Decode the image containing only license plate.

Parameters

in	*plate*	Image to be processed. It should be converted to appropriate color space before calling this function. However, image is rescaled inside according to network it uses.

Returns

Label, i.e. string containing Chinese province and alphanumeric string, e.g. <Beijing>FA9152 or WJ<Jiangsu>02009.

Examples:

main.cpp.

5.1.3.4 virtual std::string cva::lpr::LicensePlateDecoder::fetchAsyncDecodeResult ( ) [pure virtual]

Gets decoding result.

Returns

Label, i.e. string containing Chinese province and alphanumeric string, e.g. <Beijing>FA9152 or WJ<Jiangsu>02009.

5.1.3.5 virtual void cva::lpr::LicensePlateDecoder::startAsyncDecode (const cv::Mat & *plate*) [pure virtual]

Starts the image decoding asynchronously.

Parameters

in	*plate*	Image to be processed. It should be converted to appropriate color space before calling this function. However, image is rescaled inside according to network it uses.

5.1.3.6 virtual WaitResult cva::lpr::LicensePlateDecoder::waitAsync (int64_t *millis_timeout*) [pure virtual]

Waits for decoding to end. Blocks until specified millis_timeout has elapsed or the decoding ends, whichever comes first.

Parameters

in	*millis_timeout*	Maximum duration in milliseconds to block for, or -1 for an unlimited duration.

Returns

WaitResult::COMPLETED if the decoding has ended, WaitResult::TIMED_OUT otherwise.

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

- lpr.hpp

5.2 cva::lpr::Version Class Reference

A version of the library.

`#include <lpr.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.

5.2.1 Detailed Description

A version of the library.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 cva::lpr::Version::Version (std::uint32_t *major* = `0`, std::uint32_t *minor* = `0`, std::uint32_t *patch* = `0`) `[inline]`, `[explicit]`

5.2.3 Member Function Documentation

5.2.3.1 std::uint32_t cva::lpr::Version::major ( ) const `[inline]`

Get major number of the version.

5.2.3.2 std::uint32_t cva::lpr::Version::minor ( ) const `[inline]`

Get minor number of the version.

5.2.3.3 std::uint32_t cva::lpr::Version::patch ( ) const `[inline]`

Get patch number of the version.

5.2.3.4 std::string cva::lpr::Version::toString ( ) const `[inline]`

Get version string: major.minor.revision.

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

- lpr.hpp

Chapter 6

File Documentation

6.1 example.dox File Reference

6.2 lpr.hpp File Reference

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

Classes

- class cva::lpr::Version

  A version of the library.
- class cva::lpr::LicensePlateDecoder

  Abstract class for DL-based license plate decoder.

Namespaces

- cva
- cva::lpr

Macros

- #define CVA_LPR_EXPORT

Enumerations

- enum cva::lpr::BackendType { cva::lpr::BackendType::CPU, cva::lpr::BackendType::GPU }

  Possible hardware backends that can be used for computation.
- enum cva::lpr::WaitResult { cva::lpr::WaitResult::COMPLETED, cva::lpr::WaitResult::TIMED_OUT }

  Result of an asynchronous wait function.

Functions

- Version cva::lpr::version ()

  Returns the version of the library.

6.2.1 Macro Definition Documentation

6.2.1.1 #define CVA_LPR_EXPORT

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

#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <memory>

#include <cva/lpr/lpr.hpp>


int main(int argc, char** argv)
{
    const std::string keys =
            "{h help usage ?   |      | print this message       }"
            "{i image          |      | path to image            }"
            "{w weights        |      | weights file             }"
            "{m model          |      | model file               }"
            "{d dict           |      | dictionary               }"
            "{b backend        | CPU | CPU or GPU                }";

    cv::CommandLineParser parser(argc, argv, keys.c_str());
    bool need_help = parser.get<bool>("help");
    if (need_help)
    {
        parser.printMessage();
        return 0;
    }

    std::string path_to_image = parser.get<std::string>("image");
    if (path_to_image.empty())
    {
        std::cout << "ERROR: Specify image!" << std::endl;
        parser.printMessage();
        return -1;
    }

    cv::Mat input_image;
    try
```

```
    {
        input_image = cv::imread(path_to_image);
        cv::cvtColor(input_image, input_image, cv::COLOR_BGR2RGB);

    }
    catch (cv::Exception&)
    {
        std::cout << "Could not read the image: "<< path_to_image << std::endl;
        return -1;
    }

    std::string weights_file = parser.get<std::string>("weights");
    if (weights_file.empty())
    {
        std::cout << "Specify path to weights file!" << std::endl;
        parser.printMessage();
        return -1;
    }

    std::string model_file =  parser.get<std::string>("model");
    if (model_file.empty())
    {
        std::cout << "Specify path to model file" << std::endl;
        parser.printMessage();
        return -1;
    }

    std::string dict_file = parser.get<std::string>("dict");
    if (dict_file.empty())
    {
        std::cout << "Specify path to dictionary file" << std::endl;
        parser.printMessage();
        return -1;
    }

    std::string backend = parser.get<std::string>("backend");
    if ("CPU" != backend && "GPU" != backend)
    {
        std::cout << "Possible options for backend are CPU or GPU" << std::endl;
        parser.printMessage();
        return -1;
    }

    std::unique_ptr<cva::lpr::LicensePlateDecoder> decoder;
    try
    {
        decoder = cva::lpr::LicensePlateDecoder::create(
            model_file, weights_file, dict_file, "CPU" == backend ?
      cva::lpr::BackendType::CPU :
      cva::lpr::BackendType::GPU);
    }
    catch (std::exception& e)
    {
        std::cout << "Exception caught: Could not create decoder (" << e.what() << ")" << std::endl;
        return -1;
    }

    std::string label;
    try
    {
        label = decoder->decode(input_image);
    }
    catch (std::exception& e)
    {
        std::cout << "Exception caught: Could not decode license plate (" << e.what() << ")" << std::endl;
        return -1;
    }

    std::cout << "Decoded plate: " << label << std::endl;
    return 0;
}
```