onnx_runtime_cpp

xmba15, Bey Hao Yun

Aug 28, 2021

CONTENTS:

onnx_runtime_cpp is a small library that contains CPP-based example codes that shows how onnxruntime can be applied to your project.

CHAPTER

ONE

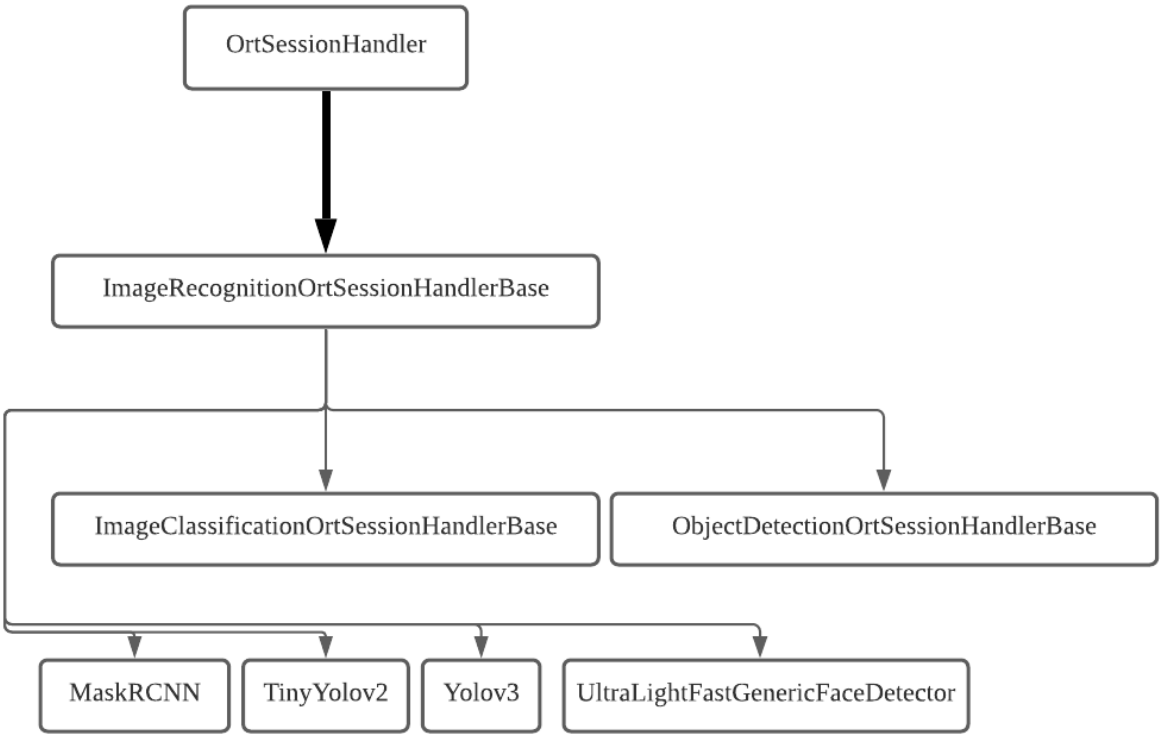
INDICES AND TABLES

- genindex
- modindex
- search

1.1 Table of Contents

1.1.1 Codebase Architecture

The illustration below shows the **inheritance hierachy** within the codebase. In other words, from this diagram, you can get a clear sense which modules depend on what.



Other Information

Note that **OrtSessionHandlerIml** private class does the real work here.

a. **Defines** DataOutputType to be std::pair<float*, std::vector<std::int64_t>>

b. **Defines** a private class called OrtSessionHandlerIml.

c. **Defines** a toString policy on how to printing the input shapes and data types in DEBUG_LOG function calls.

1.1.2 API

include

Constants

Copyright (c) organization

Author btran

namespace Ort

Variables

const std::vector<std::string> **IMAGENET_CLASSES**
A vector of strings that contains all 1000 object classes in Imagenet Dataset.

constexpr int64_t **IMAGENET_NUM_CLASSES** = 1000
A 64 bit long integer that stores the number of Imagenet object classes. This variable is used in examples/TestImageClassification.cpp and examples/TestObjectDetection.cpp.

const std::vector<float> **IMAGENET_MEAN** = {0.406, 0.456, 0.485}
A BGR mean that helps normalize an input image.

const std::vector<float> **IMAGENET_STD** = {0.225, 0.224, 0.229}
A BGR standard deviation that helps normalize an input image. This is used in tandem with IMAGENET_NUM_CLASSES.

const std::vector<std::string> **MSCOCO_CLASSES** = {"background", "person", "bicycle", "car", "motorbike", "aeroplane", "bus
A vector of strings that contains all 81 object classes in MSCOCO Dataset.

constexpr int64_t **MSCOCO_NUM_CLASSES** = 81
A 64 bit long integer that stores the number of MSCOCO object classes. This variable is used in examples/MaskRCNNApp.cpp.

const std::vector<std::array<int, 3>> **MSCOCO_COLOR_CHART** = generateColorCharts(*MSCOCO_NUM_CLASSES*)
A vector of array of 3 values that is mapped with cv::Scalar that corresponds to the MSCOCO object classes. It calls generateColorCharts function which is defined under Utility.hpp.

const std::vector<std::string> **VOC_CLASSES** = {"aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "c
A vector of strings that contains all 20 object classes in Pascal VOC Dataset.

constexpr int64_t **VOC_NUM_CLASSES** = 20
A 64 bit long integer that stores the number of Pascal VOC object classes. This variable is used in examples/TinyYolov2App.cpp.

const std::vector<std::array<int, 3>> **VOC_COLOR_CHART** = generateColorCharts(*VOC_NUM_CLASSES*)
A vector of array of 3 values that is mapped with cv::Scalar that corresponds to the Pascal VOC object classes. It calls generateColorCharts function which is defined under Utility.hpp.

ImageClassificationOrtSessionHandler

Copyright (c) organization

Author btran

namespace Ort

class ImageClassificationOrtSessionHandler : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*
#include <ImageClassificationOrtSessionHandler.hpp> An *ImageClassificationOrtSessionHandler* class object. This class inherits *ImageRecognitionOrtSessionHandlerBase* and is only used in TestImageClassication.cpp where squeezenet1.1.onnx is utilized according to the preface instructions on README.md.

Public Functions

ImageClassificationOrtSessionHandler(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)

This calls *ImageClassificationOrtSessionHandler*'s constructor. It also:

A. Initializes m_numClasses with numClasses.
B. Initializes an internal *OrtSessionHandler* with modelPath, gpuIdx and inputShapes.

~ImageClassificationOrtSessionHandler()

Calls standard destructor. Is empty destructor.

std::vector<std::pair<int, float>> **topK**(**const** std::vector<float*> &*inferenceOutput*, **const** uint16_t *k* = 1, **const** bool *useSoftmax* = true) **const**

A mutator function.

A. Utilizes external softmax function to parse resulting inference from model.
B. Map corresponding object class name index to resulting confidence score.
C. Output the top k pairs of object class name index and corresponding confidence score. K is arbitruarily set by user in TestImageClassication.cpp.

std::string **topKToString**(**const** std::vector<float*> &*inferenceOutput*, **const** uint16_t *k* = 1, **const** bool *useSoftmax* = true) **const**

A mutator function.

A. Utilizes external softmax function to parse resulting inference from model.
B. Map corresponding object class name string to resulting confidence score.
C. Print to terminal the top k pairs of object class name string and corresponding confidence score. K is arbitruarily set by user in TestImageClassication.cpp.

ImageRecognitionOrtSessionHandlerBase

Copyright (c) organization

Author btran

namespace Ort

class ImageRecognitionOrtSessionHandlerBase : **public** *Ort*::*OrtSessionHandler*

#include <ImageRecognitionOrtSessionHandlerBase.hpp> An *ImageRecognitionOrtSessionHandlerBase* class object. This class inherits from the base class, *OrtSessionHandler* and serves as the base class for *MaskRCNN*, *TinyYolov2*, *Yolov3* and *UltraLightFastGenericFaceDetector*.

Subclassed by *Ort::ImageClassificationOrtSessionHandler*, *Ort::MaskRCNN*, *Ort::ObjectDetectionOrtSessionHandler*, *Ort::TinyYolov2*, *Ort::UltraLightFastGenericFaceDetector*, *Ort::Yolov3*

Public Functions

ImageRecognitionOrtSessionHandlerBase(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)

This calls OrtSessionHandler's constructor. It also:

A. Initializes m_numClasses with numClasses.
B. Initializes empty m_classNames.
C. Populates m_classNames with numeric text strings corresponding to numClasses.

~ImageRecognitionOrtSessionHandlerBase()

Calls standard destructor. Is empty destructor.

void **initClassNames**(**const** std::vector<std::string> &*classNames*)

A mutator function.

A. Runs check on if input array of class names is equal to previously assigned number of classes in m_numClasses.
B. If true, assigns m_classNames with classNames.
C. Otherwise, report an error.

void **preprocess**(float **dst*, **const** unsigned char **src*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChanels*, **const** std::vector<float> &*meanVal* = {}, **const** std::vector<float> &*stdVal* = {}) **const**

A mutator function.

Given a certain format of an image like cv::Mat, populate dst to used during inferencing. The only difference it has as compared to the overriding counterparts is that it considers input arguments, meanVal and stdVal when populating dst.

uint16_t **numClasses**() **const**

A getter function. Returns the number of classes defined before.

const std::vector<std::string> &**classNames**() **const**

A getter function. Returns the string of class names.

Protected Attributes

const uint16_t **m_numClasses**
The number of classes defined beforehand.

std::vector<std::string> **m_classNames**
The the string of class names defined beforehand.

ObjectDetectionOrtSessionHandler

Copyright (c) organization

Author btran

namespace Ort

class ObjectDetectionOrtSessionHandler : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*
#include <ObjectDetectionOrtSessionHandler.hpp> An *ObjectDetectionOrtSessionHandler* class object. This class inherits *ImageRecognitionOrtSessionHandlerBase* and is used by TestObjectDetection.cpp under examples.

Public Functions

ObjectDetectionOrtSessionHandler(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)
This calls *ObjectDetectionOrtSessionHandler*'s constructor. It also:

A. Initializes m_numClasses with numClasses.
B. Initializes an internal *OrtSessionHandler* with modelPath, gpuIdx and inputShapes.

~ObjectDetectionOrtSessionHandler()
Calls standard destructor. Is empty destructor.

OrtSessionHandler

Copyright (c) organization

Author btran

Date 2020-04-19

namespace Ort

class OrtSessionHandler
#include <OrtSessionHandler.hpp> An *OrtSessionHandler* class object. This class serves as the base parent class which is inherited by class object , *ImageRecognitionOrtSessionHandlerBase*. This class is implemented, following Pointer to Implementation (pimpl) C++ programming methodology.

Subclassed by *Ort::ImageRecognitionOrtSessionHandlerBase*

Public Types

using DataOutputType = std::pair<float*, std::vector<int64_t>>
An alias that represents a data structure which pairs a float pointer to a std::vector of 64 bit long integer.

Public Functions

OrtSessionHandler(**const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)
A Constructor function.

~OrtSessionHandler()
A Deconstructor function.

std::vector<*DataOutputType*> **operator()**(**const** std::vector<float*> &*inputImgData*)
A custom operator which serves as the main function call that processes an input image and outputs the resulting tensor.

Private Members

std::unique_ptr<OrtSessionHandlerIml> **m_piml**
An opaque pointer to OrtSessionHandlerIml which is called within *OrtSessionHandler* constructor function.

Utility

Warning: doxygenfile: Found multiple matches for file "Utility.hpp

examples

MaskRCNN

Copyright (c) organization

Author btran

Date 2020-05-18

namespace Ort

class MaskRCNN : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*

Public Functions

MaskRCNN(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)

~MaskRCNN()

void **preprocess**(float **dst*, **const** float **src*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChannels*) **const**

void **preprocess**(float **dst*, **const** cv::Mat &*imgSrc*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChannels*) **const**

Public Static Attributes

constexpr int64_t **MIN_IMAGE_SIZE** = 800

constexpr int64_t **IMG_CHANNEL** = 3

MaskRCNNApp

Copyright (c) organization

Author btran

Date 2020-05-18

Functions

int **main**(int *argc*, char **argv*[])

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Read in input image using Opencv.

d. Instantiate a MaskRCNN class object and initialize it with the total number of prefined MSCOCO_NUM_CLASSES for an input onnx model with the file path to referenced onnx model.

e. Initialize the classNames in the class object with MSCOCO_CLASSES as defined under Constants.hpp.

f. Initializes a float-type vector variable called dst that takes into account 3 channels for expected input RGB images and a padded image.

g. Calls processOneFrame function which is defined in the same script here and gets the output detection result in the form of an image.

   a. Pads the the input RGB image proportionally to a minimum 800 pixels by 800 pixels input format for MaskRCNN.

   b. Calls preprocess function to convert the resized input image matrix to 1-dimensional float array.

   c. Run the inference with the 1-dimensional float array.

d. Extract the anchors and attributes value from the inference output, storing them in numAnchors and numAttrs variables.

e. Convert the inference output to appropriately segregated vector outputs that capture bounding boxes information, corresponding scores and class indices. Filters out any bounding box detection that falls below the defalt 0.15 confidence threshold which is pre-defined in the auxillary function call for processOneFrame.

f. If the number of bounding boxes in the inference output is zero, just return the original input image.

i. Calls the visualizeOneImageWithMask function which is defined in examples/Utilty.hpp and returns an output image with all bounding boxes with segmentation masks, class labels and confidence scores printed on image. This function call is done by default by the auxillary processOneFrame function with the boolean visualizeMask variable.

h. Write the output detection result into an image file named result.jpg.

Variables

constexpr const float **CONFIDENCE_THRESHOLD** = 0.5

const std::vector<cv::Scalar> **COLORS** = toCvScalarColors(*Ort*::*MSCOCO_COLOR_CHART*)

TestImageClassification

Copyright (c) organization

Author btran

Functions

int **main** (int *argc*, char **argv*[])

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Instantiate an ImageClassificationOrtSessionHandler class object and initialize it with the total number of pretrained ImageNet Classes for squeezenet1.1.onnx with the file path to referenced onnx model.

d. Initialize the classNames in the class object with IMAGENET_CLASSES as defined under Constants.hpp.

e. Read in input image using Opencv.

f. Check if input image is empty. If so, output error and exit with failure.

g. Resize input image down to 244 by 244, as defined in this .cpp file.

h. Convert input image to 1-dimensional float array.

i. Pass 1-dimensional float array to ImageClassificationOrtSessionHandler preprocess function to account for ImageNet images Mean and Standard Deviation. This helps normalize the input image based on how squeezenet1.1.onnx was trained on ImageNet.

j. Start debug timer.

k. Pass normalized 1-dimensional float array to ImageClassificationOrtSessionHandler inference step and store in inferenceOutput variable.

l. Pass inferenceOutput to ImageClassificationOrtSessionHandler mutator function to output the first 5 pairs of object class name strings with their corresponding output confidence score.

m. Stop debug timer. Calculate and output to terminal the time taken to run 1000 rounds of inference.

Variables

constexpr int64_t **IMG_WIDTH** = 224

constexpr int64_t **IMG_HEIGHT** = 224

constexpr int64_t **IMG_CHANNEL** = 3

constexpr int64_t **TEST_TIMES** = 1000

TestObjectDetection

Copyright (c) organization

Author btran

Functions

int **main** (int *argc*, char **argv*[])

This source file is not mentioned in the preface README.md on how to use. Follow at your own risk.

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Instantiate an ObjectDetectionOrtSessionHandler class object and initialize it with the total number of pretrained ImageNet Classes for an unknown onnx model with the file path to referenced onnx model.

d. Initialize the classNames in the class object with IMAGENET_CLASSES as defined under Constants.hpp.

e. Read in input image using Opencv.

f. Check if input image is empty. If so, output error and exit with failure.

g. Resize input image down to 416 by 416, as defined in this .cpp file.

h. Convert input image to 1-dimensional float array.

i. Pass 1-dimensional float array to ObjectDetectionOrtSessionHandler preprocess function to account for ImageNet images Mean and Standard Deviation. This helps normalize the input image based on how squeezenet1.1.onnx was trained on ImageNet.

j. Start debug timer.

k. Pass normalized 1-dimensional float array to ObjectDetectionOrtSessionHandler inference step and store in inferenceOutput variable.

l. No output tensor or image is generated in this source file. It will only output the size of the inferenceOutput variable.

m. Stop debug timer. Calculate and output to terminal the time taken to run 1000 rounds of inference.

Variables

constexpr int64_t **IMG_WIDTH** = 416

constexpr int64_t **IMG_HEIGHT** = 416

constexpr int64_t **IMG_CHANNEL** = 3

constexpr int64_t **TEST_TIMES** = 1

TinyYolov2

Copyright (c) organization

Author btran

Date 2020-05-05

namespace Ort

class TinyYolov2 : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*

Public Functions

TinyYolov2(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)

~TinyYolov2()

void **preprocess**(float **dst*, **const** unsigned char **src*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChannels*) **const**

std::tuple<std::vector<std::array<float, 4>>, std::vector<float>, std::vector<uint64_t>> **postProcess**(**const** std::vector<DataO &*inference-Output*, **const** float *confidenceThresh* = 0.5) **const**

Public Static Attributes

constexpr int64_t **IMG_WIDTH** = 416

constexpr int64_t **IMG_HEIGHT** = 416

constexpr int64_t **IMG_CHANNEL** = 3

constexpr int64_t **FEATURE_MAP_SIZE** = 13 * 13

constexpr int64_t **NUM_BOXES** = 1 * 13 * 13 * 125

constexpr int64_t **NUM_ANCHORS** = 5

constexpr float **ANCHORS**[10] = {1.08, 1.19, 3.42, 4.41, 6.63, 11.38, 9.42, 5.11, 16.62, 10.52}

TinyYolov2App

Copyright (c) organization

Author btran

Functions

int **main** (int *argc*, char **argv*[])

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Read in input image using Opencv.

d. Instantiate an TinyYolov2 class object and initialize it with the total number of a custom FACE_CLASSES for an unknown onnx model with the file path to referenced onnx model.

e. Initialize the classNames in the class object with FACE_CLASSES as defined under Constants.hpp.

f. Initializes a float-type vector variable called dst that takes into account 3 channels for expected input RGB images and a fixed height of 800 pixels and a fixed width of 800 pixels.

g. Calls processOneFrame function which is defined in the same script here and gets the output detection result in the form of an image.

a. Resizes the the input RGB image proportionally to the fixed 416 pixels by 416 pixels input format for TinyYolov2.

b. Calls preprocess function to convert the resized input image matrix to 1-dimensional float array.

c. Run the inference with the 1-dimensional float array.

d. Extract the anchors and attributes value from the inference output, storing them in numAnchors and numAttrs variables.

e. Convert the inference output to appropriately segregated vector outputs that capture bounding boxes information, corresponding scores and class indices. Filters out any bounding box detection that falls below the defalt 0.5 confidence threshold which is pre-defined in the auxillary function call for processOneFrame.

f. If the number of bounding boxes in the inference output is zero, just return the original input image.

g. Perform Non-Maximum Suppression on the segregated vector outputs and filter out bounding boxes with their corresponding confidence score and class indices, based on 0.6 nms threshold value. This value is defined in the auxillary function call for processOneFrame.

h. Store the filtered results from afterNmsBboxes and afterNmsIndices variables.

i. Calls the visualizeOneImage function which is defined in examples/Utilty.hpp and returns an output image with all bounding boxes with class label and confidence score printed on image.

h. Write the output detection result into an image file named result.jpg.

Variables

constexpr const float **CONFIDENCE_THRESHOLD** = 0.5

constexpr const float **NMS_THRESHOLD** = 0.6

const std::vector<cv::Scalar> **COLORS** = toCvScalarColors(*Ort*::*VOC_COLOR_CHART*)

UltraLightFastGenericFaceDetector

Author btran

namespace Ort

class UltraLightFastGenericFaceDetector : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*

Public Functions

UltraLightFastGenericFaceDetector(**const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<std::int64_t>>> &*inputShapes* = std::nullopt)

~UltraLightFastGenericFaceDetector()

void **preprocess**(float **dst*, **const** unsigned char **src*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChannels*) **const**

Public Static Attributes

constexpr int64_t **IMG_H** = 480

constexpr int64_t **IMG_W** = 640

constexpr int64_t **IMG_CHANNEL** = 3

UltraLightFastGenericFaceDetectorApp

Author btran

Functions

int **main** (int *argc*, char **argv*[])

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Read in input image using Opencv.

d. Instantiate an UltraLightFastGenericFaceDetector class object and initialize it with the total number of a custom FACE_CLASSES for an unknown onnx model with the file path to referenced onnx model.

e. Initialize the classNames in the class object with FACE_CLASSES as defined under Constants.hpp.

f. Initializes a float-type vector variable called dst that takes into account 3 channels for expected input RGB images and a fixed height of 480 pixels and a fixed width of 640 pixels.

g. Calls processOneFrame function which is defined in the same script here and gets the output detection result in the form of an image.

   a. Resizes the the input RGB image proportionally to the fixed 480 pixels by 640 pixels input format for UltraLightFastGenericFaceDetector.

   b. Calls preprocess function to convert the resized input image matrix to 1-dimensional float array.

   c. Run the inference with the 1-dimensional float array.

   d. Extract the anchors and attributes value from the inference output, storing them in numAnchors and numAttrs variables.

   e. Convert the inference output to appropriately segregated vector outputs that capture bounding boxes information, corresponding scores and class indices. Filters out any bounding box detection that falls below the defalt 0.7 confidence threshold which is pre-defined in the auxillary function call for processOneFrame.

   f. If the number of bounding boxes in the inference output is zero, just return the original input image.

   g. Perform Non-Maximum Suppression on the segregated vector outputs and filter out bounding boxes with their corresponding confidence score and class indices, based on 0.3 nms threshold value. This value is defined in the auxillary function call for processOneFrame.

   h. Store the filtered results from afterNmsBboxes and afterNmsIndices variables.

   i. Calls the visualizeOneImage function which is defined in examples/Utilty.hpp and returns an output image with all bounding boxes with class label and confidence score printed on image.

h. Write the output detection result into an image file named result.jpg.

Variables

const std::vector<std::string> **FACE_CLASSES** = {"face"}

constexpr int64_t **FACE_NUM_CLASSES** = 1

const std::vector<std::array<int, 3>> **FACE_COLOR_CHART** = *Ort*::generateColorCharts(*FACE_NUM_CLASSES*, 2020)

constexpr const float **CONFIDENCE_THRESHOLD** = 0.7

constexpr const float **NMS_THRESHOLD** = 0.3

const std::vector<cv::Scalar> **COLORS** = toCvScalarColors(*FACE_COLOR_CHART*)

Utilty

Warning: doxygenfile: Found multiple matches for file "Utility.hpp

This file is not documented as per normal like other files due to duplicate **Utility.hpp** in `include` folder as well. This pertains to a documentation bug that is still unresolved in **Doxygen**. Please look at this GitHub issue for more details.

The following is an approximation of what the various functions do in **Utility.hpp** under `examples` folder.

toCvScalarColors

This function is called under `MaskRCNNApp.cpp`, `TinyYolov2App.cpp`, `UltraLightFastGenericFaceDetectorApp.cpp` and `Yolov3App.cpp`.

Takes in a vector of integer array and outputs a vector of cv::Scalar.

visualizeOneImage

This function is called under `MaskRCNNApp.cpp`, `TinyYolov2App.cpp`, `UltraLightFastGenericFaceDetectorApp.cpp` and `Yolov3App.cpp`.

Takes in the segregated vector outputs that contain the detection results, parses them and draws bounding boxes with corresponding class labels and confidence scores on the output RGB image.

visualizeOneImageWithMask

This function is called under `MaskRCNNApp.cpp`.

Takes in the segregated vector outputs that contain the detection results, parses them and draws segmentation masks and bounding boxes with corresponding class labels and confidence scores on the output RGB image.

Yolov3

Copyright (c) organization

Author btran

Date 2020-05-31

namespace Ort

class Yolov3 : **public** *Ort*::*ImageRecognitionOrtSessionHandlerBase*

Public Functions

Yolov3(**const** uint16_t *numClasses*, **const** std::string &*modelPath*, **const** std::optional<size_t> &*gpuIdx* = std::nullopt, **const** std::optional<std::vector<std::vector<int64_t>>> &*inputShapes* = std::nullopt)

~Yolov3()

void **preprocess**(float **dst*, **const** unsigned char **src*, **const** int64_t *targetImgWidth*, **const** int64_t *targetImgHeight*, **const** int *numChannels*) **const**

Public Static Attributes

constexpr int64_t **IMG_H** = 800

constexpr int64_t **IMG_W** = 800

constexpr int64_t **IMG_CHANNEL** = 3

Yolov3App

Copyright (c) organization

Author btran

Date 2020-05-31

Functions

int **main**(int *argc*, char **argv*[])

The following steps outlines verbosely what the code in this .cpp file does.

a. Checks if the number of commandline arguments is not exactly 3. If true, output verbose error and exit with failure.

b. Store the first commandline argument as the file path to the referenced onnx model and the second as the file path to input image.

c. Read in input image using Opencv.

d. Instantiate a Yolov3 class object and initialize it with the total number of a custom Bird Classes for an unknown onnx model with the file path to referenced onnx model.

e. Initialize the classNames in the class object with BIRD_CLASSES as defined under Constants.hpp.

f. Initializes a float-type vector variable called dst that takes into account 3 channels for expected input RGB images and a fixed height of 800 pixels and a fixed width of 800 pixels.

g. Calls processOneFrame function which is defined in the same script here and gets the output detection result in the form of an image.

a. Resizes the the input RGB image proportionally to the fixed 800 pixels by 800 pixels input format for Yolov3.

b. Calls preprocess function to convert the resized input image matrix to 1-dimensional float array.

c. Run the inference with the 1-dimensional float array.

d. Extract the anchors and attributes value from the inference output, storing them in numAnchors and numAttrs variables.

e. Convert the inference output to appropriately segregated vector outputs that capture bounding boxes information, corresponding scores and class indices. Filters out any bounding box detection that falls below the defalt 0.15 confidence threshold which is pre-defined in the auxillary function call for processOneFrame.

f. If the number of bounding boxes in the inference output is zero, just return the original input image.

g. Perform Non-Maximum Suppression on the segregated vector outputs and filter out bounding boxes with their corresponding confidence score and class indices, based on 0.5 nms threshold value. This value is defined in the auxillary function call for processOneFrame.

h. Store the filtered results from afterNmsBboxes and afterNmsIndices variables.

i. Calls the visualizeOneImage function which is defined in examples/Utilty.hpp and returns an output image with all bounding boxes with class label and confidence score printed on image.

h. Write the output detection result into an image file named result.jpg.

Variables

const std::vector<std::string> **BIRD_CLASSES** = {"bird_small", "bird_medium", "bird_large"}

constexpr int64_t **BIRD_NUM_CLASSES** = 3

const std::vector<std::array<int, 3>> **BIRD_COLOR_CHART** = *Ort*::generateColorCharts(*BIRD_NUM_CLASSES*)

constexpr const float **CONFIDENCE_THRESHOLD** = 0.2

constexpr const float **NMS_THRESHOLD** = 0.6

const std::vector<cv::Scalar> **COLORS** = toCvScalarColors(*BIRD_COLOR_CHART*)

INDEX

T