

IEEE Standard for Neural Network-Based Image Coding

IEEE Computer Society

Developed by the
Data Compression Standards Committee

IEEE Std 1857.11™-2024

STANDARDS

IEEE Standard for Neural Network-Based Image Coding

Developed by the

Data Compression Standards Committee
of the
IEEE Computer Society

Approved 26 September 2024

IEEE SA Standards Board

Abstract: A set of tools is defined in this standard for efficient image coding, including tools for encoding, for decoding, and for encapsulation. Some of the tools are based on trained neural networks. These tools are designed to perform block partitioning, prediction, transform, quantization, entropy coding, filtering, and so on.

Keywords: block partitioning, entropy coding, filtering, IEEE 1857.11™, image coding, neural network, prediction, quantization, transform

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2024 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 20 December 2024. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 979-8-8557-1415-9 STD27455
Print: ISBN 979-8-8557-1416-6 STDPD27455

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all IEEE standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers involved in technical working groups are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning all standards, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. IEEE Standards documents do not guarantee safety, security, health, or environmental protection, or compliance with law, or guarantee against interference with or from other devices or networks. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document should rely upon their own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus balloting process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English language version published by IEEE is the approved IEEE standard.

Use by artificial intelligence systems

In no event shall material in any IEEE Standards documents be used for the purpose of creating, training, enhancing, developing, maintaining, or contributing to any artificial intelligence systems without the express, written consent of IEEE SA in advance. “Artificial intelligence” refers to any software, application, or other system that uses artificial intelligence, machine learning, or similar technologies, to analyze, train, process, or generate content. Requests for consent can be submitted using the Contact Us form.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual is not, and shall not be considered or inferred to be, the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE or IEEE SA. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter’s views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s). News releases about IEEE standards issued by entities other than IEEE SA should be considered the view of the entity issuing the release rather than the formal position of IEEE or IEEE SA.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents.**

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and subcommittees of the IEEE SA Board of Governors are not able to provide an instant response to comments or questions, except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or revisions to an IEEE standard is welcome to join the relevant IEEE SA working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the [IEEE SA myProject system](#).¹ An IEEE Account is needed to access the application.

Comments on standards should be submitted using the [Contact Us](#) form.²

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the

¹ Available at: <https://development.standards.ieee.org/myproject-web/public/view.html#landing>.

² Available at: <https://standards.ieee.org/about/contact/>.

applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#).³ For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#).⁴ Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

³ Available at: <https://ieeexplore.ieee.org/browse/standards/collection/ieee>.

⁴ Available at: <https://standards.ieee.org/standard/index.html>.

Patents

IEEE standards are developed in compliance with the [IEEE SA Patent Policy](#).⁵

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

IMPORTANT NOTICE

Technologies, application of technologies, and recommended procedures in various industries evolve over time. The IEEE standards development process allows participants to review developments in industries, technologies, and practices, and to determine what, if any, updates should be made to the IEEE standard. During this evolution, the technologies and recommendations in IEEE standards may be implemented in ways not foreseen during the standard's development. IEEE standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, data privacy, and interference protection practices and all applicable laws and regulations.

⁵ Available at: <https://standards.ieee.org/about/sasb/patcom/materials.html>.

Participants

At the time this IEEE standard was submitted to the IEEE SA Standards Board for approval, the Audio Video Coding Working Group had the following membership:

Siwei Ma, *Chair*
Cliff Reader, *Vice Chair*

At the time this IEEE standard was submitted to the IEEE SA Standards Board for approval, the 1857.11 Sub-Working Group had the following membership:

Dong Liu, *Chair*
Zhan Ma, Jiaying Liu, *Vice Chairs*
Chuanmin Jia, *Secretary*
Li Li, *Technical Editor*

Qiaoxi Chen
Tong Chen
Yeda Chen
Yihang Chen
Zhenzhong Chen
Ding Ding
Cunhui Dong
Wenhong Duan
Semih Esenlik
Zejia Fan
Aolin Feng
Yi Feng
Changsheng Gao
Ziqing Ge
Xinyu Hang
Yueyu Hu
Dong Jiang
Yiheng Jiang
Bowe Kang
Yuzhuo Kong
Haowei Kuang
Houqiang Li

Qiang Li
Shujia Li
Yue Li
Xiongzhuan Liang
Junqi Liao
Jianping Lin
Jucan Lin
Weiyao Lin
Feiyang Liu
Haojie Liu
Shan Liu
Ming Lu
Haichuan Ma
Ke Ma
Siwei Ma
Yiyang Ma
Feihong Mei
Chunmei Nian
Junqi Shi
Xiaodi Shi
Xiaoyan Sun
Chuanbo Tang
Dezhao Wang

Huairui Wang
Shanshe Wang
Wei Wang
Xining Wang
Feng Wu
Xiangji Wu
Yaojun Wu
Yutao Xie
Xiaozhong Xu
Fan Ye
Qingzhu Yuan
Haotian Zhang
Jiahang Zhang
Kai Zhang
Li Zhang
Tianyu Zhang
Xuanye Zhang
Zhaobin Zhang
Zhe Zhang
Ziwen Zhang
Han Zhu
Zhirui Zuo

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Boon Chong Ang
Juan Carreon
Pin Chang
Diego Chiozzi
Werner Hoelzl
Piotr Karocki
Li Li
Dong Liu
Qian Liu

Jinghui Lu
Yonggang Luo
Luntian Mou
Rajesh Murthy
Bansi Patel
R K Rannow
Peter Saunderson
Jhony Sembiring
Walter Struppler

Bo Sun
Yonghong Tian
Yan Wang
Yaowei Wang
Lisa Ward
Feng Wu
Lu Yu
Yu Yuan
Li Zhang

When the IEEE SA Standards Board approved this standard on 26 September 2024, it had the following membership:

David J. Law, *Chair*
Jon Walter Rosdahl, *Vice Chair*
Gary Hoffman, *Past Chair*
Alpesh Shah, *Secretary*

Sara R. Biyabani
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Guido R. Hiertz
Ronald W. Hotchkiss

Hao Hu
Yousef Kimiagar
Joseph L. Koepfinger*
Howard Li
Xiaohui Liu
John Haiying Lu
Kevin W. Lu
Hiroshi Mano

Paul Nikolich
Robby Robson
Lei Wang
F. Keith Waters
Sha Wei
Philip B. Winston
Don Wright

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1857.11-2024, IEEE Standard for Neural Network-Based Image Coding.

Throughout history, neural networks have been investigated for image coding. From the 1980s to the 1990s, numerous studies were reported on neural network-based image coding. At that time, the networks were shallow and the compression efficiency was not satisfactory. Recently, neural networks, usually under the name of “deep learning,” have been reconsidered for image/video coding, which demonstrates great potential.

The Future Video Coding Study Group (FVC-SG) originated as a community of image/video coding experts. The FVC-SG has held regular meetings since 2014 and has been studying neural network-based image/video coding since 2015. In 2020, FVC-SG called for evidence on deep learning-based image coding technologies and received several proposals. Based on the received evidence, FVC-SG developed a reference image codec, which outperformed the existing standardized image coding solutions significantly in compression efficiency. Later on, FVC-SG reached the consensus to develop a standard for neural network-based image coding. The standard developing activity has been approved by the Data Compression Standards Committee of the IEEE Computer Society. Accordingly, the experts in FVC-SG formed the 1857.11 Sub-Working Group, which drafted this standard.

Contents

1. Overview	12
1.1 Scope	12
1.2 Purpose	12
1.3 Word usage	12
2. Normative references	13
3. Definitions	13
4. Conventions	17
4.1 Overview	17
4.2 Arithmetic operators	17
4.3 Logical operators	17
4.4 Relational operators	18
4.5 Bit-wise operators	18
4.6 Assignment operators	18
4.7 Tensor operators	19
4.7.1 Tensor operators defined in IEEE Std 2941-2021	19
4.7.2 Additional tensor operators	20
4.8 Mathematical functions	28
4.9 Structure relation	30
4.10 Description methods of bitstream syntax, parsing process, and decoding process	30
4.10.1 Description method of bitstream syntax	30
4.10.2 Functions	32
4.10.3 Descriptors	35
4.10.4 Reserved, forbidden and marker	36
5. Structure of coded bitstream	36
5.1 Picture	36
5.1.1 Picture format	36
5.2 Slice	38
6. Bitstream syntax and semantics	39
6.1 Syntax description	39
6.1.1 Start code	39
6.1.2 Definition of picture	40
6.1.3 Definition of slice	44
6.2 Semantics description	63
6.2.1 Picture	63
6.2.2 Slice	71
7. Parsing process	73
7.1 Overview	73
7.2 Entropy decoding engine	73
7.3 Parsing of latents	74
7.3.1 Parsing of latents for base profile	74
7.3.2 Parsing of latents for main profile	79
7.3.3 Parsing of latents for high profile	88

8. Decoding process.....	104
8.1 Overview	104
8.2 Decoding process for base profile	104
8.2.1 Quantized latent samples reconstruction.....	104
8.2.2 Synthesis transformation process	115
8.2.3 Image cropping	122
8.2.4 Adaptive resampling	123
8.2.5 YUV to RGB color space conversion	123
8.3 Decoding process for main profile.....	123
8.3.1 Overview.....	123
8.3.2 Picture decoding.....	124
8.3.3 Filtering.....	130
8.3.4 Image conversion and cropping	131
8.4 Decoding process for high profile	131
8.4.1 Overview.....	131
8.4.2 Picture decoding.....	132
8.4.3 Filtering.....	149
Annex A (normative) Pseudo start code.....	155
Annex B (normative) Models and lookup tables	156

IEEE Standard for Neural Network-Based Image Coding

1. Overview

1.1 Scope

This standard defines a set of tools for efficient image coding, including tools for encoding, for decoding, and for encapsulation. Some of the tools are based on trained neural networks and shall perform block partitioning, prediction, transform, quantization, entropy coding, filtering, and so on.

1.2 Purpose

This standard provides efficient, neural network-based coding tools for compression, decompression, and packaging of image data, which significantly improve the compression efficiency compared with IEEE Std 1857.4™-2018 (intrapicture coding) and IEEE Std 1857.10™-2021 (intrapicture coding) under comparable settings, and facilitate the compression and decompression on top of neural network-oriented computing infrastructures like neural network processing units (NPU)s.⁶ The target applications and services include but are not limited to Internet images, user-generated images, and other image-enabled applications and services such as digital image storage and communications.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).^{7, 8}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

⁶ Information on references can be found in Clause 2.

⁷ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

⁸ The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1857.4TM-2018, IEEE Standard for Second-Generation IEEE 1857 Video Coding.^{9, 10}

IEEE Std 1857.10TM-2021, IEEE Standard for Third-Generation Video Coding.

IEEE Std 2941TM-2021, IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management.

3. Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.¹¹

all-zero padding: Padding the extended area with zeros.

array: One-dimensional storage container for a set of data.

bicubic upsample: A way of upsampling. In the process of bicubic upsampling, when we fill the (i, j) position of the upsampled image, the corresponding position (i/scale, j/scale) in the original image is located at first. Then, the value at the position of (i, j) is calculated according to the 4 × 4 area points near (i/scale, j/scale) in the original image. The filled value at (i, j) equals the weighted summation of the 16 points in the 4 × 4 area, and the weight equals the vertical_weight × horizontal_weight. The values of vertical_weight and horizontal_weight can both be calculated by the following formula, where x represents the vertical and horizontal distances between the current point and (i/scale, j/scale) when calculating vertical_weight and horizontal_weight respectively:

$$w(x) = \begin{cases} 1 - 2|x|^2 + |x|^3, & |x| < 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3, & 1 \leq |x| < 2 \\ 0, & |x| \geq 2 \end{cases}$$

bin: Symbols that make up bin string, including “0” and “1”.

bin string: An ordered string of finite number of bins. The left-most bit is the most significant bit (MSB); the right-most bit is the least significant bit (LSB).

bitstream: The binary data stream formed by the coded pictures.

bitstream buffer: A buffer to store the bitstream.

bit string: An ordered string of a finite number of binary bits. The left-most bit is the MSB; the right-most bit is the LSB.

⁹ The IEEE standards or products referred to in Clause 2 are trademarks owned by The Institute of Electrical and Electronics Engineers, Incorporated.

¹⁰ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<https://standards.ieee.org/>).

¹¹ *IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>. An IEEE account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

block: An $M \times N$ (M -column by N -row) sample matrix, or an $M \times N$ transform coefficient matrix.

block scan: A specific sequential order of quantization coefficients.

byte: A bit string of 8 bits.

byte alignment: A bit in a bitstream is byte-aligned when its position is an integral multiple of 8 from the first binary bit in the bitstream.

chroma: A sample matrix or single sample of one of the two-color difference signals C_r and C_b .

coded picture: A coded representation of a picture.

color space: A mathematical model describing the way color can be represented as a group of numbers.

component: A single matrix of the three sample matrices (one luma matrix and two chroma matrices) of the picture or a single sample in the matrices.

convolutional layer: A convolutional layer is the main building block of a convolutional neural network (CNN). It is a learned filter with kernel size $M \times N \times K$.

convolutional neural network (CNN): A neural network used to handle structured data.

decoded picture: A reconstructed picture which is derived from the bitstream.

dequantization: The process by scaling quantized coefficients gets transform coefficients.

edge crop: Crop all edges (including top edge, bottom edge, left edge and right edge) with the same width of m .

NOTE—An edge crop is shown in Figure 1 .¹²

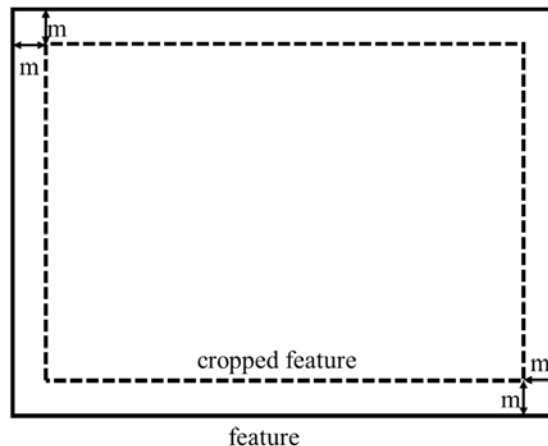


Figure 1—Edge crop with width of m

emulation prevention bits: Two bits “10” that appear after 22 consecutive “0”s are inserted into the bitstream to prevent a collision with the start code.

encoding: The process of converting the information of images into a coded form.

¹² Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

entropy decoding: The process of reconstructing data without any loss from a coded form.

equal value string: A string with all of its sample values equal to each other.

factorized model: A probability model that assumes each variable is independent.

feature map: The output of a layer whose size is $M \times N \times K$. M and N are the spatial size. K is the number of channels.

flag: A binary-state variable.

filtered sample: Sample obtained via filtering.

forbidden: Some values of syntax elements. These values shall not be used in bitstreams conforming to this standard.

Gaussian mixture model (GMM): A probabilistic model that uses the weighted average summation of different Gaussian models.

hyperprior: The side information used to construct the probability model of symbols to be coded.

Note—The aim of specifying forbidden values is usually to help avoid pseudo start code in the bitstream.

inverse transform: A part of the decoding process by which a set of transform coefficients are converted into spatial domain values.

iWave: A wavelet-like transform with learnable prediction and update filters.

latent: Variables that can only be inferred indirectly through a transform from image data.

level: It specifies a set of limits on the syntax elements and the values that may be taken by the syntax elements at a certain profile.

luma: Y represents the sample matrix or single sample of the luma signal.

masked convolutional layer: A type of convolutional layer that masks certain pixels so that the model can only predict based on pixels already seen.

model zoo: An online storage room for storing per-trained models.

parsing: The process reads a bitstream and produces syntax elements.

partition: The division of a set into subsets such that each element of the set is in exactly one of the subsets.

partition type: The type of organization of subsets obtained via partition.

patch: Adjacent number of largest coding units ordered consecutively in the raster scan.

patch block: The basic unit for coefficient probability estimation. Its size is $3 \times 1 \times 13 \times 13$. It shall include four types: LL, LH, HL, and HH.

picture: A picture can be a frame or a field.

profile: A specified subset of the syntax elements, semantics and algorithmic features of IEEE Std 1857.11.

quantization step: A parameter used by the decoding process for dequantization of quantized coefficients. The value of it represents the length of the quantization interval.

quantized coefficient: The value of transform coefficient before dequantization.

raster scan: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second and third rows, and so on. Rows are scanned from left to right.

reconstructed sample: Sample for composing decoded picture, which is decoded from bitstream by decoder.

reflection padding: Padding the extended area with mirror reflection pixels.

reserved: The term reserved, when used in the clauses specifying some values of a particular syntax element, is for future extensions of IEEE Std 1857.11. These values shall not be used in bitstreams conforming to IEEE Std 1857.11.

resolution level: The number of wavelets decomposition levels.

run: The number of consecutive identical data elements in the decoding process. Run refers to the number of the coefficients with a value of 0 preceding to a non-zero coefficient in block scan (in the block scan order).

sample: A basic element that forms a picture.

sample value: Amplitude value of a sample.

start code: A unique codeword of 32 bits embedded in the bitstream.

NOTE—Start code has many usages and can be used to identify the start of a syntax structure in bitstream.

stride: A parameter of the convolutional layer. It defines the sliding step length of the convolution kernel.

stuffing bits: A bit string inserted into the bitstream during encoding. It is disposed of before the decoding process.

subband: Frequency component obtained by wavelet transform.

subband block: A block of a subband. It is the basic unit in the parallel entropy decoding process.

syntax element: The result of parsing the data unit in the bitstream.

tensor: Multidimensional storage container for a set of data.

tensor squeeze: Squeeze the dimension of the tensor.

transform coefficient: A scalar quantity on the transform domain.

width height ratio: The ratio of the horizontal distance between the columns of luma samples in a picture to the vertical distance between the rows. It is expressed as $h \times v$, where h is the number of the samples in horizontal direction; v is the number of the samples in vertical direction.

4. Conventions

4.1 Overview

The mathematical operators and priority levels used in this standard are similar to those used in the C programming language. However, to reduce the possibility of decoding exceptions caused by floating-point errors, all operations involving bitstreams (operations in Clause 6 and Clause 7) are performed in fix-point numbers. In addition, integer division and arithmetic shift operations are specifically defined. Unless otherwise stated, numbering and counting conventions generally begin from 0.

4.2 Arithmetic operators

The following arithmetic operators are defined as Table 1.

Table 1—Definitions of arithmetic operators

Arithmetic operator	Definition
+	Additive operation.
−	Subtraction (as a binary operator) or negation (as a unary prefix operator).
×	Multiplication.
a^b	Exponentiation. Specifies a to the power of b . In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	Integer division with truncation of the result toward zero. For example, $7/4$ and $-7/-4$ are truncated to 1, $-7/4$ and $7/-4$ are truncated to −1.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{a}{b}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=a}^b f(i)$	The cumulative sum of $f(i)$ with the independent variable i taking all integer values from a up to and including b .
$a \% b$	Modular arithmetic. Remainder of a divided by b , where both a and b are positive integers.
$\lceil \cdot \rceil$	Round up.

4.3 Logical operators

The following logical operators are defined as Table 2.

Table 2—Definitions of logical operators

Logical operator	Definition
$a \&\& b$	Boolean logical “and” of a and b .
$a \parallel b$	Boolean logical “or” of a and b .
!	Boolean logical “not”.
$express ? a : b$	If the result of the expression $express$ is true or not 0, then a is used for assignment; otherwise, use b for assignment.

4.4 Relational operators

The following relational operators are defined as Table 3.

Table 3—Definitions of relational operators

Relational operator	Definition
>	Greater than.
>=	Greater than or equal to.
<	Less than.
<=	Less than or equal to.
==	Equal to.
!=	Not equal to.

4.5 Bit-wise operators

The following bit-wise operators are defined as Table 4.

Table 4—Definitions of bit-wise operators

Bit-wise operator	Definition
&	Bit-wise “and”.
	Bit-wise “or”.
~	Bit-wise “not”.
$a \gg b$	Arithmetic right shift of a two’s complement integer representation of a by b binary digits. This function is defined only for positive integer values of b .
$a \ll b$	Arithmetic left shift of a two’s complement integer representation of a by b binary digits. This function is defined only for positive integer values of b .

4.6 Assignment operators

The following assignment operators are defined as Table 5.

Table 5—Definitions of assignment operators

Assignment operator	Definition
=	Assignment operator.
++	Increment, i.e., $x++$ is equivalent to $x = x + 1$; when used in an array index, evaluates to the value of the variable prior to the increment operation.
--	Decrement, i.e., $x--$ is equivalent to $x = x - 1$; when used in an array index, evaluates to the value of the variable prior to the decrement operation.
+=	Increment by amount specified, e.g., $x += 3$ is equivalent to $x = x + 3$, and $x += (-3)$ is equivalent to $x = x + (-3)$.
-=	Decrement by amount specified, e.g., $x -= 3$ is equivalent to $x = x - 3$, and $x -= (-3)$ is equivalent to $x = x - (-3)$.

4.7 Tensor operators

4.7.1 Tensor operators defined in IEEE Std 2941-2021

The tensor operators defined in IEEE Std 2941-2021 are shown in Table 6.

Table 6—Tensor operators defined in IEEE Std 2941-2021

Assignment operator	Definition
abs	The definition of abs shall follow Table 12 from IEEE Std 2941-2021.
add	The definition of add shall follow Table 14 from IEEE Std 2941-2021.
assign	The definition of assign shall follow Table 22 from IEEE Std 2941-2021.
cast	The definition of cast shall follow Table 26 from IEEE Std 2941-2021.
ceil	The definition of ceil shall follow Table 27 from IEEE Std 2941-2021.
concat	The definition of concat shall follow Table 28 from IEEE Std 2941-2021.
crop	The definition of crop shall follow Table 38 from IEEE Std 2941-2021.
depth_to_space	The definition of depth_to_space shall follow Table 42 from IEEE Std 2941-2021.
div	The definition of div shall follow Table 44 from IEEE Std 2941-2021.
mod	The definition of mod shall follow Table 49 from IEEE Std 2941-2021.
exp	The definition of exp shall follow Table 52 from IEEE Std 2941-2021.
layer_norm	The definition of layer_norm shall follow Table 66 from IEEE Std 2941-2021.
log	The definition of log shall follow Table 68 from IEEE Std 2941-2021.
lpnorm	The definition of lpnorm shall follow Table 70 from IEEE Std 2941-2021.
mat_mul	The definition of mat_mul shall follow Table 72 from IEEE Std 2941-2021.
max_pool	The definition of max_pool shall follow Table 73 from IEEE Std 2941-2021.
mul	The definition of mul shall follow Table 75 from IEEE Std 2941-2021.
negative	The definition of negative shall follow Table 76 from IEEE Std 2941-2021.
pad	The definition of pad shall follow Table 80 from IEEE Std 2941-2021.
range	The definition of range shall follow Table 88 from IEEE Std 2941-2021.
reduce_sum	The definition of reduce_sum shall follow Table 95 from IEEE Std 2941-2021.
relu	The definition of relu shall follow Table 97 from IEEE Std 2941-2021.
reshape	The definition of reshape shall follow Table 99 from IEEE Std 2941-2021.
round	The definition of round shall follow Table 102 from IEEE Std 2941-2021.
space_to_depth	The definition of space_to_depth shall follow Table 110 from IEEE Std 2941-2021.
sub	The definition of sub shall follow Table 114 from IEEE Std 2941-2021.
switch_case	The definition of switch_case shall follow Table 115 from IEEE Std 2941-2021.
topk	The definition of topk shall follow Table 116 from IEEE Std 2941-2021.
transpose	The definition of transpose shall follow Table 118 from IEEE Std 2941-2021.
zeros	The definition of zeros shall follow Table 125 from IEEE Std 2941-2021.
leaky_relu	The definition of leaky_relu shall follow Table 136 from IEEE Std 2941-2021.
sigmoid	The definition of sigmoid shall follow Table 139 from IEEE Std 2941-2021.
softmax	The definition of softmax shall follow Table 141 from IEEE Std 2941-2021.

Assignment operator	Definition
softplus	The definition of softplus shall follow Table 142 from IEEE Std 2941-2021.
tanh	The definition of tanh shall follow Table 144 from IEEE Std 2941-2021.
resize_bilinear	The definition of resize_bilinear shall follow Table 148 from IEEE Std 2941-2021.
flip	The definition of flip shall follow Table 259 from IEEE Std 2941-2021.
rotate	The definition of rotate shall follow Table 259 from IEEE Std 2941-2021.

4.7.2 Additional tensor operators

The newly defined tensor operators are defined in Table 7.

Table 7—Newly defined operators

Assignment operator	Definition
conv2d	The definition of conv2d shall follow Table 8. Based on the definition, the padding attribute of this operation can be omitted when the stride attribute is 1, and the default is to pad the input so that the resolution remains unchanged after convolution. When the stride attribute is not 1, the padding attribute cannot be omitted.
conv2d_transpose	The definition of conv2d_transpose shall follow Table 9.
gelu	The definition of gelu shall follow Table 10.
lut_query	The definition of lut_query shall follow Table 11.
matmul	The definition of matmul shall follow Table 12.
multiply	The definition of multiply shall follow Table 13.
resize_bicubic	The definition of resize_bicubic shall follow Table 14.
resize_bilinear	The definition of resize_bicubic shall follow Table 15.
resize_lanczos	The definition of resize_lanczos shall follow Table 16.
average_pool	The definition of resize_lanczos shall follow Table 17.

4.7.2.1 conv2d operator

The conv2d operator is defined in Table 8.

Table 8—conv2d operator

Operator	Description	Field	Keyword	Definition	Data type
conv2d	Construct a two-dimensional convolutional layer, and calculate the output feature map according to the input, filter parameters, step size, filling, and other parameters	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
			W	Weight tensor	float 16 float 32 float 64 int 8 int 16
			B	Bias tensor	float 16 float 32 float 64 int 8 int 16
		Output	Y	Output tensor	float 16 float 32 float 64 int 16 int 32
		Attributes	num_channels	The number of channels of the image	int
			num_filters	Number of filters	int
			filter_size	Filter size	list of ints int
			stride	Optional, step size	list of ints int
			padding	Optional, fill size	list of ints int
			dilation	Optional, expansion coefficient size	list of ints int
			groups	Optional, the number of groups of the two dimensional convolutional layer	int
			dtype	Optional, output tensor data type	string

4.7.2.2 conv2d_transpose operator

The conv2d_transpose operator is defined in Table 9.

Table 9—conv2d_transpose operator

Operator	Description	Field	Keyword	Definition	Data type
conv2d_transpose	Construct a two-dimensional convolution transposed layer in the neural network, and output the feature map according to the input, filter parameters, step size, and other parameters	Input	X	Input tensor	float 16 float 32 float 64 int 16 int 32
			W	Weight tensor	float 16 float 32 float 64 int 16 int 32
			B	Bias tensor	float 16 float 32 float 64 int 16 int 32
		Output	Y	Output tensor	float 16 float 32 float 64 int 8 int 16
		Attributes	num_channels	The number of channels of the image	int
			num_filters	Number of filters	int
			filter_size	Filter size	int list of ints
			output_size	Optional, the size of the output feature map	int list of ints
			padding	Optional, fill size	int list of ints
			stride	Optional, step size	int list of ints
			dilation	Optional, expansion ratio	int list of ints
			groups	Optional, the number of groups of the two-dimensional convolutional layer	int
			dtype	Optional, input and output tensor format	string

4.7.2.3 gelu operator

The gelu operator is defined in Table 10.

Table 10—gelu operator

Operator	Description	Field	Keyword	Definition	Data type
gelu	Apply element-wise the function $\text{gelu}(x) = x * \phi(x)$, where $\phi(x)$ is the Cumulative Distribution Function for Gaussian Distribution	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
		Output	Y	Output tensor	float 16 float 32 float 64 int 8 int 16

4.7.2.4 lut_query operator

The lut_query operator is defined in Table 11.

Table 11—lut_query operator

Operator	Description	Field	Keyword	Definition	Data type
lut_query	Compute the activation value from the table according to the input.	Input	key	The index of look-up-table	int 16
			Table	Look-up-table	int 16
		Output	Y	Output tensor	int 16

4.7.2.5 matmul operator

The matmul operator is defined in Table 12.

Table 12—matmul operator

Operator	Description	Field	Keyword	Definition	Data type
matmul	Compute the value based on the matrix multiply according the input.	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
			Y	Input tensor	float 16 float 32 float 64 int 8 int 16
		Output	Z	Output tensor	float 16 float 32 float 64 int 8 int 16

4.7.2.6 multiply operator

The multiply operator is defined in Table 13.

Table 13—multiply operator

Operator	Description	Field	Keyword	Definition	Data type
multiply	Apply element-wise multiplication function multiply(x,y) = x * y, where x, y are the input tensors which have the same shape.	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
		Input	Y	Input tensor	float 16 float 32 float 64 int 8 int 16
		Output	Z	Output tensor	float 16 float 32 float 64 int 8 int 16

4.7.2.7 resize_bicubic operator

The resize_bicubic operator is defined in Table 14.

Table 14—resize_bicubic operator

Operator	Description	Field	Keyword	Definition	Data type
resize_bicubic	Use bicubic interpolation to resize the input image data to the specified size	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
			shape	Enter the size tensor after the resize	int32 int64
		Output	Z	Output tensor	float 16 float 32 float 64 int 8 int 16
		Attributes	Align_corners	If True, the corner pixels of the input and output tensors are aligned, and thus preserving the values at those pixels	bool

4.7.2.8 resize_bilinear operator

The resize_bilinear is defined in Table 15.

Table 15—resize_bilinear operator

Operator	Description	Field	Keyword	Definition	Data type
resize_bilinear	Use bilinear interpolation to resize the input image data to the specified size	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
			shape	Enter the size tensor after the resize	int32 int64
		Output	Z	Output tensor	float 16 float 32 float 64 int 8 int 16
		Attributes	Align_corners	If True, the corner pixels of the input and output tensors are aligned, and thus preserving the values at those pixels	bool

4.7.2.9 resize_lanczos operator

The resize_lanczos operator is defined in Table 16.

Table 16—resize_lanczos operator

Operator	Description	Field	Keyword	Definition	Data type
resize_lanczos	Use lanczos interpolation to resize the input image data to the specified size	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
			shape	Enter the size tensor after the resize	int32 int64
		Output	Z	Output tensor	float 16 float 32 float 64 int 8 int 16

4.7.2.10 average_pool operator

The average_pool operator is defined in Table 17.

Table 17—average_pool operator

Operator	Description	Field	Keyword	Definition	Data type
average_pool	Apply average pooling on the input tensor according to kernel_shape, strides, and pad	Input	X	Input tensor	float 16 float 32 float 64 int 8 int 16
		Output	Y	Output tensor	float 16 float 32 float 64 int 8 int 16
		Attributes	auto_pad	Valid or Same, and can not be pads used with	string
			kernel_shape	Core size	list of ints
			pads	For the beginning and end, the padding of each dimension	list of ints
			strides	Step size for each dimension	list of ints
			data_format	Input and output data format	string

4.8 Mathematical functions

The definitions of mathematical functions are shown in Equation (1) through Equation (13).

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (1)$$

where

x independent variable x .

$$\text{Ceil}(x) = \lceil x \rceil \quad (2)$$

where

x independent variable x .

$$\text{Floor}(x) = \lfloor x \rfloor \quad (3)$$

where

x independent variable x .

$$\text{Clip1}(x) = \text{Clip3}(0, 2^{\text{BitDepth}} - 1, x) \quad (4)$$

where

x independent variable x .
 BitDepth precision of coded sample.

$$\text{Clip3}(i, j, x) = \begin{cases} i & ; x < i \\ j & ; x > j \\ x & ; \text{others} \end{cases} \quad (5)$$

where

x independent variable x .
 i lower bound.
 j upper bound.

$$\text{Median}(x, y, z) = x + y + z - \text{Min}(x, \text{Min}(y, z)) - \text{Max}(x, \text{Max}(y, z)) \quad (6)$$

where

x independent variable x .
 y independent variable y .
 z independent variable z .

$$\text{Min}(x, y) = \begin{cases} x & ; x \leq y \\ y & ; x > y \end{cases} \quad (7)$$

where

x independent variable x .
 y independent variable y .

$$\text{Max}(x, y) = \begin{cases} x & ; x \geq y \\ y & ; x < y \end{cases} \quad (8)$$

where

x independent variable x .
 y independent variable y .

$$\text{Sign}(x) = \begin{cases} 1 & ; x \geq 0 \\ -1 & ; x < 0 \end{cases} \quad (9)$$

where

x independent variable x .

$$\text{Log}(x) = \log_2 x \quad (10)$$

where

x independent variable x .

$$\text{Ln}(x) = \log_e x \quad (11)$$

where

x independent variable x .
 e base of the natural logarithm, with a value of 2.718281828...

$$\text{Rounding}(x, s) = \text{Sign}(x) \times ((\text{Abs}(x) + (1 \ll (s - 1))) \gg s) \quad s \geq 1 \quad (12)$$

where

x independent variable x .
 s independent variable s .

$$\text{Exchange}(x, y) = (\text{Swap the value of } x \text{ and } y) \quad (13)$$

where

x independent variable x .
 y independent variable y .

4.9 Structure relation

The structure relation is defined as Table 18.

Table 18—Structure relation

Structure relation	Definition
->	For example, $a \rightarrow b$ means that a is a structure and b is a member variable of a .

4.10 Description methods of bitstream syntax, parsing process, and decoding process

4.10.1 Description method of bitstream syntax

The description method of bitstream syntax is similar to C language. Syntax elements in the bitstream are represented in bold type. Each syntax element is described by its name (all lowercase letters with underscored characters), syntax, and semantics. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases, the syntax tables may use the values of other variables derived from syntax elements. Such variables appear in the syntax tables, or text, named by a mixture of lowercase letters and uppercase letters without underscoring. Variables starting with an uppercase letter are used for the decoding of the current and related syntax structures. Variables starting with an uppercase letter may also be used for the decoding of subsequent syntax structures. Variables starting with a lowercase letter are only used within the subclause from which they are derived.

The relationship between the mnemonic symbols of the syntax element value, the mnemonic symbols of the variable value and their values is described in the text. In some cases, the two are equivalent. A mnemonic symbol is represented by one or more groups of letters separated by underscores; each group starts with an uppercase letter and may contain more uppercase letters.

When the length of the bit string is an integral multiple of 4, it can be represented by a hexadecimal notation. The prefix of hexadecimal notation is “0x”, for example, “0x1a” represents a bit string “0001 1010”.

In the conditional statement, 0 means FALSE and nonzero means TRUE.

The syntax tables describe all the supersets of the bitstream syntax that conform to this standard, and the additional syntax restrictions are described in the relevant clauses.

Table 19 shows an example of pseudo-code describing syntax. It always means to read one data element from the bitstream whenever syntax elements are there.

Table 19—Pseudo-code of syntax description

Pseudo-code	Descriptor
/* A statement is a descriptor of a syntax element, or indicates the existence, type, and value of a syntax element, two examples are given below. */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
...	
}	
/*A "while" structure tests whether the condition is TRUE. If TRUE, the statement is executed repeatedly until the condition is not TRUE. */	
while (condition)	
statement	
/*A "do ... while" structure executes the statement once, and then tests the condition. It repeatedly executes the statement while the condition remains true. */	
do	
statement	
while (condition)	
/*An "if ... else" structure tests the condition first. If it is true, the primary statement is executed. Otherwise, the alternative statement is executed. If the alternative statement is unnecessary to be executed, the "else" part and corresponding alternative statement can be omitted. */	
if (condition)	
primary statement	

Pseudo-code	Descriptor
else	
alternative statement	
/*A “for” structure executes the initial statement first, then tests the condition. If it is TRUE, the primary and subsequent statements are executed repeatedly until the condition is not TRUE.*/	
for (initial statement; condition; subsequent statement)	
primary statement	
/*A “break” statement is used in do-while, while and for loop body, which can make the current loop body immediately terminate the loop.*/	
break	

The parsing process and decoding process are described in text and pseudo-code similar to C programming language.

4.10.2 Functions

4.10.2.1 Overview

Functions in this clause are used to describe syntax. These functions assume that there is a bitstream pointer in the decoder. The pointer indicates the position of the next bit to be read from the bitstream. A function name and the parameters in the left and right parentheses make up a function; however, a function may also exist without parameters.

4.10.2.2 byte_aligned()

If the current position of bitstream is byte-aligned, it returns TRUE. Otherwise, it returns FALSE.

4.10.2.3 next_bits(n)

next_bits(n) returns the next n bits in the bitstream with MSB first, without changing the bitstream pointer. If the remaining number of bits is less than n, it returns 0.

4.10.2.4 byte_aligned_next_bits(n)

If the current position in bitstream is not byte-aligned, byte_aligned_next_bits(n) returns n bits with MSB first starting from the next byte after the current bitstream pointer position. The bitstream pointer is not changed. If the current position of bitstream is byte-aligned, byte_aligned_next_bits(n) returns next n bits in the bitstream with MSB first, without changing the bitstream pointer. If the remaining number of bits is less than n, it returns 0.

4.10.2.5 next_start_code()

next_start_code() seeks the next start code in bitstream, with the bitstream pointer pointing to the first bit of the start code prefix. The definition of the function is shown in Table 20.

Table 20—Definition of next_start_code function

Definition	Descriptor
next_start_code() × {	
stuffing_bit	“1”
while (! byte_aligned())	
stuffing_bit	“0”
while (next_bits(24) != “0000 0000 0000 0000 0000 0001”)	
stuffing_byte	“00000000”
}	

stuffing_byte shall be present in bitstream after the picture header and before the start code of the first patch.

4.10.2.6 is_end_of_patch()

is_end_of_patch() checks whether the patch is end in the bitstream, if the patch is end in the bitstream, it returns TRUE, otherwise it returns FALSE. This function does not modify the bitstream pointer. The definition of the function is shown in Table 21.

Table 21—Definition of is_end_of_patch function

Definition	Descriptor
is_end_of_patch() {	
if (byte_aligned()) {	
if (next_bits(32) == 0x80000001)	
return TRUE /* End of patch */	
}	
else {	
if ((byte_aligned_next_bits(24) == 0x000001) && is_stuffing_pattern())	
return TRUE /* End of patch */	
}	
return FALSE;	
}	

4.10.2.7 is_stuffing_pattern()

is_stuffing_pattern() checks whether the remaining bits of the current byte or the next byte at byte-aligned position are stuffing bits at the end of patch. If yes, it returns TRUE; otherwise it returns FALSE. This function does not modify the bitstream pointer. The definition of the function is shown in Table 22.

Table 22—Definition of is_stuffing_pattern function

Definition	Descriptor
is_stuffing_pattern() {	
if (next_bits(8-n) == (1<<(7-n))) /* n = 0 to 7, is the position offset of the bitstream pointer at current byte; if n is 0, the bitstream pointer points to the most significant bit of current byte*/	
return TRUE	
Else	
return FALSE;	
}	

4.10.2.8 read_bits(n)

read_bits(n) returns the next n bits in the bitstream with MSB first and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) returns 0 and does not advance the bitstream pointer.

The function is also used to describe the parsing process and decoding process.

4.10.2.9 lut_query(key, table)

lut_query(key, table) returns the value from the table according to the input key index.

4.10.2.10 qconv2d(X, W, B, num_channels, num_filters, filter_size, stride, padding, dilation, groups)

qconv2d(X, W, B, num_channels, num_filters, filter_size, stride, padding, dilation, groups) constructs a two-dimensional, quantized convolutional layer, and it calculates the output feature map according to the input, filter parameters, step size, filling, and other parameters. Specifically, the function is defined as follows:

$$r(w) = \text{round}(\max(|w|, \dim = 1) \times 2^{n-1}) / 2^{n-1}$$

$$Q(w; n) = \text{round}\left(\frac{w}{r(w)} \times 2^{n-1}\right)$$

$$\begin{aligned} r_b &= r(B) \\ r_w &= r(W) \\ r_x &= r(X) \\ W_q &= Q(W; n) \\ B_q &= \text{round}\left(\frac{2^{n-1} \times r_b}{r_w \times r_x} \times Q(B; n)\right) \\ X_q &= Q(X; n) \end{aligned}$$

4.10.2.11 qconv2d_transpose(X, W, B, num_channels, num_filters, filter_size, stride, padding, dilation, groups)

qconv2d_transpose(X, W, B, num_channels, num_filters, filter_size, stride, padding, dilation, groups) constructs a two-dimensional, quantized, transposed convolutional layer, and it calculates the output feature map according to the input, filter parameters, step size, filling, and other parameters. Specifically, the function is defined as follows:

$$r(w) = \text{round}(\max(|w|, \dim = 1) \times 2^{n-1}) / 2^{n-1}$$

$$Q(w; n) = \text{round}\left(\frac{w}{r(w)} \times 2^{n-1}\right)$$

$$\begin{aligned} r_b &= r(B) \\ r_w &= r(W) \\ r_x &= r(X) \\ W_q &= Q(W; n) \\ B_q &= \text{round}\left(\frac{2^{n-1} \times r_b}{r_w \times r_x} \times Q(B; n)\right) \\ X_q &= Q(X; n) \end{aligned}$$

$$= \text{conv2d_transpose}(X_q, W_q, B_q, \text{num_channels}, \text{num_filters}, \text{filter_size}, \text{stride}, \text{padding}, \text{dilation}, \text{groups})$$

$$Y = Y_{int} \times \frac{r_w \times r_x}{2^{2(n-1)}}$$

4.10.2.12 bounded_leaky_ReLU

bounded_leaky_ReLU(X , α , $abit$, $xbit$) conducted the integerized leaky ReLU activation function. The function of the bounded leaky ReLU is shown as follows:

$$\text{bounded_leaky_ReLU}(X, \alpha, abit, xbit) = \begin{cases} \text{Clip3}(-2 \ll 15, 2 \ll 15 - 1, \text{round}(X \ll xbit) \ll abit) \gg (abit + xbit) & X \geq 0 \\ \text{Clip3}(-2 \ll 15, 2 \ll 15 - 1, \text{round}(X \ll xbit) \times \text{round}(\alpha \ll abit) \gg (abit + xbit)) & X < 0 \end{cases}$$

where α , $abit$, and $xbit$ are the parameters stored in the weights.

4.10.2.13 qsoftmax

qsoftmax(x) is the quantized *softmax* function, defined as follows:

$$qsoftmax(x_i) = \begin{cases} \frac{LUT_{exp}(\text{round}(\tilde{x}_i \times s_{index}))}{\sum_j LUT_{exp}(\tilde{x}_j \times s_{index})}, & -11 \leq \tilde{x}_i \leq 0 \\ softmax(-11), & \tilde{x}_i < -11 \end{cases}$$

where

$$\tilde{x}_i = e^{x_i - \max(x)}$$

$$LUT_{exp}(x) = \text{round}\left(\exp\left(\frac{x}{s_{index}}\right) \times s_{value}\right), x \in \mathbb{Z} \text{ and } x \sim U(a, b)$$

s_{index} and s_{value} are scaling factors of input and output.

4.10.3 Descriptors

The descriptors below specify the parsing process of syntax elements, see Table 23.

Table 23—Descriptors

Descriptor	Description
ae(v)	Advanced entropy-coded syntax element.
b(8)	A byte with arbitrary value. The parsing process for this descriptor is specified by the returned value of read_bits(8).
f(n)	Consecutive n bits taking designated value. The parsing process is specified by the returned value of read_bits(n).
i(n)	Signed integer with n bits. In syntax table, if n is 'v', the number of bits is determined by other syntax elements. The parsing process is specified by the return value of read_bits(n), the returned value is interpreted as two's complement with MSB first.
r(n)	Consecutive n bits '0'. The parsing process is specified by the returned value of read_bits(n).
se(v)	Signed integer Exp-Golomb-coded syntax element.
u(n)	Unsigned integer with n bits. In syntax table, if n is 'v', the number of bits is determined by other syntax elements. The parsing process is specified by the returned value of read_bits(n), the returned value is interpreted as binary system with MSB first.
ue(v)	Unsigned integer Exp-Golomb-coded syntax element.

4.10.4 Reserved, forbidden and marker

In the bitstream syntax defined by this standard, the value of some syntax elements is marked as “reserved” or “forbidden.”

The term *reserved* specifies some particular syntax element values for future extensions of this standard. These values shall not be used in bitstreams conforming to this standard.

The term *forbidden* specifies some particular syntax element values that shall not be used in bitstreams conforming to this standard.

marker_bit specifies a bit with value 1.

reserved_bits in bitstream indicate that some syntax elements are reserved for future extensions of this standard. The decode processing shall ignore these bits. In the series of consecutive *reserved_bits* bytes, there shall not be a string of more than 21 consecutive “0” starting from any byte-aligned position.

5. Structure of coded bitstream

5.1 Picture

5.1.1 Picture format

5.1.1.1 RGB 4:4:4 format

In this format, the R, G, and B matrices are the same size in both horizontal and vertical dimensions.

The location of the R sample, the G sample, and the B sample is shown in Figure 2.

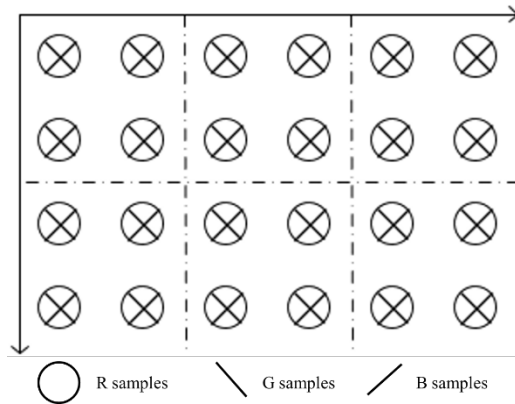


Figure 2—Location of R, G, and B samples in RGB 4:4:4 format

5.1.1.2 YUV 4:0:0 format

In this format, there is only a Y matrix.

The location of the luma sample is shown in Figure 3.

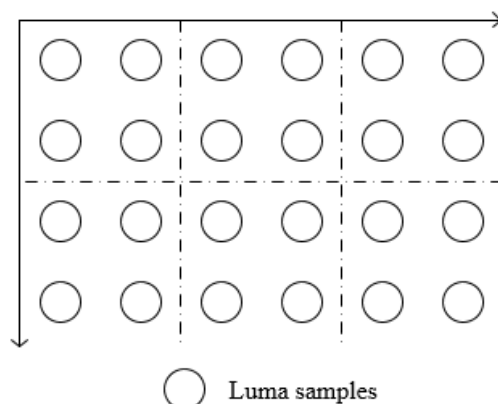


Figure 3—Location of luma sample in 4:0:0 format

5.1.1.3 YUV 4:2:0 format

In this format, the Cb and Cr matrices are half the size of the Y matrix in both horizontal and vertical dimensions, and the Y matrix shall have an even number of rows and samples. In addition, if the coded data of two fields of the picture appear in sequential order, the number of rows of the Y matrix should be divided exactly by 4. The location of luma sample and chroma sample is shown in Figure 4.

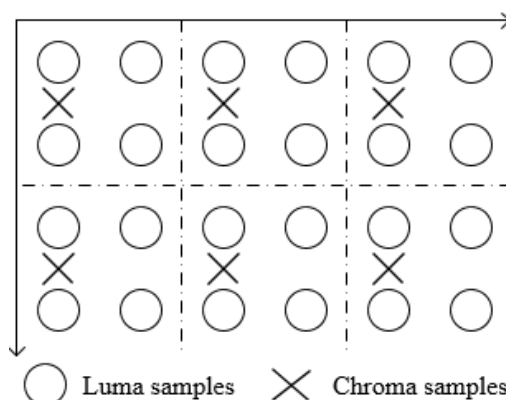


Figure 4—Location of luma and chroma samples in 4:2:0 format

5.1.1.4 YUV 4:2:2 format

In this format, the Cb and Cr matrices are half the size of the Y matrix in horizontal dimension and the same size of the Y matrix in vertical dimension. The number of samples per row of the Y matrix should be an even number. If the coded data of two fields of the picture appear in sequential order, the number of rows of the Y matrix should also be an even number.

The location of luma sample and chroma sample is shown in Figure 5.

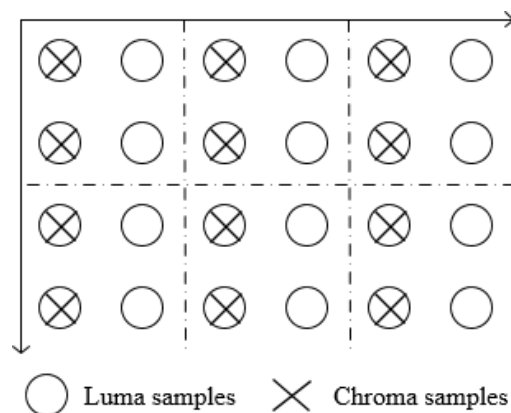


Figure 5—Location of luma and chroma samples in 4:2:2 format

5.1.1.5 YUV 4:4:4 format

In this format, the Cb and Cr matrices are the same size of the Y matrix in both horizontal and vertical dimensions. If the coded data of two fields of the picture appear in sequential order, the number of rows of the Y matrix should be an even number.

The location of luma sample and chroma sample is shown in Figure 6.

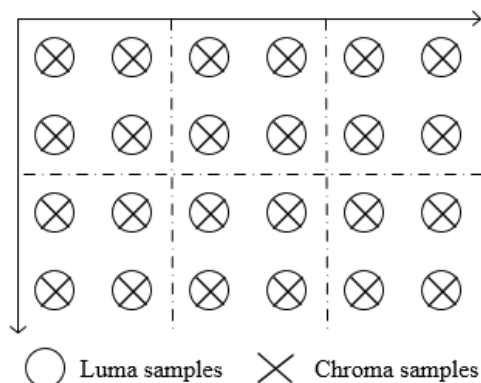


Figure 6—Location of luma and chroma samples in 4:4:4 format

5.2 Slice

A slice is a rectangular area in a picture, which is the basic unit of coding. At the same time, during the encoding process, the `parameter_set_id` can be rewritten by modifying the corresponding parameters. See Figure 7 for the slice structure.

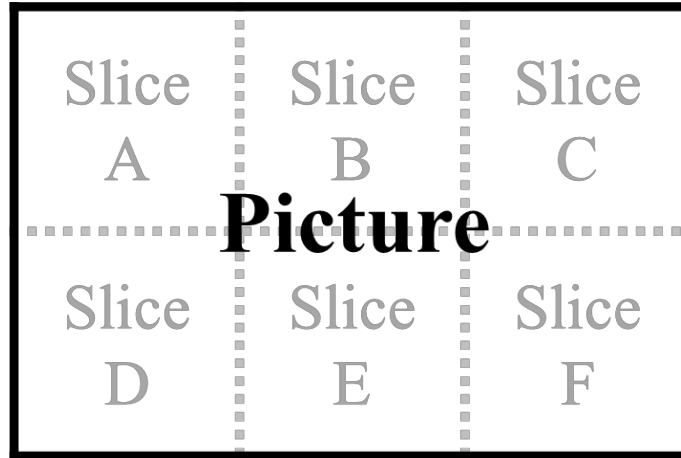


Figure 7—Slice structure

6. Bitstream syntax and semantics

6.1 Syntax description

6.1.1 Start code

A picture begins with the picture start code followed by a string of coded picture data.

start code prefix: A unique sequence of three bytes equal to 0x000001 embedded in the byte stream as a prefix to each byte stream.

NOTE—The location of a start code prefix shall be used by a decoder to identify the beginning of a coding unit and the end of a previous byte stream.

start code: A unique codeword of four bytes embedded in the bitstream. A start code consists of a start code prefix and a start code value. All start codes shall be byte aligned.

NOTE—Start code is a special bit string. In bitstreams conforming to this standard, the same bit string should not occur anywhere except in start codes. Emulation of start code prefixes is prevented within coding units by the inclusion of emulation prevention bytes.

The start code value is an 8-bit integer that identifies the type of start code, as shown in Table 24.

Table 24—Start code value

Start code type	Start code value
picture_start_code	0x00
slice_start_code	0x01
reserved	0x02~0xFF

6.1.2 Definition of picture

Definition of picture is shown in Table 25.

Table 25—Definition of picture

Definition of picture	Descriptor
picture() {	
picture_start_code	f(32)
picture_header()	
do {	
slice()	
} while (nextbits(32) == slice_start_code)	
}	

6.1.2.1 Definition of picture header

Definition of picture header is shown in Table 26.

Table 26—Definition of picture header

Definition of picture header	Descriptor
picture_header() {	
profile_id	u(3)
level_id	u(6)
model_id	u(3)
parameter_set_id	u(7)
quality_id	u(5)
picture_size_h	u(16)
picture_size_w	u(16)
output_bit_depth	u(3)
picture_format	u(3)
log2_slice_size_h_minus6	u(4)
log2_slice_size_w_minus6	u(4)
reserved_2_bit	u(2)
coding_tool(profile_id, model_id)	
byte_alignment()	
}	

6.1.2.2 Definition of coding tool

Definition of coding tool is shown in Table 27.

Table 27—Definition of coding tool

Definition of coding tool	Descriptor
<code>coding_tool(model_id) {</code>	
<code>if (profile_id == 0) {</code>	
<code>coding_tool_0()</code>	
<code>}</code>	
<code>if (profile_id == 1) {</code>	
<code>coding_tool_1()</code>	
<code>}</code>	
<code>if (profile_id == 2) {</code>	
<code>coding_tool_2(model_id)</code>	
<code>}</code>	
<code>}</code>	

6.1.2.2.1 Definition of the function “coding_tool_0()”

Definition of the function “coding_tool_0()” is shown in Table 28.

Table 28—Definition of coding_tool_0

Definition of coding_tool_0	Descriptor
<code>coding_tool_0() {</code>	
<code>resized_size_h</code>	u(16)
<code>resized_size_w</code>	u(16)
<code>latent_code_shape_h</code>	u(16)
<code>latent_code_shape_w</code>	u(16)
<code>mask_scale()</code>	
<code>num_first_level_tile</code>	u(8)
<code>num_second_level_tile</code>	u(8)
<code>adaptive_offset()</code>	
<code>output_bit_shift</code>	u(4)
<code>double_precision_processing_flag</code>	u(1)
<code>deterministic_processing_flag</code>	u(1)
<code>fast_resize_flag</code>	u(1)
<code>byte_alignment()</code>	
<code>}</code>	

Definitions of the functions “mask_scale()” and “adaptive_offset()” are shown in Table 29 and Table 30, respectively.

Table 29—Definition of adaptive masking and scaling syntax

Definition of mask_scale	Descriptor
mask_scale() {	
num_adaptive_quant_params	u(8)
num_block_based_skip_params	u(8)
num_latent_post_process_params	u(8)
for (idx = 0; idx < num_adaptive_quant_params + num_block_based_skip_params + num_latent_post_process_params; idx++) {	
filter_list[idx][0]	u(4)
filter_list[idx][1]	u(1)
filter_list[idx][1] += 1	
filter_list[idx][2]	u(1)
filter_list[idx][3]	u(1)
filter_list[idx][4]	u(1)
if (filter_list[idx][1] > 1)	
filter_list[idx][1]	u(8)
if (filter_list[idx][3] == 1)	
filter_list[idx][5]	u(8)
else	
filter_list[idx][5]	u(16)
if (filter_list[idx][4] == 1)	
filter_list[idx][6]	u(8)
else	
filter_list[idx][6]	u(16)
if (filter_list[idx][0] == 5)	
if (filter_list[idx][3] == 1)	
filter_list[idx][7]	u(8)
else	
filter_list[idx][7]	u(16)
if (filter_list[idx][0] == 4){	
filter_list[idx][8]	u(8)
if (filter_list[idx][8] == 1){	
channel_num	u(8)
filter_list[idx][9][channel_num] = True	
}	
}	

Table 30—Definition of adaptive offset syntax

Definition of adaptive_offset	Descriptor
adaptive_offset() {	
adaptive_offset_enabled_flag	u(1)
reserved_7_bit	u(7)
if (adaptive_offset_enabled_flag) {	
num_horizontal_split	u(8)
num_vertical_split	u(8)
offset_precision	u(8)
for (idx = 0; idx < 192; idx++)	
offsets_signalled[idx]	u(1)
for (idx = 0; idx < 192; idx++)	
for (idx2 = 0; idx2 < num_horizontal_split; idx2++)	
for (idx3 = 0; idx3 < num_vertical_split; idx3++){	
if (offsets_signalled[idx]){	
offsets_list[idx][idx2][idx3][0]	u(7)
offsets_list[idx][idx2][idx3][1]	u(1)
}	
else{	
offsets_list[idx][idx2][idx3][0] = 0	
offsets_list[idx][idx2][idx3][1] = 0	
}	
}	
}	
}	

6.1.2.2.2 Definition of the function “coding_tool_1()”

Definition of the function “coding_tool_1()” is shown in Table 31.

Table 31—Definition of coding_tool_1

Description of coding_tool_1 function	Descriptor
coding_tool_1() {	
geo_flag	u(1)
filtering_flag	u(1)
enhance_flag	u(1)
reserved_5_bits	u(5)
if (filtering_flag) {	
filtering_CTU	u(16)
filtering_info(enhance_flag)	
}	
}	

Definition of the function “filtering_info()” is shown in Table 32.

Table 32—Definition of filtering_info

Description of filtering_info function	Descriptor
filtering_info(enhance_flag) {	
if (enhance_flag) {	
for (i=0; i<84; i++) {	
flt_block_params[i]	u(16)
}	
}	
}	

6.1.2.2.3 Definition of the function “coding_tool_2()”

Definition of the function “coding_tool_2(model_id)” is shown in Table 33.

Table 33—Definition of coding_tool_2

Definition of coding_tool_2	Descriptor
coding_tool_2(model_id){	
channel_num	u(3)
resolution_level	u(3)
if (model_id != 2) {	
filtering_model_id	u(3)
}	
for (i=0; i<channel_num; i++) {	
for (j=0; j<3×resolution_level+1; j++) {	
positive_min_v[i][j]	u(15)
positive_max_v[i][j]	u(15)
}	
}	
}	

6.1.3 Definition of slice

Definition of slice is shown in Table 34.

Table 34—Definition of slice

Definition of slice	Descriptor
slice () {	
slice_start_code	f(32)
slice_header()	
slice_data()	
byte_alignment()	
}	

6.1.3.1 Definition of slice header

Definition of slice header is shown in Table 35.

Table 35—Definition of slice header

Definition of slice header	Descriptor
slice_header(model_id) {	
slice_id	u(8)
slice_parameter_set_id_rewrite_flag	u(1)
slice_quality_id_rewrite_flag	u(1)
if (slice_parameter_set_id_rewrite_flag == 1) {	
slice_parameter_set_id	u(7)
reserverd_1_bit	u(1)
}	
if (slice_quality_id_rewrite_flag == 1) {	
slice_quality_id	u(5)
reserverd_3_bit	u(3)
}	
if (profile_id == 1) {	
min_main	i(16)
max_main	i(16)
min_hyper_1	i(16)
max_hyper_1	i(16)
min_hyper_2	i(16)
max_hyper_2	i(16)
file_size	u(32)
if (geo_flag) {	
geo_index	u(8)
}	
threshold	u(8)
}	
if (profile_id == 2) {	
log2_subband_block_height_minus4	u(3)
log2_subband_block_width_minus4	u(3)
if (model_id != 2) {	
no_estimated_delta_scale	u(1)
}	
}	
}	

6.1.3.2 Definition of slice data

Definition of slice data is shown in Table 36.

Table 36—Definition of slice data

Definition of slice data	Descriptor
slice_data() {	
get_parameters()	
if (profile_id == 0) {	
slice_data_0()	
}	
if (profile_id == 1) {	
slice_data_1()	
}	
if (profile_id == 2) {	
slice_data_2()	
}	
}	

The function “get_parameters()” is used to load the model parameters. Definition of “get_parameters()” is shown in Table 37.

Table 37—Definition of get_parameters

Definition of get_parameters	Descriptor
get_parameters(model_id, parameter_set_id, slice_parameter_set_id=-1, filtering_model_id=-1, filtering_parameter_set_id=-1){	
W = {}	
B = {}	
M = {}	
F = {}	
Alpha = {}	
parameter_set_id = slice_parameter_set_id != -1 ? slice_parameter_set_id : parameter_set_id	
lut_tables = load_lut()	
model_parameters = load(model_id, parameter_set_id)	
if (profile_id==0) {	
lut_tables[‘FactorizedModel’] = model_parameters.modules[‘FactorizedModel’]	
lut_tables[‘ScaleTable’] = model_parameters.modules[‘ScaleTable’]	
W[‘HyperScaleDecoder’], B[‘HyperScaleDecoder’], Alpha[‘HyperScale Decoder’] = model_parameters.modules[‘HyperScaleDecoder’]	
W[‘HyperDecoder’], B[‘HyperDecoder’], Alpha[‘HyperDecoder’] = model_parameters.modules[‘HyperDecoder’]	
W[‘ChannelSpaceShuffleContextModel’], B[‘ChannelSpaceShuffleContextModel’], Alpha[‘ChannelSpaceShuffleContextModel’] = model_parameters.modules[‘ChannelSpaceShuffleContextModel’]	
W[‘DecoderFirstPart’], B[‘DecoderFirstPart’], Alpha[‘DecoderFirstPart’] = model_parameters.modules[‘DecoderFirstPart’]	
W[‘DecoderSecondPart’], B[‘DecoderSecondPart’], Alpha[‘DecoderSecond Part’] = model_parameters.modules[‘DecoderSecondPart’]	

Definition of get_parameters	Descriptor
}	
if (profile_id == 1) {	
M['factorized_entropy_func'], F['factorized_entropy_func'], B['factorized_entropy_func'] = model_parameters.modules['Entropy_bottleneck']	
W['hyper_1_dec'], B['hyper_1_dec'] = model_parameters.modules['hyper_1_dec']	
W['hyper_2_dec'], B['hyper_2_dec'] = model_parameters.modules['hyper_2_dec']	
W['p'], B['p'] = model_parameters.modules['p']	
W['p_2'], B['p_2'] = model_parameters.modules['p_2']	
W['Y'], B['Y'] = model_parameters.modules['Y']	
W['Y_2'], B['Y_2'] = model_parameters.modules['Y_2']	
W['decoder'], B['decoder'] = model_parameters.modules['decoder']	
}	
if (profile_id == 2) {	
W['Net_delta_scale'] = model_parameters.modules['Net_delta_scale']	
W['CodingLL'], B['CodingLL'] = model_parameters.modules['CodingLL']	
W['CodingHL'] = []	
W['CodingLH'] = []	
W['CodingHH'] = []	
B['CodingHL'] = []	
B['CodingLH'] = []	
B['CodingHH'] = []	
for (i=0; i<resolution_level; i++) {	
W_tmp, B_tmp = model_parameters.modules['CodingHL'][i]	
W['CodingHL'].append(W_tmp)	
B['CodingHL'].append(B_tmp)	
W_tmp, B_tmp = model_parameters.modules['CodingLH'][i]	
W['CodingLH'].append(W_tmp)	
B['CodingLH'].append(B_tmp)	
W_tmp, B_tmp = model_parameters.modules['CodingHH'][i]	
W['CodingHH'].append(W_tmp)	
B['CodingHH'].append(B_tmp)	
}	
W['Transform'], B['Transform'] = model_parameters.modules['Transform']	
if (model_id != 2) {	
if (filtering_model_id != 0) {	
if (filtering_model_id == 1) {	
W['Filtering'], B['Filtering'] = model_parameters.modules['Restormer']	
}	
}	
}	
}	