Person Reidentification

Generated by Doxygen 1.8.11

Contents

1 Main Page **1**

2 Namespace Index **3**

2.1 Namespace List 3

3 Class Index **5**

3.1 Class List 5

4 File Index **7**

4.1 File List 7

5 Namespace Documentation **9**

5.1 cva Namespace Reference 9

5.2 cva::reid Namespace Reference 9

5.2.1 Function Documentation 9

5.2.1.1 version() 9

6 Class Documentation **11**

6.1 cva::reid::PersonReidentifier Class Reference 11

6.1.1 Detailed Description 12

6.1.2 Member Enumeration Documentation 12

6.1.2.1 anonymous enum 12

6.1.2.2 anonymous enum 12

6.1.3 Constructor & Destructor Documentation 12

6.1.3.1 ∼PersonReidentifier()=default 12

6.1.4 Member Function Documentation 12

6.1.4.1 calcEmbd(const ac::ConstImageView &image, ac::Span< float > embeddings)=0 12

6.1.4.2 calcEmbdBatch(ac::Span< const ac::ConstImageView > images, ac::Span< const ac::Span< float >> embd_buffers)=0 13

6.1.4.3 createCpu(const std::size_t batch_size=8) 13

6.1.4.4 createGpu(const ac::Precision precision=ac::Precision::FP32, const std::size_↩ t batch_size=8) 14

6.1.4.5 embdSize() 14

7 File Documentation **15**

7.1 example.dox File Reference 15

7.2 mainpage.dox File Reference 15

7.3 reid.hpp File Reference 15

7.3.1 Macro Definition Documentation 15

7.3.1.1 CVA_REID_EXPORT 15

8 Example Documentation **17**

8.1 main.cpp 17

Chapter 1

Main Page

Note

This library will load data files located in the directory located at `../share/cva/Person↩ Reidentification/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:

cva 9
cva::reid 9

Chapter 3

Class Index

3.1 Class List

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

cva::reid::PersonReidentifier
Interface to implementation of person reidentifier 11

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

reid.hpp 15

Chapter 5

Namespace Documentation

5.1 cva Namespace Reference

Namespaces

- reid

5.2 cva::reid Namespace Reference

Classes

- class PersonReidentifier

  Interface to implementation of person reidentifier.

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ac::Version version ()

  Returns version number of the library.

5.2.1 Function Documentation

5.2.1.1 CVA_AC_SHARED_LIBRARY_IMPORT ac::Version cva::reid::version ( )

Returns version number of the library.

Chapter 6

Class Documentation

6.1 cva::reid::PersonReidentifier Class Reference

Interface to implementation of person reidentifier.

`#include <reid.hpp>`

Public Types

- enum { MAX_IMAGE_WIDTH = 4096, MAX_IMAGE_HEIGHT = 4096 }
- enum { MAX_BATCH_SIZE = 16 }

Public Member Functions

- virtual ~PersonReidentifier ()=default
- virtual void calcEmbd (const ac::ConstImageView &image, ac::Span< float > embeddings)=0

  Calculate embedding `for` an image.
- virtual void calcEmbdBatch (ac::Span< const ac::ConstImageView > images, ac::Span< const ac::Span< float >> embd_buffers)=0

  Calculate embeddings for a batch of `images`, filling the corresponding elements of `embd_buffers`.

Static Public Member Functions

- static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t embdSize ()

  Returns the size of embedding the reidentifier produces.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< PersonReidentifier > createCpu (const std::size_t batch_size=8)

  Returns a pointer to a new object implementing PersonReidentifier that uses the CPU for computations.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< PersonReidentifier > createGpu (const ac↩ ::Precision precision=ac::Precision::FP32, const std::size_t batch_size=8)

  Returns a pointer to a new object implementing PersonReidentifier that uses the GPU for computations.

6.1.1 Detailed Description

Interface to implementation of person reidentifier.

A PersonReidentifier object allows to calculate for a pedestrian image an "embedding" – a float vector, characterizing the pedestrian appearance.

If there is a pedestrian on a frame received from some surveillance camera, and there is a pedestrian on another frame received from the same (or even from another) camera, the patches of the frames containing the pedestrians may be passed to a PersonReidentifier object to calculate the embeddings.

Then a float value characterizing similarity of appearance of the pedestrians may be calculated as follows:

- normalize the both embedding vectors in the Euclidean space
- get the dot product of the normalized vectors

Applying a threshold to this similarity value may give a decision if these pedestrians are the same person (shot from two points of view or in two different time moments) or different persons.

Note that the PersonReidentifier object has no mutable state. That is, it will always calculate the same embedding if it is given the same image.

6.1.2 Member Enumeration Documentation

6.1.2.1 anonymous enum

Enumerator

MAX_IMAGE_WIDTH	The maximal supported image width.
MAX_IMAGE_HEIGHT	The maximal supported image height.

6.1.2.2 anonymous enum

Enumerator

MAX_BATCH_SIZE	The maximal supported batch size.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 virtual cva::reid::PersonReidentifier::∼PersonReidentifier ( ) `[virtual]`, `[default]`

6.1.4 Member Function Documentation

6.1.4.1 virtual void cva::reid::PersonReidentifier::calcEmbd (const ac::ConstImageView & *image*, ac::Span< float > *embeddings*) `[pure virtual]`

Calculate embedding `for` an image.

Parameters

`in`	*image*	input image
`out`	*embeddings*	output calculated embeddings

Precondition

`image.format()` is ImageFormat::RGB_8 or ImageFormat::BGR_8
`image.width()` <= MAX_IMAGE_WIDTH
`image.height()` <= MAX_IMAGE_HEIGHT
`embeddings.size()` == `embdSize()`
Please note, that `embeddings` should be pre-allocated first, before passing to function.

This method must not be invoked on the same object from more than one thread at a time.

Examples:

main.cpp.

6.1.4.2 virtual void cva::reid::PersonReidentifier::calcEmbdBatch (ac::Span< const ac::ConstImageView > *images*, ac::Span< const ac::Span< float >> *embd_buffers*) `[pure virtual]`

Calculate embeddings for a batch of `images`, filling the corresponding elements of `embd_buffers`.

Equivalent to a sequence of calcEmbd() calls on each pair of corresponding elements from `images` and `embeddings` buffers, but may execute faster. This call will result in multiple infer calls if `images.size()` is bigger than `batch_size`.

Parameters

`in`	*images*	input images
`out`	*embd_buffers*	output calculated embedding buffers

Precondition

`embd_buffers.size()` == `images.size()`
Please note, that `embd_buffers` should be pre-allocated first, before passing to function.
Each element of `images` satisfies the preconditions on `image` in calcEmbd().
Each element of `embd_buffers` satisfy the preconditions on `embeddings` in calcEmbd().

This method must not be invoked on the same object from more than one thread at a time.

6.1.4.3 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<PersonReidentifier> cva::reid::PersonReidentifier::createCpu (const std::size_t *batch_size* = 8) `[static]`

Returns a pointer to a new object implementing PersonReidentifier that uses the CPU for computations.

`batch_size` is the maximum number of images that calcEmbdBatch() will be able to handle. Increasing of this parameter will also increase the amount of memory used by the object.

Precondition

1 <= `batch_size` <= MAX_BATCH_SIZE

Examples:

main.cpp.

6.1.4.4 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<PersonReidentifier> cva::reid::PersonReidentifier↩ ::createGpu (const ac::Precision *precision* = `ac::Precision::FP32`, const std::size_t *batch_size* = `8`) `[static]`

Returns a pointer to a new object implementing PersonReidentifier that uses the GPU for computations.

`precision` depicts precision of the model.

Precondition

`precision` is ac::Precision::FP32 or ac::Precision::FP16.

See createCpu() for the meaning of and preconditions on `batch_size`.

Examples:

main.cpp.

6.1.4.5 static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t cva::reid::PersonReidentifier::embdSize ( ) `[static]`

Returns the size of embedding the reidentifier produces.

Will not throw runtime exceptions.

Examples:

main.cpp.

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

- reid.hpp

Chapter 7

File Documentation

7.1 example.dox File Reference

7.2 mainpage.dox File Reference

7.3 reid.hpp File Reference

```
#include <cva/ac/api.hpp>
#include <cva/ac/image_view.hpp>
#include <cva/ac/span.hpp>
#include <cva/ac/precision.hpp>
#include <cstdint>
#include <cstdlib>
#include <memory>
```

Classes

- class cva::reid::PersonReidentifier

  Interface to implementation of person reidentifier.

Namespaces

- cva
- cva::reid

Macros

- #define CVA_REID_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

Functions

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

  Returns version number of the library.

7.3.1 Macro Definition Documentation

7.3.1.1 #define CVA_REID_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

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 <algorithm>
#include <cstdlib>
#include <exception>
#include <iomanip>
#include <iostream>
#include <numeric>
#include <utility>
#include <vector>

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

#include <cva/ac/ocv/image_view.hpp>
#include <cva/reid/reid.hpp>


namespace ac = cva::ac;

int main(int argc, char *argv[])
try
{
    // Parse the command line arguments using OpenCV.

    cv::CommandLineParser parser(argc, argv,
        "{ @image |        | image }"
        "{ help h |        | print this message }"
        "{ impl   | CPU    | implementation to use. Possible values are CPU, GPUFP32, GPUFP16 }");

    if (!parser.check())
    {
        parser.printErrors();
        return EXIT_FAILURE;
    }

    if (parser.has("help"))
    {
        parser.printMessage();
        return EXIT_SUCCESS;
    }
    if (!parser.has("@image"))
    {
        std::cerr << argv[0] << ": image parameter missing\n";
        return EXIT_FAILURE;
    }

    auto image_path = parser.get<cv::String>("@image");

    // Read the input image and verify it was correctly loaded and its dimensions
    // are suitable.

    cv::Mat image = cv::imread(image_path);
    if (!image.data)
    {
        std::cerr << argv[0] << ": couldn't load image \"" << image_path << "\"\n";
        return EXIT_FAILURE;
    }

    if (image.cols > cva::reid::PersonReidentifier::MAX_IMAGE_WIDTH
       ||
            image.rows > cva::reid::PersonReidentifier::MAX_IMAGE_HEIGHT
      )
    {
        std::cerr << argv[0] << ": image size (" << image.cols << "x" << image.rows << ") "
            << "is greater than what the net supports (<= " <<
      cva::reid::PersonReidentifier::MAX_IMAGE_WIDTH
            << "x" << cva::reid::PersonReidentifier::MAX_IMAGE_HEIGHT
       << ")\n";
        return EXIT_FAILURE;
    }

    std::vector<float> embds(cva::reid::PersonReidentifier::embdSize
      ());

    // Create net and perform embedding calculation. Note that OpenCV
    // loads images in the BGR format.

    std::unique_ptr<cva::reid::PersonReidentifier> net;
    auto impl = parser.get<cv::String>("impl");
    if (impl == "CPU")
    {
        net = cva::reid::PersonReidentifier::createCpu(1);
    }
    else if (impl == "GPUFP32")
    {
        net = cva::reid::PersonReidentifier::createGpu(
      cva::ac::Precision::FP32, 1);
    }
    else if (impl == "GPUFP16")
    {
        net = cva::reid::PersonReidentifier::createGpu(
      cva::ac::Precision::FP16, 1);
    }
    else
    {
        std::cerr << argv[0] << ": --impl must be either \"CPU\" or \"GPUFP32\" or \"GPUFP16\"\n";
        return EXIT_FAILURE;
    }

    net->calcEmbd(ac::ocv::toImageView(ac::ImageFormat::BGR_8, image),
                              ac::toSpan(embds));
    std::cout << "Calculated embedding (of size: " <<
      cva::reid::PersonReidentifier::embdSize() <<  ") :" << std::endl;
    std::cout << std::endl;
    for (size_t i=0; i < cva::reid::PersonReidentifier::embdSize();
      i++)
        std::cout << "  [" << i <<"] = " << std::fixed << std::setprecision(3) << embds[i] << std::endl;
    std::cout << std::endl;

    return EXIT_SUCCESS;
}
catch (std::exception &e)
{
    std::cerr << argv[0] << ": " << e.what() << "\n";
    return EXIT_FAILURE;
}
catch (...)
{
    std::cerr << argv[0] << ": " << "unidentified error\n";
    return EXIT_FAILURE;
}
```