Vehicle Attributes Classification

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/Vehicle↩AttributesClassification/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

- vc

5.2 cva::vc Namespace Reference

Classes

- class VehicleClassifier

  Class which represents vehicle classifier that can estimate, for an image depicting an vehicle, type of the vehicle and its color.

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ac::Version version ()

  Returns the version number of the library.

5.2.1 Function Documentation

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

Returns the version number of the library.

Chapter 6

Class Documentation

6.1 cva::vc::VehicleClassifier Class Reference

Class which represents vehicle classifier that can estimate, for an image depicting an vehicle, type of the vehicle and its color.

`#include <vc.hpp>`

Public Types

- enum { MAX_IMAGE_WIDTH = 4096, MAX_IMAGE_HEIGHT = 4096 }

Public Member Functions

- virtual ~VehicleClassifier ()=default

  Virtual destructor.
- virtual void classify (const ac::Span< const ac::ConstImageView > images, const ac::Span< const ac::↩ Span< float >> colors_probabilities, const ac::Span< const ac::Span< float >> types_probabilities)=0

  Classify `images`, filling `colors_probabilities` and `types_probabilities`.

Static Public Member Functions

- static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t numTypes ()

  Returns the number of vehicle types the classifier knows.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t numColors ()

  Returns the number of colors the classifier knows.
- static CVA_AC_SHARED_LIBRARY_IMPORT const char * typeDescription (std::size_t class_index)

  Returns a text description for vehicle type with index `class_index`.
- static CVA_AC_SHARED_LIBRARY_IMPORT const char * colorDescription (std::size_t class_index)

  Returns a text description for vehicle color with index `class_index`.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< VehicleClassifier > createCpu (const std↩ ::size_t batch_size=1)

  Returns a pointer to a new object implementing VehicleClassifier that uses the CPU for computations.
- static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr< VehicleClassifier > createGpu (const ac::↩ Precision precision=ac::Precision::FP32, const std::size_t batch_size=1)

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

6.1.1 Detailed Description

Class which represents vehicle classifier that can estimate, for an image depicting an vehicle, type of the vehicle and its color.

The supported types include:

- `bus`
- `car`
- `truck`
- `van`

The supported colors include:

- `black`
- `blue`
- `gray`
- `green`
- `red`
- `white`
- `yellow`

typeDescription() and colorDescription() can be used to determine the index of each supported type or color, respectively.

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.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 virtual cva::vc::VehicleClassifier::~VehicleClassifier ( ) [virtual], [default]

Virtual destructor.

6.1.4 Member Function Documentation

6.1.4.1 virtual void cva::vc::VehicleClassifier::classify (const ac::Span< const ac::ConstImageView > *images*, const ac::Span< const ac::Span< float >> *colors_probabilities*, const ac::Span< const ac::Span< float >> *types_probabilities*) [pure virtual]

Classify `images`, filling `colors_probabilities` and `types_probabilities`.

Precondition

`images.size()` == `colors_probabilities.size()`
`images.size()` == `types_probabilities.size()`
For every `b`, `images[b].format()` is ImageFormat::RGB_8 or ImageFormat::BGR_8
For every `b`, `images[b].width()` <= MAX_IMAGE_WIDTH
For every `b`, `images[b].height()` <= MAX_IMAGE_HEIGHT
For every `b`, `color_probabilities[b].size()` == numColors()
For every `b`, `type_probabilities[b].size()` == numTypes()

Postcondition

For every `b` and `i`, `color_probabilities[b][i]` is the probability that the vehicle `images[b]` depicts has color with index `i`.
For every `b` and `i`, `type_probabilities[b][i]` is the probability that the vehicle `images[b]` depicts has type with index `i`.

6.1.4.2 static CVA_AC_SHARED_LIBRARY_IMPORT const char* cva::vc::VehicleClassifier::colorDescription (std::size_t *class_index*) [static]

Returns a text description for vehicle color with index `class_index`.

Precondition

`class_index` < numColors()

The return value is a pointer to a NUL-terminated non-empty string of printable ASCII characters. The string will remain valid for the entire lifetime of the library and must not be freed.

Examples:

main.cpp.

6.1.4.3 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<VehicleClassifier> cva::vc::VehicleClassifier::createCpu (const std::size_t *batch_size* = `1`) [static]

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

`batch_size` is the maximum number of images that classify() will be able to handle at once. If size of input for classify() is bigger then batch_size, there will be more than one inference call. Increasing this parameter will also increase the amount of memory used by the object.

Examples:

main.cpp.

6.1.4.4 static CVA_AC_SHARED_LIBRARY_IMPORT std::unique_ptr<VehicleClassifier> cva::vc::VehicleClassifier::createGpu (const ac::Precision *precision* = ac::Precision::FP32, const std::size_t *batch_size* = 1) [static]

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

Precondition

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

`precision` selects precision of the model.

`batch_size` is the maximum number of images that classify() will be able to handle at once. If size of input for classify() is bigger then `batch_size`, there will be more than one inference call. Increasing this parameter will also increase the amount of memory used by the object.

Examples:

main.cpp.

6.1.4.5 static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t cva::vc::VehicleClassifier::numColors ( ) [static]

Returns the number of colors the classifier knows.

Examples:

main.cpp.

6.1.4.6 static CVA_AC_SHARED_LIBRARY_IMPORT std::size_t cva::vc::VehicleClassifier::numTypes ( ) [static]

Returns the number of vehicle types the classifier knows.

Examples:

main.cpp.

6.1.4.7 static CVA_AC_SHARED_LIBRARY_IMPORT const char* cva::vc::VehicleClassifier::typeDescription (std::size_t *class_index*) [static]

Returns a text description for vehicle type with index `class_index`.

Precondition

`class_index` < numTypes()

The return value is a pointer to a NUL-terminated non-empty string of printable ASCII characters. The string will remain valid for the entire lifetime of the library and must not be freed.

Examples:

main.cpp.

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

- vc.hpp

Chapter 7

File Documentation

7.1 example.dox File Reference

7.2 mainpage.dox File Reference

7.3 vc.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::vc::VehicleClassifier

  Class which represents vehicle classifier that can estimate, for an image depicting an vehicle, type of the vehicle and its color.

Namespaces

- cva
- cva::vc

Macros

- #define CVA_VC_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

Functions

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

  Returns the version number of the library.

7.3.1 Macro Definition Documentation

7.3.1.1 #define CVA_VC_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.
*/

/*
    This example program classifies an image supplied by the user and prints
    the probabilities of it belonging to each class.
*/

#include <cva/vc/vc.hpp>

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

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

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

namespace ac = cva::ac;

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

    cv::CommandLineParser parser(argc, argv,
        "{ help h |        | print this message }"
        "{ @image | <none> | image to classify }"
        "{ impl   |  CPU   | classifier 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");

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;
}

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

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

// Allocate space for the probabilities.

std::vector<float> color_probs(cva::vc::VehicleClassifier::numColors
  ());
std::vector<float> type_probs(cva::vc::VehicleClassifier::numTypes(
  ));

ac::ConstImageView image_view = ac::ocv::toImageView(ac::ImageFormat::BGR_8, image);
ac::Span<float> color_probs_span = ac::toSpan(color_probs);
ac::Span<float> type_probs_span = ac::toSpan(type_probs);

// Create the classifier and perform the classification. Note that OpenCV
// loads images in the BGR format.

auto classifier = impl == "CPU"
    ? cva::vc::VehicleClassifier::createCpu()
    : impl == "GPUFP32"
        ? cva::vc::VehicleClassifier::createGpu(
  ac::Precision::FP32)
        : cva::vc::VehicleClassifier::createGpu(
  ac::Precision::FP16);

classifier->classify({&image_view, 1},
    {&color_probs_span, 1}, {&type_probs_span, 1});

// We need to sort the probabilities, but we don't want to lose track of
// their original indexes, since we'll need those indexes to look up the
// class descriptions. So we create a separate vector with just the indexes,
// and sort those instead, using a custom comparator.

std::vector<std::size_t> color_indexes(cva::vc::VehicleClassifier::numColors
  ());
std::vector<std::size_t> type_indexes(cva::vc::VehicleClassifier::numTypes
  ());
std::iota(color_indexes.begin(), color_indexes.end(), 0);
std::iota(type_indexes.begin(), type_indexes.end(), 0);

std::sort(color_indexes.begin(), color_indexes.end(),
    [&color_probs](std::size_t left, std::size_t right) {
      return color_probs[left] > color_probs[right];
    });

std::sort(type_indexes.begin(), type_indexes.end(),
```

```
        [&type_probs](std::size_t left, std::size_t right) {
          return type_probs[left] > type_probs[right];
        });


    // Print the top probabilities and the descriptions of the corresponding classes.

    std::cout << std::fixed << std::setprecision(2);

    for (auto color_index: color_indexes)
    {
        std::cout << std::setw(5) << color_probs[color_index] * 100.f
                  << "% - " << cva::vc::VehicleClassifier::colorDescription
      (color_index) << '\n';
    }

    std::cout << '\n';

    for (auto type_index: type_indexes)
    {
        std::cout << std::setw(5) << type_probs[type_index] * 100.f
                  << "% - " << cva::vc::VehicleClassifier::typeDescription
      (type_index) << '\n';
    }

    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;
}
```