Program Listing for File data_processor.hpp

Return to documentation for file (`modules/holoinfer/src/process/data_processor.hpp`)

```
/* * SPDX-FileCopyrightText: Copyright (c) 2022-2024 NVIDIA CORPORATION &
AFFILIATES. All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * Licensed
under the Apache License, Version 2.0 (the "License"); * you may not use this file
except in compliance with the License. * You may obtain a copy of the License at * *
http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law
or agreed to in writing, software * distributed under the License is distributed on an
"AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. * See the License for the specific language governing
permissions and * limitations under the License. */ #ifndef
_HOLOSCAN_DATA_PROCESSOR_H #define _HOLOSCAN_DATA_PROCESSOR_H
#include <bits/stdc++.h> #include <cstring> #include <functional> #include
<iostream> #include <map> #include <memory> #include <sstream> #include
<string> #include <vector> #include <holoinfer.hpp> #include
<holoinfer_constants.hpp> #include <holoinfer_utils.hpp> #include
<holoscan/core/analytics/data_exporter.hpp> #include
<process/transforms/generate_boxes.hpp> namespace holoscan { namespace
inference { using processor_FP = std::function<InferStatus(const std::vector<int>&,
const void*, std::vector<int64_t>&, DataMap&, const std::vector<std::string>&
output_tensors, const std::vector<std::string>& custom_strings)>; // Declaration of
function callback for transforms that need configuration (via a yaml file). // Transforms
additionally support multiple inputs and outputs from the processing. using
transforms_FP = std::function<InferStatus(const std::string&, const
std::map<std::string, void*>&, const std::map<std::string, std::vector<int>>&,
DataMap&, DimType&)>; class DataProcessor { public: DataProcessor() {} InferStatus
initialize(const MultiMappings& process_operations, const std::string config_path);
InferStatus process_operation(const std::string& operation, const std::vector<int>&
in_dims, const void* in_data, std::vector<int64_t>& processed_dims, DataMap&
processed_data_map, const std::vector<std::string>& output_tensors, const
std::vector<std::string>& custom_strings); InferStatus process_transform(const
std::string& transform, const std::string& key, const std::map<std::string, void*>&
indata, const std::map<std::string, std::vector<int>>& indim, DataMap&
processed_data, DimType& processed_dims); InferStatus
compute_max_per_channel_cpu(const std::vector<int>& in_dims, const void*
in_data, std::vector<int64_t>& out_dims, DataMap& out_data_map, const
std::vector<std::string>& output_tensors); InferStatus scale_intensity_cpu(const
std::vector<int>& in_dims, const void* in_data, std::vector<int64_t>& out_dims,
DataMap& out_data_map, const std::vector<std::string>& output_tensors);
InferStatus print_results(const std::vector<int>& in_dims, const void* in_data);
InferStatus print_results_int32(const std::vector<int>& in_dims, const void* in_data);
InferStatus print_custom_binary_classification(const std::vector<int>& in_dims,
const void* in_data, const std::vector<std::string>& custom_strings); InferStatus
export_binary_classification_to_csv(const std::vector<int>& in_dims, const void*
in_data, const std::vector<std::string>& custom_strings); private: inline static const
std::map<std::string, holoinfer_data_processor> supported_compute_operations_{
{"max_per_channel_scaled", holoinfer_data_processor::h_HOST},
{"scale_intensity_cpu", holoinfer_data_processor::h_HOST}}; inline static const
std::map<std::string, holoinfer_data_processor> supported_transforms_{
{"generate_boxes", holoinfer_data_processor::h_HOST}}; // Map with operation name
as key, with pointer to its object std::map<std::string,
std::unique_ptr<TransformBase>> transforms_; inline static const
std::map<std::string, holoinfer_data_processor> supported_print_operations_{
{"print", holoinfer_data_processor::h_HOST}, {"print_int32",
holoinfer_data_processor::h_HOST}, {"print_custom_binary_classification",
holoinfer_data_processor::h_HOST}}; inline static const std::map<std::string,
holoinfer_data_processor> supported_export_operations_{
{"export_binary_classification_to_csv", holoinfer_data_processor::h_HOST}};
processor_FP max_per_channel_scaled_fp_ = [this](auto& in_dims, const void*
in_data, std::vector<int64_t>& out_dims, DataMap& out_data, auto&
output_tensors, auto& custom_strings) { return
compute_max_per_channel_cpu(in_dims, in_data, out_dims, out_data,
output_tensors); }; processor_FP scale_intensity_cpu_fp_ = [this](auto& in_dims,
const void* in_data, std::vector<int64_t>& out_dims, DataMap& out_data, auto&
output_tensors, auto& custom_strings) { return scale_intensity_cpu(in_dims, in_data,
out_dims, out_data, output_tensors); }; processor_FP print_results_fp_ = [this](auto&
in_dims, const void* in_data, std::vector<int64_t>& out_dims, DataMap& out_data,
auto& output_tensors, auto& custom_strings) { return print_results(in_dims,
in_data); }; processor_FP print_custom_binary_classification_fp_ = [this](auto&
in_dims, const void* in_data, std::vector<int64_t>& out_dims, DataMap& out_data,
auto& output_tensors, auto& custom_strings) { return
```

```
print_custom_binary_classification(in_dims, in_data, custom_strings); }; processor_FP
export_binary_classification_to_csv_fp_ = [this](auto& in_dims, const void* in_data,
std::vector<int64_t>& out_dims, DataMap& out_data, auto& output_tensors, auto&
custom_strings) { return export_binary_classification_to_csv(in_dims, in_data,
custom_strings); }; processor_FP print_results_i32_fp_ = [this](auto& in_dims, const
void* in_data, std::vector<int64_t>& out_dims, DataMap& out_data, auto&
output_tensors, auto& custom_strings) { return print_results_int32(in_dims, in_data);
}; const std::map<std::string, processor_FP> oper_to_fp_{
{"max_per_channel_scaled", max_per_channel_scaled_fp_}, {"scale_intensity_cpu",
scale_intensity_cpu_fp_}, {"print", print_results_fp_}, {"print_int32",
print_results_i32_fp_}, {"print_custom_binary_classification",
print_custom_binary_classification_fp_}, {"export_binary_classification_to_csv",
export_binary_classification_to_csv_fp_}}; transforms_FP generate_boxes_fp_ = [this]
(const std::string& key, const std::map<std::string, void*>& indata, const
std::map<std::string, std::vector<int>>& indim, DataMap& processed_data,
DimType& processed_dims) { return transforms_.at(key)->execute(indata, indim,
processed_data, processed_dims); }; const std::map<std::string, transforms_FP>
transform_to_fp_{ {"generate_boxes", generate_boxes_fp_}}; std::string config_path_
= {}; std::unique_ptr<DataExporter> data_exporter_ = nullptr; }; } // namespace
inference } // namespace holoscan #endif
```