INTERNATIONAL STANDARD

ISO/IEC 15938-18

First edition
2023-07

Information technology — Multimedia content description interface —

Part 18: Conformance and reference software for compression of neural networks

Technologies de l'information — Interface de description du contenu multimédia —

Partie 18: Conformité et logiciel de référence pour la compression des réseaux neuronaux

Reference number
ISO/IEC 15938-18:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 15938 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document describes conformance testing and the reference software for ISO/IEC 15938-17 Compression of neural networks for multimedia content description and analysis. The reference software includes both encoder and decoder functionality.

The reference software is useful in aiding users of a standard for coding neural networks to establish and test conformance and interoperability, and to educate users and demonstrate the capabilities of the standard. For these purposes, the accompanying software is provided as an aid for the study and implementation of 15938-17 compression of neural networks for multimedia content description and analysis.

The purpose of this document is to provide the following:

— A set of reference bitstreams conforming to ISO/IEC 15938-17.

— Description of procedures to test conformance of bitstreams and decoders to ISO/IEC 15938-17.

— Reference decoder software capable of decoding bitstreams that conform to ISO/IEC 15938-17 in a manner that conforms to the decoding process specified in ISO/IEC 15938-17.

— Reference encoder software capable of producing bitstreams that conform to ISO/IEC 15938-17.

Information technology — Multimedia content description interface —

Part 18: Conformance and reference software for compression of neural networks

1 Scope

This document specifies conformance testing procedures for implementations of ISO/IEC 15938-17 and provides conformance bitstreams. It also provides the reference software for ISO/IEC 15938-17 which is an integral part of this document.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 15938-17:2022, *Information technology — Multimedia content description interface — Part 17: Compression of neural networks for multimedia content description and analysis*

ISO/IEC 21778, *Information technology — The JSON data interchange syntax*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 15938-17 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
array
ordered list of elements where all elements are of the same type

3.2
dictionary
ordered list of key/value pairs where each key/value pair is an list of two elements with the first element being denoted ‘key’ and with the second element being denoted ‘value’

4 Conformance testing

4.1 General

Subclauses 4.2 through 4.6 specify tests for verifying the conformance of bitstreams as well as decoders. These tests make use of test data (bitstreams and related models) provided at https://standards.iso

.org/iso-iec/15938/-18/ed-1/en (described in detail in 4.4) and follow the procedure described in subclause 4.5.

4.2 Conformance testing for decoder

The decoder conformance is specified in ISO/IEC 15938-17:2022, Clause 7.

4.3 Conformance testing for bitstreams

The bitstream conformance is specified in ISO/IEC 15938-17:2022, Clause 7.

4.4 Models and reference bitstreams

A set of bitstreams and related neural network models is provided for conformance testing. When neural network models are provided, they are the source models used to generate one or more compressed bitstreams. The code defining these models is provided as part of the reference software distribution at https://standards.iso.org/iso-iec/15938/-18/ed-1/en. A bitstream can be generated:

— from a commonly used neural network model,

— from synthetic data by encoding it with the reference encoder,

— by creating a conformant synthetic bitstream directly, without using an encoder.

Where applicable, the dataset used to train the model is provided for information only. However, the dataset is not needed for conformance testing and thus not provided.

Table 1 summarizes the provided bitstreams. It also lists the features tested with the bitstreams, and the reference encoder configuration used to generate the bitstream (where applicable). If decoding the bitstream requires out-of-band parameters (e.g. information that is derived from the network topology description), those parameters are also provided. Source models and datasets are referred to using the following names:

— ImageNet for the dataset described in Reference [5],

— CIFAR-100 for the dataset described in Reference [6],

— DCase for the dataset described in Reference [7] and the model described in Reference [8],

— MobileNet V2 for model described in Reference [9],

— UC12B for the model described in Reference [10] and

— VGG16 for the model described in Reference [11].

Table 1 — Bitstreams and related models for conformance testing

Bitstream id	Source model	Data-set	Relevant technology ISO/IEC 15938-17:2022	Features tested	Reference encoder configuration
perf_map_sparse_MobileNetV2.nctm	MobileNetV2	ImageNet	6.3.4.3	sparsification performance map	qp_density = 2 scan_order = 1 approx_method = "codebook" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
perf_map_prune_DCase.nctm	DCase	DCase	6.3.4.3	pruning performance map	qp_density = 2 scan_order = 1 approx_method = "codebook" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
perf_map_sparse_prune_UC12B.nctm	UC12B		6.3.4.3	Sparsification and Pruning Performance map	qp_density = 2 scan_order = 1 approx_method = "codebook" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
perf_map_sparse_VGG16.nctm	VGG16	ImageNet	6.3.4.3	Sparsification Performance Map (pruned model)	qp_density = 2 scan_order = 1 approx_method = "codebook" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
prune_tpl_cont_sparse_bm_DCase.nctm	DCase	DCase	6.3.4.5	Prune Topology - sparse bitmask	encode_tpl_only = True partial_data_counter = 0

Table 1 *(continued)*

Bitstream id	Source model	Data-set	Relevant technology ISO/IEC 15938-17:2022	Features tested	Reference encoder configuration
prune_tpl_cont_prune_bm_VGG16.nctm	VGG16	Image Net	6.3.4.5	Prune Topology - prune bitmask	encode_tpl_only = True partial_data_counter = 0
prune_tpl_cont_comb_bm_VGG16.nctm	VGG16	Image Net	6.3.4.5	Prune Topology - combined bitmask	encode_tpl_only = True partial_data_counter = 0
prune_tpl_cont_prune_dictionary_DCase.nctm	DCase	DCase	6.3.4.5	Prune Topology - prune dictionary	encode_tpl_only = True topology_indexed_reference_flag = False partial_data_counter = 0
prune_tpl_cont_prune_dictionary_idx_ResNet50.nctm	ResNet50	Image Net	6.3.4.5	Prune Topology - prune dictionary (indexed elem id)	encode_tpl_only = True topology_indexed_reference_flag = True partial_data_counter = 0
tpl_reflist_DCase.nctm	DCase	DCase	6.3.4.5, 6.3.3.7	Topology Reflist	encode_tpl_only = True partial_data_counter = 0
partial_data_counter_VGG16_ndu_size_65536.nctm	VGG16	Image Net	6.3.3.1	Partial data counter	max_ndu_nnr_unit_size = 65536
partial_data_counter_VGG16_ndu_size_32768.nctm	VGG16	Image Net	6.3.3.1	Partial data counter	max_ndu_nnr_unit_size = 32768
partial_data_counter_VGG16_ndu_size_16384.nctm	VGG16	Image Net	6.3.3.1	Partial data counter	max_ndu_nnr_unit_size = 16384
partial_data_counter_DCase_ndu_size_2048.nctm	DCase	DCase	6.3.3.1	Partial data counter	max_ndu_nnr_unit_size = 2048
partial_data_counter_DCase_ndu_size_1024.nctm	DCase	DCase	6.3.3.1	Partial data counter	max_ndu_nnr_unit_size = 1024
deepCABAC_ResNet50_1_qp-38_qp_density2.nctm	ResNet50	Image Net	10 9.1.1 / 9.2.1	DeepCABAC entropy coding, uniform quantization	see verify_all.sh
dependent_quantization_ResNet50_2_qp-38_qp_density2.nctm	ResNet50	Image Net	9.1.3 / 9.2.3 6.3.3.7	Dependent scalar quantization	see verify_all.sh
deepCABAC_qp_density_MobileNetV2_3_qp-38_qp_density2.nctm	MobileNetV2	Image Net	9.2	QpDensity	see verify_all.sh

Table 1 *(continued)*

Bitstream id	Source model	Data-set	Relevant technology ISO/IEC 15938-17:2022	Features tested	Reference encoder configuration
deepCABAC_qp_density_MobileNetV2_4_qp-76_qp_density3.nctm	MobileNetV2	ImageNet	9.2	QpDensity	see verify_all.sh
block_scan_order_8x8_cabac_entry_points_ResNet50_5_qp-38_qp_density2.nctm	ResNet50	ImageNet	4.12 / 6.4.3.7 / 6.4.3.8 / 7.3.6	Block scan order / cabac entry points	see verify_all.sh
block_scan_order_16x16_cabac_entry_points_ResNet50_6_qp-38_qp_density2.nctm	ResNet50	ImageNet	4.12 / 6.4.3.7 / 6.4.3.8 / 7.3.6	Block scan order / cabac entry points	see verify_all.sh
codebook_signaling_MobileNetV2_7_qMse0.00001.nctm	MobileNetV2	ImageNet	9.1.3 / 9.2.2	Codebook-based quantization	see verify_all.sh
local_scaling_DCase_8_qp-38_qp_density2.nctm	DCase	DCase	8.2.7 / 8.3.7	Local scaling	see verify_all.sh
batchnorm_folding_MobileNetV2_9_qp-38_qp_density2.nctm	MobileNetV2	ImageNet	8.2.6 / 8.3.6	BatchNorm Folding	see verify_all.sh
out_of_band_signaling_ResNet50_10_qp-38_qp_density2.nctm	ResNet50	ImageNet	6.3.3.7 / 6.4.3.7	Out-of-band signaling [a]	see verify_all.sh
deepCABAC_8bit_ResNet50_PYTzoo_11_qp0_qp_density4.pt.nctm	ResNet50	ImageNet	9.1.1 / 9.2.1	Uniform quantization with limited precision (8bit)	see verify_all.sh
deepCABAC_8bit_MobileNetV2_PYTzoo_12_qp0_qp_density4.pt.nctm	MobileNetV2	ImageNet	9.1.1 / 9.2.1	Uniform quantization with limited precision (8bit)	see verify_all.sh
deepCABAC_4bit_VGG16_PYTzoo_13_qp0_qp_density4.pt.nctm	VGG16	ImageNet	9.1.1 / 9.2.1	Uniform quantization with limited precision (4bit)	see verify_all.sh
deepCABAC_8bit_UC12B_14_qp0_qp_density4.nctm	UC12B	CIFAR-100	9.1.1 / 9.2.1	Uniform quantization with limited precision (8bit)	see verify_all.sh

Table 1 *(continued)*

Bitstream id	Source model	Data-set	Relevant technology ISO/IEC 15938-17:2022	Features tested	Reference encoder configuration
deepCABAC_4bit_DCase_15_qp0_qp_density4.nctm	DCase	DCase	9.1.1 / 9.2.1	Uniform quantization with limited precision (4bit)	see verify_all.sh
perf_map_sparse_MobileNetV2_bw8.nctm	MobileNetV2	Image Net	6.3.4.3	sparsification performance map (8bit)	qp_density = 2 scan_order = 1 approx_method = "uniform" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
perf_map_prune_DCase_bw8.nctm	DCase	DCase	6.3.4.3	pruning performance map (8bit)	qp_density = 2 scan_order = 1 approx_method = "uniform" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
perf_map_sparse_prune_UC12B_bw8.nctm	UC12B		6.3.4.3	Sparsification and Pruning Performance map (8bit)	qp_density = 2 scan_order = 1 approx_method = "uniform" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0

Table 1 *(continued)*

Bitstream id	Source model	Data-set	Relevant technology ISO/IEC 15938-17:2022	Features tested	Reference encoder configuration
perf_map_sparse_VGG16_bw8.nctm	VGG16	Image Net	6.3.4.3	Sparsification Performance Map (pruned model) (8bit)	qp_density = 2 scan_order = 1 approx_method = "uniform" qp = 35 qp_density = 2 opt_qp = False disable_dq = True lambda_scale = 0.0 cb_size_ratio = 5000 q_mse = 0.00001 param_opt_flag = False cabac_unary_length_minus1 = 9 partial_data_counter = 0
partial_data_counter_DCase_ndu_size_2048_bw8.nctm	DCase	DCase	6.3.3.1	Partial data counter (8bit)	max_ndu_nnr_unit_size = 2048 approx_method = "uniform"
partial_data_counter_DCase_ndu_size_1024_bw8.nctm	DCase	DCase	6.3.3.1	Partial data counter (8bit)	max_ndu_nnr_unit_size = 1024 approx_method = "uniform"
[a] As specified in out_of_band_signaling_ResNet50_10_qp-38_qp_density2_oob.json					

The bitstreams and models are provided at https://standards.iso.org/iso-iec/15938/-18/ed-1/en/.

The parameters of the bitstreams, for which no detailed parameters are listed, can be obtained from the scripts given in column “Reference encoder configuration”, for example, verify_all.sh in “run/conformance_bitstreams” of the reference software. Furthermore, it also provides detailed reference software configurations for each test. If out-of-band parameters are required, they are provided in a file conforming to ISO/IEC 21778 (JSON), specified in a footnote.

4.5 Procedure to test decoders

4.5.1 General

For testing a decoder, all bitstreams that cover mandatory features as specified in ISO/IEC 15938-17 shall be tested. If support for optional features is claimed, also the bitstreams covering these features shall be tested.

On platforms without floating point support, or limited integer precision support, only those bitstreams indicated as "(8bit)" or "(4bit)" shall be tested. On platforms with floating point support, testing these limited precision bitstreams is optional.

For each bitstream, the procedure includes the following steps:

1) Decode bitstream B with the reference decoder to obtain the decoded model R_r according to subclause 4.5.2 if no out-of-band parameters are specified and according to subclause 4.5.3, otherwise,

2) Decode bitstream B with the decoder under test to obtain the decoded model R_t according to subclause 4.5.2 if no out-of-band parameters are specified and according to subclause 4.5.3, otherwise.

3) Compare every tensor in R_r against the corresponding tensor in R_t. The test shall be considered as passed if the following conditions hold for every pair of tensors:

   — If an integer tensor has been obtained as a result (decoding method specified in ISO/IEC 15938-17:2022, 7.3.2), the tensors shall be identical.

   — If a floating-point tensor has been obtained as the result of another decoding method, no corresponding pair of floating point values shall have an absolute difference larger than 0,000001.

4.5.2 Decoding self-contained NNC bitstreams

For self-contained bitstreams, the decoder shall not use any other inputs than the bitstream to be decoded.

4.5.3 Decoding NNC bitstreams using out-of-band parameters

The testing of decoding bitstreams using out-of-band parameters is agnostic to whether those are application provided or decoded from a particular third party topology format. The decoder shall use only the input bitstream and the parameters provided for this bitstream as specified in Table 1 as inputs.

4.6 Procedure to test bitstreams

In order to test bitstreams created with an encoder other than the reference encoder, the bitstream shall be decoded using the reference decoder. The bitstream is considered conformant, if no errors occur during the decoding process.

All bitstreams that have been created with the script "verify_all.sh" specified in Table 1 can be decoded with the script "decode_all.sh" in "run/conformance_bitstreams" of the reference software. The bitstreams are considered conformant, if no errors occur during the decoding process.

5 Reference software

5.1 General

The purpose of this clause is to provide the following:

— Reference decoder software capable of decoding bitstreams that conform to ISO/IEC 15938-17 in a manner that conforms to the decoding process specified in ISO/IEC 15938-17.

— Reference encoder software capable of producing bitstreams that conform to ISO/IEC 15938-17.

Some examples of uses that may be appropriate for the reference decoder software are as follows:

— As an illustration of how to perform the decoding process specified in ISO/IEC 15938-17.

— As the starting basis for the implementation of a decoder that conforms to ISO/IEC 15938-17.

— For testing the conformance of a decoder implementation with the decoding process specified in ISO/IEC 15938-17.

— For testing the conformance of a bitstream to the constraints specified for bitstream conformance in ISO/IEC 15938-17, as the software can detect and report many bitstream conformance violations.

However, the lack of the detection of any conformance violation by the reference decoder software should not be considered as definitive proof that the bitstream conforms to all constraints specified for bitstream conformance in ISO/IEC 15938-17.

Some examples of uses that may be appropriate for the reference encoder software are as follows:

— As an illustration of how to perform an encoding process that produces bitstreams that conform to the constraints specified for bitstream conformance in ISO/IEC 15938-17.

— As the starting basis for the implementation of an encoder that conforms to ISO/IEC 15938-17.

— As a means of generating bitstreams for testing the conformance of a decoder implementation with the decoding process specified in ISO/IEC 15938-17.

— As a means of evaluating and demonstrating examples of the quality that can be achieved by an encoding process that conforms to ISO/IEC 15938-17.

5.2 Software location and license

The reference software is available at https://standards.iso.org/iso-iec/15938/-18/ed-1/en/.

The software license is provided in the file LICENSE included with the software.

5.3 Software installation

Installation instructions are provided in the file Readme.md included with the software.

5.4 Software architecture

5.4.1 General

Figure 1 gives a general overview of the discussed modules that are commonly present in the end-to-end neural network compression pipeline. Each of the grey boxes specifies a module representing a type of compression tools. Therefore, the interfaces that define the input/output of these components are specified in order to increase the interoperability between the different components.

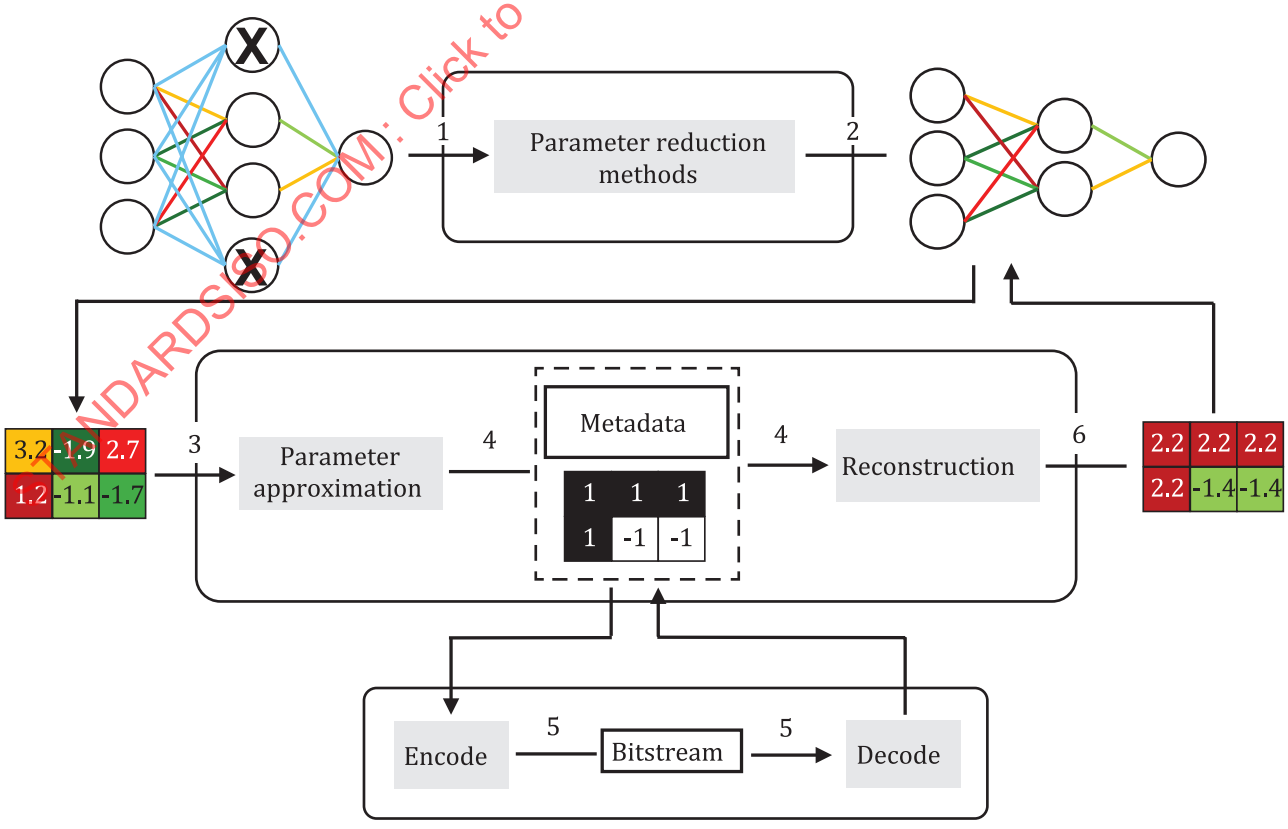

Figure 1 — Reference software architecture

The reference software supports out-of-band signalling to allow providing configurations and other parameters to the decoder for testing purposes. Those configurations and parameters are not part of the bitstream and expected to be provided by the application. This may ease testing of supported functionalities. Out-of-band signalling mechanisms are not part of conformance to ISO/IEC 15938-17.

In the following subclauses, float is defined as in 4.4 of ISO/IEC 15938-17:2022, and uint8, uint32, int32 conform to u(8), u(32), i(32) as defined in 6.1.3 of ISO/IEC 15938-17:2022. Mappings of the types to the Python®[1)] programming language are defined in Annex A.

5.4.2 Parameter reduction methods

This module represents the set of technologies that reduce the set of parameters of the network.

Input (#1): neural network (parameters + architecture) as specified by a particular deep learning framework. Examples of these frameworks are Keras®[2)]/Tensorflow®[3)] [2] or PyTorch®[4)] [1].

Output (#2): neural network (parameters + architecture) as specified by a particular deep learning framework. Examples of these frameworks are Keras/Tensorflow[2] or PyTorch[1].

5.4.3 Parameter approximation

This module applies parameter approximation techniques on the extracted parameter tensors. This may include techniques such as quantization, transformation, prediction, etc.

Input (#3)**:** A dictionary containing all parameter tensors of the network as float tensors and their respective names as keys. The names of each parameter tensor should be consistent with the nomenclature specified by the respective model definition provided by the coordinators of the model.

Output (#4): A dictionary as specified in subclause 5.5.2.

5.4.4 Reconstruction

Each parameter approximation technology provides a method that allows for reconstruction of the approximated parameter tensors resulting from the approximation procedure.

Input (#4): A dictionary in the same format as specified in output (#4) of the approximator.

Output (#6): A dictionary in the same format as specified in input (#3) of the approximator.

5.4.5 Encode

This module performs entropy encoding techniques for input data formats specified in ISO/IEC 15938-17. Auxiliary data shall be passed into the bitstream without entropy encoding in a serialized manner.

Input (# 4): A dictionary in the same format as specified in output (#4) of the approximator. The key names and the dictionary structure shall not be encoded into the bitstream.

Output (#5): A bitstream representing the entropy encoded input dictionary as array of uint8 values.

1) Python is the trademark of a product supplied by the Python Software Foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO/IEC of the product named.

2) Keras is the trademark of an API by François Chollet. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO/IEC of the product named.

3) TensorFlow is the trademark of a product supplied by Google LLC. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO/IEC of the product named.

4) PyTorch is the trademark of a product supplied by Facebook, Inc.. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO/IEC of the product named.

5.4.6 Decode

This module performs the lossless decoding of given bitstreams.

Input (#5): The bitstreams as specified in output (#5).

Output (#4): A dictionary in the same format as specified in output (#4) of the approximator, however, with the same keys and their items are bit-exact to each other.

5.5 Data structures and interfaces

This subclause describes data structures and methods that correspond to interfaces described above.

5.5.1 model_info: Shared model information

Information about the model architecture and the corresponding deep learning framework that may be required by the test model is provided as variable *model_info*.

The variable *model_info* is a dictionary containing the key/value pairs specified in Table 2.

Table 2 — Key/value pairs in model_info

Key	Value
'parameter_type'	Dictionary containing a key value pair per parameter tensor of the model. Each key is a string containing the name of the parameter tensor as specified by the model architecture and the corresponding value is one of the strings *'conv.weight'*, *'fc.weight'*, *'conv.bias'*, *'fc.bias'*, *'bn.beta'*, *'bn.gamma'*, *'bn.mean'*, *'bn.var'*, or *'unspecified'*.
'block_identifier'	Dictionary containing a key value pair per parameter tensor of the model that shall be grouped in a block. Each key is a string containing the name of the parameter tensor as specified by the model architecture and the corresponding value is a string identifying a block in the model architecture the parameter belongs to.
'parameter_dimensions'	Dictionary containing a key value pair per parameter tensor of the model for which the parameter dimensions are not present in the bitstream. Each key is a string containing the name of the parameter tensor as specified by the model architecture and the corresponding value is a list containing the dimensions of the corresponding parameter tensor.
'parameter_index'	Dictionary containing a key value pair per parameter tensor of the model. Each key is a string containing the name of the parameter tensor as specified by the model architecture and the corresponding value is an unsigned integer index. Sorting the integer indexes in ascending order defines the order of the parameter tensors.
'topology_storage_format'	An integer value representing syntax element topology_storage_format. A value equal to 'None' indicates that no topology unit is contained in the bitstream.
'performance_maps'	Dictionary containing a key value pair per included performance map. Each key is either *'sparsification'* or *'pruning'* depending on the performance map type and each corresponding value is a Dictionary containing the performance map information.
'prune_rep_type'	Integer indicating the type of the prune information. It should match either the identifier NNR_TPL_BMSK or NNR_TPL_DICT.
'sparse_bit_mask'	A list of zero/one values indicating parameter weights that should be zeroed when performing sparsification.
'bit_mask_order_flag'	A Boolean specifying the processing order of bit masks. True indicates row-major order and False indicates column-major order.
'topology_elem_id_list'	A list of strings representing the labels of each parameters in a model. Gets written into the topology reflist.

Each performance map included in *performance_maps* is a dictionary containing the key/value pairs depending on the performance map type of dictionary *model_info['performance_maps']* specified in Table 3.

Table 3 — Key/value pairs in model_info['performance_maps'].

Values in *model_info ['performance_maps']* contain	Key	Value
'sparsification'	*'count_thresholds'*	A uint8 specifying the size of *'sparsification_threshold'* and *'non_zero_ratio'*.
'sparsification'	*'sparsification_threshold'*	A list of float specifying the threshold which is applied to the weights of the decoded neural network in order to set the weights to zero.
'sparsification'	*'non_zero_ratio'*	A list of float specifying the non-zero ratios that are achieved by applying the sparsification thresholds in *'sparsification_threshold'* to sparsify the weights.
'pruning'	*'count_pruning_ratios'*	A uint8 specifying the size of the *'pruning_ratio'* list.
'pruning'	*'pruning_ratio'*	A list of float specifying the ratios of pruned weights to the total number of weights.
'sparsification' or 'pruning'	*'nn_accuracy'*	A list of float specifying the overall accuracies of the NN for different sparsification thresholds or pruning ratios.
'sparsification' or 'pruning'	*'count_classes'*	A list of uint8 specifying the number of classes for which separate accuracies are provided for each sparsification threshold or pruning ratio.
'sparsification' or 'pruning'	*'class_bitmask'*	A list of uint32 specifying a class bitmask for each threshold/ratio that defines the classes included in *'nn_class_accuracy'*. A 1-bit represents the presence of the class with a matching index.
'sparsification' or 'pruning'	*'nn_class_accuracy'*	A list of float specifying the accuracy for a certain class, when a certain sparsification threshold or pruning ratio is applied.

Depending on the values of *model_info['prune_rep_type'],* the dictionary contains one of the key/value pairs specified in Table 4.

Table 4 — Key/value pairs in model_info['prune_rep_type']

Value in *model_info ['prune_rep_type'] is*	Key	Value
NNR_TPL_BMSK	*'prune_bit_mask'*	A list of zero/one values indicating parameter weights that should be pruned when performing pruning.
NNR_TPL_DICT	*'prune_tpl_elem_ids'*	A list of strings identifying the parameters for which new dimensions are specified. Should be a unique identifier as long as the topology definition is active.
NNR_TPL_DICT	*'prune_tpl_count_dims'*	A list of uint32 values indicating the number of dimensions for each parameter.
NNR_TPL_DICT	*'prune_tpl_dims'*	A list of uint32 values specifying the new dimensions of each parameter.

5.5.2 approx_data – Data structure for interface #4

Interfaces #3, #4, and #6 use a data structure defined as variable *approx_data*.

The variable *approx_data* is a dictionary containing the key/value pairs specified in Table 5.

Table 5 — Key/value pairs in approx_data

Key	Value
'approx_method'	Dictionary containing a key value pair per approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is specifying the parameter approximation method. Allowed values are '*uniform*' or '*codebook*'.
'parameters'	Dictionary containing a key value pair per parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is a tensor. In the case of a float parameter, it contains float values and in the case of an approximated parameter, it contains int32 values.
'compressed_parameter_types'	Dictionary containing a key value pair per block identifier of the model. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element *compressed_parameter_types*.
'decomposition_rank'	Dictionary containing a key value pair per block identifier of the model for which low-rank decomposition is used. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element *decomposition_rank*.
'g_number_of_rows'	Dictionary containing a key value pair per block identifier of the model for which low-rank decomposition is used. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element *g_number_of_rows*.
'scan_order'	Dictionary containing a key value pair per weight parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is identical to syntax element *scan_order*.
'dq_flag'	Dictionary containing a key value pair per approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is a flag indicating whether dependent quantization is used.
'qp'	Dictionary containing a key value pair per approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is an int32 value specifying the quantization parameter.
qp_density'	An int32 value specifying syntax element *qp_density*.

The parameter tensors of the model are derived from *model_info* and from *approx_data['compressed_parameter_types']*. This may result in added or removed parameters relative to model_info dependng on the values in *approx_data['compressed_parameter_types']*.

In addition, the dictionary contains the key/value pairs depending on the values of dictionary *approx_data['approx_method']* specified in Table 6.

Table 6 — Key/value pairs in approx_data['approx_method']

Values in *approx_data['approx_method']* contain (at least once)	Key	Value
'codebook'	*'codebooks'*	Dictionary containing a key value pair per codebook-approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is an int32 array specifying the codebook values.
'codebook'	*'codebook_zero_offsets'*	Dictionary containing a key value pair per codebook-approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is a int32 value specifying the element of the codebook that corresponds to integer index 0.
'codebook'	*'codebooks_egk'*	Dictionary containing a key value pair per codebook-approximated parameter tensor of the model. Each key is a string containing the name of the parameter tensor and the corresponding value is an int32 value specifying syntax element *codebook_egk*.

The metadata specific to the approximation method shall contain all necessary information required for the later reconstruction process. This is to ensure that the subsequent reconstruction method will be able to entirely reconstruct the int32 parameter tensors of the network back to a float representation. The metadata may be shared across parameters (global), or may also be parameter-specific (local). For the latter, a respective mapping between the parameter name and the metadata should be provided. In order to ensure flexibility across different approximation methods, the choice of the specific format of the metadata is up to the particular method. However, the use of a dictionary structure is recommended in order to ensure consistency.

5.5.3 nctm – Main module

5.5.3.1 General

This module provides the submodules `approximator`, `coder` and `approx_coder`.

5.5.3.2 nctm.approximator – Parameter approximation module

This module provides an interface definition for parameter approximation and reconstruction.

```
nctm.approximator.approx(approx_info, model_info, approx_data)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

Variable *approx_info* is a dictionary containing the key/value pairs specified in Table 7.

Table 7 — Key/value pairs in approx_info

Key	Value
'approx_method'	String specifying the parameter approximation method. Allowed values are *'uniform'* and *'codebook'*.
'to_approximate'	A list of strings identifying the parameter types that are approximated.

Table 7 *(continued)*

Key	Value
'dq_flag'	Dictionary containing a key value pair per parameter tensor of the model that are approximated. Each key is a string containing the name of the parameter tensor and the corresponding value is flag indicating whether dependent quantization is used.
'lambda_scale'	A float value specifying the lambda scale value to be used in dependent quantization.
'cabac_unary_length_minus1'	An int32 value specifying the syntax element *cabac_unary_length_minus1*.

The dictionary *approx_info* shall contain further keys value pairs depending on the value *approx_info['approx_method']* according to Table 8.

Table 8 — Key/value pairs in approx_info['approx_method']

approx_method	Key	Value
'uniform'	*'qp'*	Dictionary containing a key value pair per parameter tensor of the model that are approximated. Each key is a string containing the name of the parameter tensor and the corresponding value is an int32 value specifying the quantisation parameter.
'codebook'	*'qMse'*	A float value specifying the maximum mean squared error between the quantized tensors and the original tensors. If the quantization algorithm fails to achieve this MSE for a tensor using the maximum quantization bits specified by *qMax*, then the tensor is not quantized.
'codebook'	*'cb_size_ratio'*	A float value specifying the minimum ratio of tensor to codebook size.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

Returns a dictionary according to subclause 5.5.2. The approximation methods are specified in ISO/IEC 15938-17:2022, Clause 8.

```
nctm.approximator.rec(approx_data, model_info)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

Quantized values in variable *approx_data* are replaced with reconstructed versions. The employed reconstruction methods are specified in ISO/IEC 15938-17:2022, subclause 7.3.

```
nctm.approximator.fold_bn(model_info, approx_data, approx_info)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

Variable *approx_info* is a dictionary containing the key/value pairs specified in Table 7.

The four batch-norm parameters of each batch-norm layer in *approx_data* are replaced with only two parameters, a scaling and a bias parameter. The two parameters are obtained by merging the batch-norm parameters as specified in ISO/IEC 15938-17:2022, subclause 8.2.6.

```
nctm.approximator.unfold_bn(model_info, approx_data)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

Replaces scaling and bias parameters in *approx_data* with recreated equivalent batch-norm parameters. The recreation process is specified in ISO/IEC 15938-17:2022, subclause 8.2.6.

```
nctm.approximator.set_lsa(model_info, approx_data, lsa_params)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

The variable *lsa_params* is a dictionary containing a key value pair per parameter tensor of the model for which scaling parameters are specified. Each key is a string containing the name of the parameter tensor the scaling parameters are applied to and the corresponding value is a 1D tensor. In the case of a float parameter, it contains float values and in the case of an approximated parameter, it contains int32 values.

Adds scaling parameters for each tensor specified by *lsa_params* to *approx_data* as specified in ISO/IEC 15938-17:2022, subclause 8.2.6. The reference software provides an algorithm for deriving values for *lsa_params,* although it is out of scope of the standard.

```
nctm.approximator.apply_lsa(model_info, approx_data)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

Applies the scaling parameters to each tensor specified and replaces the parameter tensors in *approx_data* with the scaled versions. Then removes the scaling parameters from *approx_data.* The process is specified in ISO/IEC 15938-17:2022, subclause 8.2.6.

5.5.3.3 nctm.coder – Entropy coding module

This module provides access to lossless entropy encoding and decoding functions.

```
nctm.coder.encode(enc_info, model_info, approx_data)
```

The variable *model_info* is a dictionary according to subclause 5.5.1.

Variable *enc_info* is a dictionary containing the keys/value pairs specified in Table 9.

Table 9 — Key/value pairs in enc_info

Key	Value
'cabac_unary_length_minus1"	An int32 value specifying the syntax element cabac_unary_length_minus1.
'param_opt_flag'	A flag specifying whether cabac optimization is used.
'encode_tpl_only'	A Boolean specifying if model data should be written into the bitstream. If True, only topology data is written and no approximation data etc. is written.
'partial_data_counter'	A uint8 value specifying the number of partitions for the payload of each ndu nnr unit. Directly sets the initial value of the syntax element *partial_data_counter*.
'max_ndu_nnr_unit_size'	An int32 value specifying the maximum size (in bytes) of a ndu nnr unit. The payloads of ndu nnr units are partitioned to achieve the desired size. Overrides the value of *'partial_data_counter'*.
'topology_indexed_reference_flag'	A Boolean value specifying if indexed references are used. Setting the flag to True enables element IDs to be coded using indices. If False, string-based labels are used instead.
'encode_tpl_only'	A Boolean specifying if model data is written into the bitstream. If True, only topology data is written and no approximation data etc. is written.

Table 9 *(continued)*

Key	Value
'out_of_band_signaling'	A Boolean specifying if input parameters (i. e. *count_tensor_dimensions*, *tensor_dimensions*, *cabac_unary_length_minus1*, *compressed_parameter_types*, *decomposition_rank* and *g_number_of_rows*) are signaled out of band. If True, none of these parameters is written to the bitstream but to a dictionary which is stored as JSON file. If False, each parameter, if present, is written to the bitstream.

The variable *approx_data* is a dictionary according to subclause 5.5.2.

The return value is a list containing the two elements *bs* and *oob_dictionary*, where *bs* is the encoded bitstream according to ISO/IEC 15938-17:2022, Clause 6 as a 1D array containing values of type uint8 and *oob_dictionary* is a dictionary containing key value pairs according to the table specifying the out-of-band parameters.

```
nctm.coder.decode(bitstream, model_info, oob_dictionary)
```

The variable *bitstream* is a 1D array containing values of type uint8 representing the bitstream that has been encoded with `nctm.coder.encode()`.

The variable *model_info* is either a dictionary according to subclause 5.5.1 or, in the case where the bitstream stores the required information in a topology NNR unit, it is 'None'.

The variable *oob_dictionary* is either a dictionary containing the out of band parameters according to Table 10 or, in the case where all input parameters are obtained from the bitstream, it is 'None'.

Table 10 — Key/value pairs in oob_dictionary

Key	Value
'cabac_unary_length_minus1'	An int32 value specifying the syntax element *cabac_unary_length_minus1*.
'tensor_dimensions'	Dictionary containing a key value pair per tensor of the model. Each key is a string containing the name of the parameter tensor as specified by the model architecture and the corresponding value is a list containing the dimensions of the corresponding parameter tensor.
'count_tensor_dimensions'	An int32 value specifying the length of the *tensor_dimensions* list, and thus the number of tensor dimensions.
'compressed_parameter_types'	Dictionary containing a key value pair per block identifier of the model. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element compressed_parameter_types.
'decomposition_rank'	Dictionary containing a key value pair per block identifier of the model for which low-rank decomposition is used. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element *decomposition_rank*.
'g_number_of_rows'	Dictionary containing a key value pair per block identifier of the model for which low-rank decomposition is used. Each key is a string containing the name of the block identifier as specified in *model_info* and the corresponding value is identical to syntax element *g_number_of_rows*.

Returns a dictionary *approx_data* that is identical to the corresponding dictionary passed to `nctm.coder.encode()` during the encoding of the bitstream.

5.5.4 nctm.nnr_model – Module for handling model related functionalities

```
nctm.nnr_model.NNRModel.add_performance_map
```

This function allows adding performance map definitions into the *model_info* structure. Only one type of performance map is added per function call.

`nctm.nnr_model.NNRModel.add_performance_map(sparsification_thresholds, non_zero_ratios, pruning_ratios, nn_accuracy, class_bitmasks, nn_class_accuracy)`

— The variable sparsification_thresholds is a list containing the sparsification thresholds that are included in the performance map (the list contains entries when a sparsification performance map is specified, and is empty otherwise).

— The variable non_zero_ratios is a list containing the non zero ratios for each sparsification threshold included in the performance map (the list contains entries when a sparsification performance map is specified, and is empty otherwise).

— The variable pruning_ratios is a list containing the pruning ratios that are included in the performance map (the list contains entries when a pruning performance map is specified, and is empty otherwise).

— The variable nn_accuracy is a list containing the overall accuracy of the model for each sparsification threshold or pruning ratio included in the performance map.

— The variable class_bitmasks is a list of integers representing a class bitmask. The number of 1-bits should match the respective number of class accuracies in nn_class_accuracy.

— The variable nn_class_accuracy is a nested list containing class accuracies for each sparsification threshold or pruning ratio included in the performance map.

— Does not return anything. Adds the new performance map to the model_info structure of the NNC Model.

`nctm.nnr_model.NNRModel.set_prune_topology`

This function allows adding the prune topology container definition, as specified in subclause 5.5.1, into the *model_info* structure. Additionally, the *'topology_storage_format'* in *model_info* is set to NNR_TPL_REFLIST.

`nctm.nnr_model.NNRModel.set_prune_topology(prune_bit_mask, prune_tpl_dictionary, sparse_bit_mask, bit_mask_order, pack_bit_masks)`

— The (optional) variable *prune_bit_mask* is a list of zero/one values indicating parameter weights that should be pruned when performing pruning. If *prune_bit_mask* is not specified, no prune bit mask is included in the topology definition. Should not be included if *prune_tpl_dictionary* is also specified.

— The (optional) variable *prune_tpl_dictionary* is a dictionary containing keys for each parameter, matching their element id string. The respective values should be lists of integers with the dimensions of the respective pruned parameter tensor. If *prune_tpl_dictionary* is not specified, no prune dictionary information is included in the topology definition. Should not be included if *prune_tpl_dictionary* is also specified.

— The (optional) variable *sparse_bit_mask* is a list of zero/one values indicating parameter weights that should be zeroed when performing sparsification. If *sparse_bit_mask* is not specified, sparse bit mask won't be included in the topology definition.

— The (optional) variable *bit_mask_order* is a Boolean specifying the processing order of bit masks. True indicates row-major order and False indicates column-major order.

— The (optional) variable *pack_bit_masks* is a Boolean specifying whether bitmasks are packed or not. If the value is True, individual bit mask values are packed into chunks of 8-bit integer values.

— Does not return anything. Adds the relevant prune topology related information to *model_info*.

`nctm.nnr_model.NNRModel.gen_sparse_bit_mask`

This function allows generating and setting a sparse bit mask to the *model_info* as specified in subclause 5.5.1. Uses the *approx_data* (#3) in the NNRModel to generate a sparse bit mask.