(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 July 2024 (18.07.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/149396 A1

(51) International Patent Classification:
H04N 19/184 (2014.01)

(21) International Application Number:
PCT/CN2024/072173

(22) International Filing Date:
12 January 2024 (12.01.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PCT/CN2023/072218
14 January 2023 (14.01.2023) CN

(71) Applicants: **DOUYIN VISION CO., LTD.** [CN/CN]; Room B-0035, 2/F, No. 3 Building, No. 30, Shixing Road, Shijingshan District, Beijing 100041 (CN). **BYTEDANCE INC.** [US/US]; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US).

(72) Inventors: **LI, Yue**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **WANG, Ye-Kui**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **LI, Junru**; Jinritoutiao Post Office, China Satellite Communications Tower, No. 63, Zhichun Road, Haidian District, Beijing 100080 (CN). **LIN, Chaoyi**; Jinritoutiao Post Office, China Satellite Communications Tower, No. 63, Zhichun Road, Haidian District, Beijing 100080 (CN). **ZHANG, Kai**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Li**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **XU, Jizheng**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US).

(74) Agent: **SHIHUI PARTNERS**; 42/F, Tower C, Beijing Yintai Centre, No. 2 Jianguomenwai Avenue, Chaoyang District, Beijing 100022 (CN).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE,

(54) Title: METHOD, APPARATUS, AND MEDIUM FOR VIDEO PROCESSING

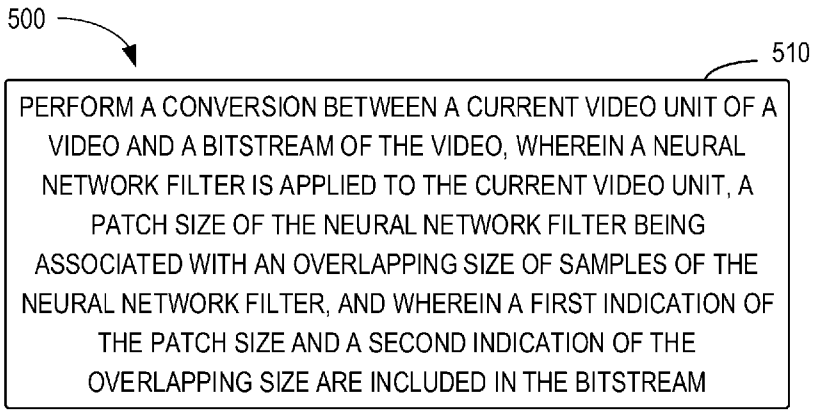

Fig. 5

(57) Abstract: Embodiments of the present disclosure provide a solution for video processing. A method for video processing is proposed. In the method, a conversion between a current video unit of a video and a bitstream of the video is performed. A neural network filter is applied to the current video unit. A patch size of the neural network filter is associated with an overlapping size of samples of the neural network filter. A first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

WO 2024/149396 A1

[Continued on next page]

SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

METHOD, APPARATUS, AND MEDIUM FOR VIDEO PROCESSING

FIELDS

[0001] Embodiments of the present disclosure relates generally to video processing techniques, and more particularly, to neural network filter for video processing.

BACKGROUND

[0002] In nowadays, digital video capabilities are being applied in various aspects of peoples' lives. Multiple types of video compression technologies, such as MPEG-2, MPEG-4, ITU-TH.263, ITU-TH.264/MPEG-4 Part 10 Advanced Video Coding (AVC), ITU-TH.265 high efficiency video coding (HEVC) standard, versatile video coding (VVC) standard, have been proposed for video encoding/decoding. However, coding efficiency of video coding techniques is generally expected to be further improved.

SUMMARY

[0003] Embodiments of the present disclosure provide a solution for video processing.

[0004] In a first aspect, a method for video processing is proposed. The method comprises: performing a conversion between a current video unit of a video and a bitstream of the video, wherein a neural network filter is applied to the current video unit, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream. The method in accordance with the second aspect of the present disclosure applies the neural network based on the patch size indicated by at least one indication, and thus can improve the coding efficiency and coding effectiveness of video coding.

[0005] In a second aspect, an apparatus for video processing is proposed. The apparatus comprises a processor and a non-transitory memory with instructions thereon. The instructions upon execution by the processor, cause the processor to perform a method in accordance with the first aspect of the present disclosure.

[0006] In a third aspect, a non-transitory computer-readable storage medium is proposed. The non-transitory computer-readable storage medium stores instructions that cause a processor to perform a method in accordance with the first aspect of the present

disclosure.

[0007] In a fourth aspect, another non-transitory computer-readable recording medium is proposed. The non-transitory computer-readable recording medium stores a bitstream of a video which is generated by a method performed by an apparatus for video processing. The method comprises: generating the bitstream, wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0008] In a fifth aspect, a method for storing a bitstream of a video is proposed. The method comprises: generating the bitstream; and storing the bitstream in a non-transitory computer-readable recording medium, wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0009] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Through the following detailed description with reference to the accompanying drawings, the above and other objectives, features, and advantages of example embodiments of the present disclosure will become more apparent. In the example embodiments of the present disclosure, the same reference numerals usually refer to the same components.

[0011] Fig. 1 illustrates a block diagram that illustrates an example video coding system, in accordance with some embodiments of the present disclosure;

[0012] Fig. 2 illustrates a block diagram that illustrates a first example video encoder, in accordance with some embodiments of the present disclosure;

[0013] Fig. 3 illustrates a block diagram that illustrates an example video decoder, in

accordance with some embodiments of the present disclosure;

[0014] Fig. 4 illustrates an illustration of luma data channels of nnpfc_inp_order_idc equal to 3 (informative);

[0015] Fig. 5 illustrates a flowchart of a method for video processing in accordance with embodiments of the present disclosure; and

[0016] Fig. 6 illustrates a block diagram of a computing device in which various embodiments of the present disclosure can be implemented.

[0017] Throughout the drawings, the same or similar reference numerals usually refer to the same or similar elements.

DETAILED DESCRIPTION

[0018] Principle of the present disclosure will now be described with reference to some embodiments. It is to be understood that these embodiments are described only for the purpose of illustration and help those skilled in the art to understand and implement the present disclosure, without suggesting any limitation as to the scope of the disclosure. The disclosure described herein can be implemented in various manners other than the ones described below.

[0019] In the following description and claims, unless defined otherwise, all technical and scientific terms used herein have the same meaning as commonly understood by one of ordinary skills in the art to which this disclosure belongs.

[0020] References in the present disclosure to "one embodiment," "an embodiment," "an example embodiment," and the like indicate that the embodiment described may include a particular feature, structure, or characteristic, but it is not necessary that every embodiment includes the particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an example embodiment, it is submitted that it is within the knowledge of one skilled in the art to affect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described.

[0021] It shall be understood that although the terms "first" and "second" etc. may be used herein to describe various elements, these elements should not be limited by these

terms. These terms are only used to distinguish one element from another. For example, a first element could be termed a second element, and similarly, a second element could be termed a first element, without departing from the scope of example embodiments. As used herein, the term "and/or" includes any and all combinations of one or more of the listed terms.

[0022] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting of example embodiments. As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises", "comprising", "has", "having", "includes" and/or "including", when used herein, specify the presence of stated features, elements, and/or components etc., but do not preclude the presence or addition of one or more other features, elements, components and/ or combinations thereof.

Example Environment

[0023] Fig. 1 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure. As shown, the video coding system 100 may include a source device 110 and a destination device 120. The source device 110 can be also referred to as a video encoding device, and the destination device 120 can be also referred to as a video decoding device. In operation, the source device 110 can be configured to generate encoded video data and the destination device 120 can be configured to decode the encoded video data generated by the source device 110. The source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[0024] The video source 112 may include a source such as a video capture device. Examples of the video capture device include, but are not limited to, an interface to receive video data from a video content provider, a computer graphics system for generating video data, and/or a combination thereof.

[0025] The video data may comprise one or more pictures. The video encoder 114 encodes the video data from the video source 112 to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded

representation of a picture. The associated data may include sequence parameter sets, picture parameter sets, and other syntax structures. The I/O interface 116 may include a modulator/demodulator and/or a transmitter. The encoded video data may be transmitted directly to destination device 120 via the I/O interface 116 through the network 130A. The encoded video data may also be stored onto a storage medium/server 130B for access by destination device 120.

[0026] The destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122. The I/O interface 126 may include a receiver and/or a modem. The I/O interface 126 may acquire encoded video data from the source device 110 or the storage medium/server 130B. The video decoder 124 may decode the encoded video data. The display device 122 may display the decoded video data to a user. The display device 122 may be integrated with the destination device 120, or may be external to the destination device 120 which is configured to interface with an external display device.

[0027] The video encoder 114 and the video decoder 124 may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[0028] Fig. 2 is a block diagram illustrating an example of a video encoder 200, which may be an example of the video encoder 114 in the system 100 illustrated in Fig. 1, in accordance with some embodiments of the present disclosure.

[0029] The video encoder 200 may be configured to implement any or all of the techniques of this disclosure. In the example of Fig. 2, the video encoder 200 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video encoder 200. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0030] In some embodiments, the video encoder 200 may include a partition unit 201, a prediction unit 202 which may include a mode select unit 203, a motion estimation unit 204, a motion compensation unit 205 and an intra-prediction unit 206, a residual generation unit 207, a transform unit 208, a quantization unit 209, an inverse quantization unit 210, an inverse transform unit 211, a reconstruction unit 212, a buffer 213, and an entropy encoding unit 214.

[0031] In other examples, the video encoder 200 may include more, fewer, or different functional components. In an example, the prediction unit 202 may include an intra block copy (IBC) unit. The IBC unit may perform prediction in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[0032] Furthermore, although some components, such as the motion estimation unit 204 and the motion compensation unit 205, may be integrated, but are represented in the example of Fig. 2 separately for purposes of explanation.

[0033] The partition unit 201 may partition a picture into one or more video blocks. The video encoder 200 and the video decoder 300 may support various video block sizes.

[0034] The mode select unit 203 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra-coded or inter-coded block to a residual generation unit 207 to generate residual block data and to a reconstruction unit 212 to reconstruct the encoded block for use as a reference picture. In some examples, the mode select unit 203 may select a combination of intra and inter prediction (CIIP) mode in which the prediction is based on an inter prediction signal and an intra prediction signal. The mode select unit 203 may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter-prediction.

[0035] To perform inter prediction on a current video block, the motion estimation unit 204 may generate motion information for the current video block by comparing one or more reference frames from buffer 213 to the current video block. The motion compensation unit 205 may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from the buffer 213 other than the picture associated with the current video block.

[0036] The motion estimation unit 204 and the motion compensation unit 205 may perform different operations for a current video block, for example, depending on whether the current video block is in an I-slice, a P-slice, or a B-slice. As used herein, an "I-slice" may refer to a portion of a picture composed of macroblocks, all of which are based upon macroblocks within the same picture. Further, as used herein, in some aspects, "P-slices" and "B-slices" may refer to portions of a picture composed of macroblocks that are not dependent on macroblocks in the same picture.

[0037] In some examples, the motion estimation unit 204 may perform uni-directional

prediction for the current video block, and the motion estimation unit 204 may search reference pictures of list 0 or list 1 for a reference video block for the current video block. The motion estimation unit 204 may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. The motion estimation unit 204 may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. The motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video block indicated by the motion information of the current video block.

[0038] Alternatively, in other examples, the motion estimation unit 204 may perform bi-directional prediction for the current video block. The motion estimation unit 204 may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in list 1 for another reference video block for the current video block. The motion estimation unit 204 may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. The motion estimation unit 204 may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. The motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.

[0039] In some examples, the motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder. Alternatively, in some embodiments, the motion estimation unit 204 may signal the motion information of the current video block with reference to the motion information of another video block. For example, the motion estimation unit 204 may determine that the motion information of the current video block is sufficiently similar to the motion information of a neighboring video block.

[0040] In one example, the motion estimation unit 204 may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder 300 that the current video block has the same motion information as the another video block.

[0041] In another example, the motion estimation unit 204 may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder 300 may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.

[0042] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder 200 include advanced motion vector prediction (AMVP) and merge mode signaling.

[0043] The intra prediction unit 206 may perform intra prediction on the current video block. When the intra prediction unit 206 performs intra prediction on the current video block, the intra prediction unit 206 may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.

[0044] The residual generation unit 207 may generate residual data for the current video block by subtracting (e.g., indicated by the minus sign) the predicted video block (s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.

[0045] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and the residual generation unit 207 may not perform the subtracting operation.

[0046] The transform processing unit 208 may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.

[0047] After the transform processing unit 208 generates a transform coefficient video block associated with the current video block, the quantization unit 209 may quantize the transform coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.

[0048] The inverse quantization unit 210 and the inverse transform unit 211 may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. The reconstruction unit 212 may add the reconstructed residual video block to corresponding samples from one or more predicted video blocks generated by the prediction unit 202 to produce a reconstructed video block associated with the current video block for storage in the buffer 213.

[0049] After the reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed to reduce video blocking artifacts in the video block.

[0050] The entropy encoding unit 214 may receive data from other functional components of the video encoder 200. When the entropy encoding unit 214 receives the data, the entropy encoding unit 214 may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.

[0051] Fig. 3 is a block diagram illustrating an example of a video decoder 300, which may be an example of the video decoder 124 in the system 100 illustrated in Fig. 1, in accordance with some embodiments of the present disclosure.

[0052] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of Fig. 3, the video decoder 300 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder 300. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0053] In the example of Fig. 3, the video decoder 300 includes an entropy decoding unit 301, a motion compensation unit 302, an intra prediction unit 303, an inverse quantization unit 304, an inverse transformation unit 305, and a reconstruction unit 306 and a buffer 307. The video decoder 300 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 200.

[0054] The entropy decoding unit 301 may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). The entropy decoding unit 301 may decode the entropy coded video data, and from the entropy decoded video data, the motion compensation unit 302 may determine motion information

including motion vectors, motion vector precision, reference picture list indexes, and other motion information. The motion compensation unit 302 may, for example, determine such information by performing the AMVP and merge mode. AMVP is used, including derivation of several most probable candidates based on data from adjacent PBs and the reference picture. Motion information typically includes the horizontal and vertical motion vector displacement values, one or two reference picture indices, and, in the case of prediction regions in B slices, an identification of which reference picture list is associated with each index. As used herein, in some aspects, a "merge mode" may refer to deriving the motion information from spatially or temporally neighboring blocks.

[0055] The motion compensation unit 302 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.

[0056] The motion compensation unit 302 may use the interpolation filters as used by the video encoder 200 during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. The motion compensation unit 302 may determine the interpolation filters used by the video encoder 200 according to the received syntax information and use the interpolation filters to produce predictive blocks.

[0057] The motion compensation unit 302 may use at least part of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and reference frame lists) for each inter-encoded block, and other information to decode the encoded video sequence. As used herein, in some aspects, a "slice" may refer to a data structure that can be decoded independently from other slices of the same picture, in terms of entropy coding, signal prediction, and residual signal reconstruction. A slice can either be an entire picture or a region of a picture.

[0058] The intra prediction unit 303 may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. The inverse quantization unit 304 inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit 301.

The inverse transform unit 305 applies an inverse transform.

[0059] The reconstruction unit 306 may obtain the decoded blocks, e.g., by summing the residual blocks with the corresponding prediction blocks generated by the motion compensation unit 302 or intra-prediction unit 303. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in the buffer 307, which provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.

[0060] Some exemplary embodiments of the present disclosure will be described in detailed hereinafter. It should be understood that section headings are used in the present document to facilitate ease of understanding and do not limit the embodiments disclosed in a section to only that section. Furthermore, while certain embodiments are described with reference to Versatile Video Coding or other specific video codecs, the disclosed techniques are applicable to other video coding technologies also. Furthermore, while some embodiments describe video coding steps in detail, it will be understood that corresponding steps decoding that undo the coding will be implemented by a decoder. Furthermore, the term video processing encompasses video coding or compression, video decoding or decompression and video transcoding in which video pixels are represented from one compressed format into another compressed format or at a different compressed bitrate.

1. Brief Summary

This disclosure is related to image/video coding technologies. Specifically, it is related to the improvement of neural-network post-processing filters. The improvements include signalling of visual quality improvement types, signalling of more auxiliary input data, dealing with different chroma components, and removing invalid operation from existing neural-network post-processing filters. The ideas may be applied individually or in various combinations, for video bitstreams coded by any codec, e.g., the versatile video coding (VVC) standard and/or the versatile SEI messages for coded video bitstreams (VSEI) standard.

2. Abbreviations

APS	Adaptation Parameter Set
AU	Access Unit
CLVS	Coded Layer Video Sequence

CLVSS	Coded Layer Video Sequence Start
CRC	Cyclic Redundancy Check
CVS	Coded Video Sequence
FIR	Finite Impulse Response
IRAP	Intra Random Access Point
NAL	Network Abstraction Layer
PPS	Picture Parameter Set
PU	Picture Unit
RASL	Random Access Skipped Leading
SEI	Supplemental Enhancement Information
STSA	Step-wise Temporal Sublayer Access
VCL	Video Coding Layer
VSEI	versatile supplemental enhancement information (Rec. ITU-T H.274 | ISO/IEC 23002-7)
VUI	Video Usability Information
VVC	versatile video coding (Rec. ITU-T H.266 | ISO/IEC 23090-3)

3. Introduction

3.1. Video coding standards

Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC standards. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, the Joint Video Exploration Team (JVET) was founded by VCEG and MPEG jointly in 2015. Since then, many new methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM). The JVET was later renamed to be the Joint Video Experts Team (JVET) when the Versatile Video Coding (VVC) project officially started. VVC is the new coding standard, targeting at 50% bitrate reduction as compared to HEVC, that has been finalized by the JVET at its 19th meeting ended at July 1, 2020.

The Versatile Video Coding (VVC) standard (ITU-T H.266 | ISO/IEC 23090-3) and the associated Versatile Supplemental Enhancement Information for coded video bitstreams (VSEI) standard (ITU-T H.274 | ISO/IEC 23002-7) have been designed for use in a maximally broad range of applications, including both the traditional uses such as television broadcast, video

conferencing, or playback from storage media, and also newer and more advanced use cases such as adaptive bit rate streaming, video region extraction, composition and merging of content from multiple coded video bitstreams, multiview video, scalable layered coding, and viewport-adaptive 360°immersive media.

The Essential Video Coding (EVC) standard (ISO/IEC 23094-1) is another video coding standard that has recently been developed by MPEG.

3.2. SEI messages in general and in VVC and VSEI

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance. Some SEI messages are required for checking bitstream conformance and for output timing decoder conformance. Other SEI messages are not required for check bitstream conformance. Annex D of VVC specifies syntax and semantics for SEI message payloads for some SEI messages, and specifies the use of the SEI messages and VUI parameters for which the syntax and semantics are specified in ITU-T H.274 | ISO/IEC 23002-7.

3.3. Signalling of neural-network post-filters

The specification of two SEI messages for signalling of neural-network post-filters is shown as follows.

Neural-network post-filter characteristics SEI message

Neural-network post-filter characteristics SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
nnpfc_id	ue(v)
nnpfc_mode_idc	ue(v)
nnpfc_purpose_and_formatting_flag	u(1)
if(nnpfc_purpose_and_formatting_flag) {	
nnpfc_purpose	ue(v)
if(nnpfc_purpose == 2 || nnpfc_purpose == 4)	
nnpfc_out_sub_c_flag	u(1)
if(nnpfc_purpose == 3 || nnpfc_purpose == 4) {	

nnpfc_pic_width_in_luma_samples	ue(v)
nnpfc_pic_height_in_luma_samples	ue(v)
}	
nnpfc_component_last_flag	u(1)
nnpfc_inp_format_flag	u(1)
if(nnpfc_inp_format_flag == 1)	
nnpfc_inp_tensor_bitdepth_minus8	ue(v)
nnpfc_inp_order_idc	ue(v)
nnpfc_auxiliary_inp_idc	ue(v)
nnpfc_separate_colour_description_present_flag	u(1)
if(nnpfc_separate_colour_description_present_flag) {	
nnpfc_colour_primaries	u(8)
nnpfc_transfer_characteristics	u(8)
nnpfc_matrix_coeffs	u(8)
}	
nnpfc_out_format_flag	u(1)
if(nnpfc_out_format_flag == 1)	
nnpfc_out_tensor_bitdepth_minus8	ue(v)
nnpfc_out_order_idc	ue(v)
nnpfc_constant_patch_size_flag	u(1)
nnpfc_patch_width_minus1	ue(v)
nnpfc_patch_height_minus1	ue(v)
nnpfc_overlap	ue(v)
nnpfc_padding_type	ue(v)
if(nnpfc_padding_type == 4){	
nnpfc_luma_padding_val	ue(v)
nnpfc_cb_padding_val	ue(v)
nnpfc_cr_padding_val	ue(v)
}	
nnpfc_complexity_idc	ue(v)
if(nnpfc_complexity_idc > 0)	
nnpfc_complexity_element(nnpfc_complexity_idc)	

if(nnpfc_mode_idc == 2) {	
while(!byte_aligned( ))	
nnpfc_reserved_zero_bit	u(1)
nnpfc_uri_tag[i]	st(v)
nnpfc_uri[i]	st(v)
}	
}	
/* filter specified or updated by ISO/IEC 15938-17 bitstream */	
if(nnpfc_mode_idc == 1) {	
while(!byte_aligned( ))	
nnpfc_reserved_zero_bit	u(1)
for(i = 0; more_data_in_payload( ); i++)	
nnpfc_payload_byte[i]	b(8)
}	
}	

nnpfc_complexity_element(nnpfc_complexity_idc) {	**Descriptor**
if(nnpfc_complexity_idc == 1) {	
nnpfc_parameter_type_idc	u(2)
if (nnpfc_parameter_type_idc ! = 2)	
nnpfc_log2_parameter_bit_length_minus3	u(2)
nnpfc_num_parameters_idc	u(6)
nnpfc_num_kmac_operations_idc	ue(v)
}	
}	

Neural-network post-filter characteristics SEI message semantics

This SEI message specifies a neural network that may be used as a post-processing filter. The use of specified post-processing filters for specific pictures is indicated with neural-network post-filter activation SEI messages.

Use of this SEI message requires the definition of the following variables:

– Cropped decoded output picture width and height in units of luma samples, denoted herein

by CroppedWidth and CroppedHeight, respectively.

- Luma sample array CroppedYPic[x][y] and chroma sample arrays CroppedCbPic[x][y] and CroppedCrPic[x][y], when present, of the cropped decoded output picture for vertical coordinates y and horizontal coordinates x, where the top-left corner of the sample array has coordinates y equal to 0 and x equal to 0.
- Bit depth $BitDepth_Y$ for the luma sample array of the cropped decoded output picture.
- Bit depth $BitDepth_C$ for the chroma sample arrays, if any, of the cropped decoded output picture.
- A chroma format indicator, denoted herein by ChromaFormatIdc, as described in clause 7.3.
- When nnpfc_auxiliary_inp_idc is equal to 1, a quantization strength value StrengthControlVal.

When this SEI message specifies a neural network that may be used as a post-processing filter, the semantics specify the derivation of the luma sample array FilteredYPic[x][y] and chroma sample arrays FilteredCbPic[x][y] and FilteredCrPic[x][y], as indicated by the value of nnpfc_out_order_idc, that contain the output of the post-processing filter.

The variables SubWidthC and SubHeightC are derived from ChromaFormatIdc as specified by Table 2.

nnpfc_id contains an identifying number that may be used to identify a post-processing filter. The value of nnpfc_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of nnpfc_id from 256 to 511, inclusive, and from 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of nnpfc_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore it.

nnpfc_mode_idc equal to 0 specifies that the post-processing filter associated with the nnpfc_id value is determined by external means not specified in this Specification.

nnpfc_mode_idc equal to 1 specifies that the post-processing filter associated with the nnpfc_id value is a neural network represented by the ISO/IEC 15938-17 bitstream contained in this SEI message.

nnpfc_mode_idc equal to 2 specifies that the post-processing filter associated with the nnpfc_id value is a neural network identified by a specified tag Uniform Resource Identifier (URI) (nnpfc_uri_tag[i]) and neural network information URI (nnpfc_uri[i]).

The value of nnpfc_mode_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_mode_idc greater than 2 are reserved for future specification by ITU-T | ISO/IEC and

shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_mode_idc.

nnpfc_purpose_and_formatting_flag equal to 0 specifies that no syntax elements related to the filter purpose, input formatting, output formatting, and complexity are present. nnpfc_purpose_and_formatting_flag equal to 1 specifies that syntax elements related to the filter purpose, input formatting, output formatting, and complexity are present.

When nnpfc_mode_idc is equal to 1 and the current CLVS does not contain a preceding neural-network post-filter characteristics SEI message, in decoding order, that has the value of nnpfc_id equal to the value of nnpfc_id in this SEI message, nnpfc_purpose_and_formatting_flag shall be equal to 1.

When the current CLVS contains a preceding neural-network post-filter characteristics SEI message, in decoding order, that has the same value of nnpfc_id equal to the value of nnpfc_id in this SEI message, at least one of the following conditions shall apply:

- This SEI message has nnpfc_mode_idc equal to 1 and nnpfc_purpose_and_formatting_flag equal to 0 in order to provide a neural network update.
- This SEI message has the same content as the preceding neural-network post-filter characteristics SEI message.

When this SEI message is the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within the current CLVS, it specifies a base post-processing filter that pertains to the current decoded picture and all subsequent decoded pictures of the current layer, in output order, until the end of the current CLVS. When this SEI message is not the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within the current CLVS, this SEI message pertains to the current decoded picture and all subsequent decoded pictures of the current layer, in output order, until the end of the current CLVS or the next neural-network post-filter characteristics SEI message having that particular nnpfc_id value, in output order, within the current CLVS.

nnpfc_purpose indicates the purpose of post-processing filter as specified in Table 1. The value of nnpfc_purpose shall be in the range of 0 to $2^{32} - 2$, inclusive. Values of nnpfc_purpose that do not appear in Table 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_purpose.

Table 1 – Definition of nnpfc_purpose

Value	Interpretation
0	Unknown or unspecified
1	Visual quality improvement
2	Chroma upsampling from the 4:2:0 chroma format to the 4:2:2 or 4:4:4 chroma format, or from the 4:2:2 chroma format to the 4:4:4 chroma format
3	Increasing the width or height of the cropped decoded output picture without changing the chroma format
4	Increasing the width or height of the cropped decoded output picture and upsampling the chroma format

NOTE 1 – When a reserved value of nnpfc_purpose is taken into use in the future by ITU-T | ISO/IEC, the syntax of this SEI message could be extended with syntax elements whose presence is conditioned by nnpfc_purpose being equal to that value.

When SubWidthC is equal to 1 and SubHeightC is equal to 1, nnpfc_purpose shall not be equal to 2 or 4.

nnpfc_out_sub_c_flag equal to 1 specifies that outSubWidthC is equal to 1 and outSubHeightC is equal to 1. nnpfc_out_sub_c_flag equal to 0 specifies that outSubWidthC is equal to 2 and outSubHeightC is equal to 1. When nnpfc_out_sub_c_flag is not present, outSubWidthC is inferred to be equal to SubWidthC and outSubHeightC is inferred to be equal to SubHeightC. If SubWidthC is equal to 2 and SubHeightC is equal to 1, nnpfc_out_sub_c_flag shall not be equal to 0.

nnpfc_pic_width_in_luma_samples and **nnpfc_pic_height_in_luma_samples** specify the width and height, respectively, of the luma sample array of the picture resulting by applying the post-processing filter identified by nnpfc_id to a cropped decoded output picture. When nnpfc_pic_width_in_luma_samples and nnpfc_pic_height_in_luma_samples are not present, they are inferred to be equal to CroppedWidth and CroppedHeight, respectively.

nnpfc_component_last_flag equal to 0 specifies that the second dimension in the input tensor inputTensor to the post-processing filter and the output tensor outputTensor resulting from the post-processing filter is used for the channel. nnpfc_component_last_flag equal to 1 specifies that the last dimension in the input tensor inputTensor to the post-processing filter and the output tensor outputTensor resulting from the post-processing filter is used for the channel.

NOTE 2 – The first dimension in the input tensor and in the output tensor is used for the

batch index, which is a practice in some neural network frameworks. While the semantics of this SEI message use batch size equal to 1, it is up to the post-processing implementation to determine the batch size used as input to the neural network inference.

NOTE 3 – A colour component is an example of a channel.

nnpfc_inp_format_flag indicates the method of converting a sample value of the cropped decoded output picture to an input value to the post-processing filter. When nnpfc_inp_format_flag is equal to 0, the input values to the post-processing filter are real numbers and the functions InpY and InpC are specified as follows:

$$InpY(x) = x \div ((1 << BitDepth_Y) - 1) \quad (75)$$

$$InpC(x) = x \div ((1 << BitDepth_C) - 1) \quad (76)$$

When nnpfc_inp_format_flag is equal to 1, the input values to the post-processing filter are unsigned integer numbers and the functions InpY and InpC are specified as follows:

shift = $BitDepth_Y$ − inpTensorBitDepth
if(inpTensorBitDepth >= $BitDepth_Y$)
    InpY(x) = x << (inpTensorBitDepth − $BitDepth_Y$)
else
    InpY(x) = Clip3(0, (1 << inpTensorBitDepth) − 1, (x + (1 << (shift − 1))) >> shift) (77)
shift = $BitDepth_C$ − inpTensorBitDepth
if(inpTensorBitDepth >= $BitDepth_C$)
    InpC(x) = x << (inpTensorBitDepth − $BitDepth_C$)
else
    InpC(x) = Clip3(0, (1 << inpTensorBitDepth) − 1, (x + (1 << (shift − 1))) >> shift).

The variable inpTensorBitDepth is derived from the syntax element nnpfc_inp_tensor_bitdepth_minus8 as specified below.

nnpfc_inp_tensor_bitdepth_minus8 plus 8 specifies the bit depth of luma sample values in the input integer tensor. The value of inpTensorBitDepth is derived as follows:

$$inpTensorBitDepth = nnpfc_inp_tensor_bitdepth_minus8 + 8 \quad (78)$$

It is a requirement of bitstream conformance that the value of nnpfc_inp_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

nnpfc_auxiliary_inp_idc not equal to 0 specifies auxiliary input data is present in the input

tensor of the neural-network post-filter. nnpfc_auxiliary_inp_idc equal to 0 indicates that auxiliary input data is not present in the input tensor. nnpfc_auxiliary_inp_idc equal to 1 specifies that auxiliary input data is derived as specified in Table 4 below. The value of nnpfc_auxiliary_inp_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_auxiliary_inp_idc greater than 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_auxiliary_inp_idc.

nnpfc_separate_colour_description_present_flag equal to 1 indicates that a distinct combination of colour primaries, transfer characteristics, and matrix coefficients for the picture resulting from the post-processing filter is specified in the SEI message syntax structure. nnfpc_separate_colour_description_present_flag equal to 0 indicates that the combination of colour primaries, transfer characteristics, and matrix coefficients for the picture resulting from the post-processing filter is the same as indicated in VUI parameters for the CLVS.

nnpfc_colour_primaries has the same semantics as specified in clause 7.3 for the vui_colour_primaries syntax element, except as follows:

– nnpfc_colour_primaries specifies the colour primaries of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the colour primaries used for the CLVS.

– When nnpfc_colour_primaries is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_colour_primaries is inferred to be equal to vui_colour_primaries.

nnpfc_transfer_characteristics has the same semantics as specified in clause 7.3 for the vui_transfer_characteristics syntax element, except as follows:

– nnpfc_transfer_characteristics specifies the transfer characteristics of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the transfer characteristics used for the CLVS.

– When nnpfc_transfer_characteristics is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_transfer_characteristics is inferred to be equal to vui_transfer_characteristics.

nnpfc_matrix_coeffs has the same semantics as specified in clause 7.3 for the vui_matrix_coeffs syntax element, except as follows:

– nnpfc_matrix_coeffs specifies the matrix coefficients of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the matrix

coefficients used for the CLVS.

– When nnpfc_matrix_coeffs is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_matrix_coeffs is inferred to be equal to vui_matrix_coeffs.

– The values allowed for nnpfc_matrix_coeffs are not constrained by the chroma format of the decoded video pictures that is indicated by the value of ChromaFormatIdc for the semantics of the VUI parameters.

– When nnpfc_matrix_coeffs is equal to 0, nnpfc_out_order_idc shall not be equal to 1 or 3.

nnpfc_inp_order_idc indicates the method of ordering the sample arrays of a cropped decoded output picture as the input to the post-processing filter. Table 2 below contains an informative description of nnpfc_inp_order_idc values. The semantics of nnpfc_inp_order_idc in the range of 0 to 3, inclusive, are specified in Table 3 below, which specifies a process for deriving the input tensors inputTensor for different values of nnpfc_inp_order_idc and a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors. When the chroma format of the cropped decoded output picture is not 4:2:0, nnpfc_inp_order_idc shall not be equal to 3. The value of nnpfc_inp_order_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_inp_order_idc greater than 3 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_inp_order_idc.

Table 2 – Informative description of nnpfc_inp_order_idc values

nnpfc_inp_order_idc	**Description**
0	If nnpfc_auxiliary_inp_idc is equal to 0, one luma matrix is present in the input tensor, thus the number of channels is 1. Otherwise, nnpfc_auxiliary_inp_idc is not equal to 0 and one luma matrix and one auxiliary input matrix are present, thus the number of channels is 2.
1	If nnpfc_auxiliary_inp_idc is equal to 0, two chroma matrices are present in the input tensor, thus the number of channels is 2. Otherwise, nnpfc_auxiliary_inp_idc is not equal to 0 and two chroma matrices and one auxiliary input matrix are present, thus the number of channels is 3.
2	If nnpfc_auxiliary_inp_idc is equal to 0, one luma and two chroma matrices

	are present in the input tensor, thus the number of channels is 3. Otherwise, nnpfc_auxiliary_inp_idc is not equal to 0 and one luma matrix, two chroma matrices and one auxiliary input matrix are present, thus the number of channels is 4.
3	If nnpfc_auxiliary_inp_idc is equal to 0, four luma matrices and two chroma matrices are present in the input tensor, thus the number of channels is 6. Otherwise, nnpfc_auxiliary_inp_idc is not equal to 0 and four luma matrices, two chroma matrices, and one auxiliary input matrix are present in the input tensor, thus the number of channels is 7. The luma channels are derived in an interleaved manner as illustrated in Fig. 4. This nnpfc_inp_order_idc can only be used when the chroma format is 4:2:0.
4..255	reserved

Fig. 4 illustrates an illustration 400 of luma data channels of nnpfc_inp_order_idc equal to 3 (informative).

A patch is a rectangular array of samples from a component (e.g., a luma or chroma component) of a picture.

nnpfc_constant_patch_size_flag equal to 0 specifies that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. When nnpfc_constant_patch_size_flag is equal to 0 the patch size width shall be less than or equal to CroppedWidth. When nnpfc_constant_patch_size_flag is equal to 0 the patch size height shall be less than or equal to CroppedHeight. nnpfc_constant_patch_size_flag equal to 1 specifies that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the horizontal sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_width_minus1 + 1) may be used as the horizontal sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the vertical sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_height_minus1 + 1) may be used as the vertical sample counts of the patch

size used for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

nnpfc_overlap specifies the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

The variables inpPatchWidth, inpPatchHeight, outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:

```
inpPatchWidth = nnpfc_patch_width_minus1 + 1
inpPatchHeight = nnpfc_patch_height_minus1 + 1
outPatchWidth = ( nnpfc_pic_width_in_luma_samples * inpPatchWidth ) / CroppedWidth
outPatchHeight = ( nnpfc_pic_height_in_luma_samples * inpPatchHeight ) / CroppedHeight
horCScaling = SubWidthC / outSubWidthC
verCScaling = SubHeightC / outSubHeightC                                    (79)
outPatchCWidth = outPatchWidth * horCScaling
outPatchCHeight = outPatchHeight * verCScaling
overlapSize = nnpfc_overlap
```

It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.

nnpfc_padding_type specifies the process of padding when referencing sample locations outside the boundaries of the cropped decoded output picture as described in Table 3 below. The value of nnpfc_padding_type shall be in the range of 0 to 15, inclusive.

Table 3 – Informative description of nnpfc_padding_type values

nnpfc_padding_type	Description
0	zero padding
1	replication padding
2	reflection padding
3	wrap-around padding
4	fixed padding
5..15	reserved

nnpfc_luma_padding_val specifies the luma value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cb_padding_val specifies the Cb value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cr_padding_val specifies the Cr value to be used for padding when nnpfc_padding_type is equal to 4.

The function InpSampleVal(y, x, picHeight, picWidth, croppedPic) with inputs being a vertical sample location y, a horizontal sample location x, a picture height picHeight, a picture width picWidth, and sample array croppedPic returns the value of sampleVal derived as follows:

```
if( nnpfc_padding_type == 0 )
    if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
      sampleVal = 0
    else
      sampleVal = croppedPic[ x ][ y ]                                    (80)
else if( nnpfc_padding_type == 1 )
    sampleVal = croppedPic[ Clip3( 0, picWidth − 1, x ) ][ Clip3( 0, picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 2 )
    sampleVal = croppedPic[ Reflect( picWidth − 1, x ) ][ Reflect( picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 3 )
    if(y >= 0 && y < picHeight)
      sampleVal = croppedPic[ Wrap( picWidth − 1, x ) ][ y ]
else if( nnpfc_padding_type == 4 )
    if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
      sampleVal[ 0 ] = nnpfc_luma_padding_val
          sampleVal[ 1 ] = nnpfc_cb_padding_val
      sampleVal[ 2 ] = nnpfc_cr_padding_val
    else
      sampleVal = croppedPic[ x ][ y ]
```

Table 4 – Process for deriving the input tensors inputTensor for a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors

nnpfc_inp_order_idc	Process DeriveInputTensors( ) for deriving input tensors

0	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpVal = InpY(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedYPic))
	if(nnpfc_component_last_flag == 0)

	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpVal
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpVal
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

1	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpCbVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpCbVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCrVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

2	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++) for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {

	yY = cTop + yP
	xY = cLeft + xP
	yC = yY / SubHeightC
	xC = xY / SubWidthC
	inpYVal = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPic))
	inpCbVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpYVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCbVal
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpCrVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpYVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

3	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yTL = cTop + yP * 2
	xTL = cLeft + xP * 2
	yBR = yTL + 1
	xBR = xTL + 1
	yC = cTop / 2 + yP
	xC = cLeft / 2 + xP
	inpTLVal = InpY(InpSampleVal(yTL, xTL, CroppedHeight,
	CroppedWidth, CroppedYPic))

	inpTRVal = InpY(InpSampleVal(yTL, xBR, CroppedHeight, CroppedWidth, CroppedYPic))
	inpBLVal = InpY(InpSampleVal(yBR, xTL, CroppedHeight, CroppedWidth, CroppedYPic))
	inpBRVal = InpY(InpSampleVal(yBR, xBR, CroppedHeight, CroppedWidth, CroppedYPic))
	inpCbVal = InpC(InpSampleVal(yC, xC, CroppedHeight / 2, CroppedWidth / 2, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(yC, xC, CroppedHeight / 2, CroppedWidth / 2, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpTLVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpTRVal
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpBLVal
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpBRVal
	inputTensor[0][4][yP + overlapSize][xP + overlapSize] = inpCbVal
	inputTensor[0][5][yP + overlapSize][xP + overlapSize] = inpCrVal
	inputTensor[0][6][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpTLVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpTRVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpBLVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpBRVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][4] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][5] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][6][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][6] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

4..255	reserved

nnpfc_complexity_idc greater than 0 specifies that one or more syntax elements that indicate the complexity of the post-processing filter associated with the nnpfc_id may be present. nnpfc_complexity_idc equal to 0 specifies that no syntax element that indicates the complexity of the post-processing filter associated with the nnpfc_id is present. The value nnpfc_complexity_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_complexity_idc greater than 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_complexity_idc.

nnpfc_out_format_flag equal to 0 indicates that the sample values output by the post-processing filter are real numbers and the functions OutY and OutC for converting luma sample values and chroma sample values, respectively, output by the post-processing, to integer values at bit depths $BitDepth_Y$ and $BitDepth_C$, respectively, are specified as follows:

$$OutY(x) = Clip3(0, (1 << BitDepth_Y) - 1, Round(x * ((1 << BitDepth_Y) - 1))) \quad (81)$$

$$OutC(x) = Clip3(0, (1 << BitDepth_C) - 1, Round(x * ((1 << BitDepth_C) - 1))) \quad (82)$$

nnpfc_out_format_flag equal to 1 indicates that the sample values output by the post-processing filter are unsigned integer numbers and the functions OutY and OutC are specified as follows:

```
shift = outTensorBitDepth − BitDepthY
if( shift > 0 )
    OutY( x ) = Clip3( 0, ( 1 << BitDepthY ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )
else
    OutY( x ) = x << ( BitDepthY − outTensorBitDepth )                    (83)
shift = outTensorBitDepth − BitDepthC
if( shift > 0 )
    OutC( x ) = Clip3( 0, ( 1 << BitDepthC ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )
else
    OutC( x ) = x << ( BitDepthC − outTensorBitDepth )
```

The variable outTensorBitDepth is derived from the syntax element nnpfc_out_tensor_bitdepth_minus8 as described below.

nnpfc_out_tensor_bitdepth_minus8 plus 8 specifies the bit depth of sample values in the

output integer tensor. The value of outTensorBitDepth is derived as follows:

$$outTensorBitDepth = nnpfc_out_tensor_bitdepth_minus8 + 8 \quad (84)$$

It is a requirement of bitstream conformance that the value of nnpfc_out_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

nnpfc_out_order_idc indicates the output order of samples resulting from the post-processing filter. Table 5 contains an informative description of nnpfc_out_order_idc values. The semantics of nnpfc_out_order_idc in the range of 0 to 3, inclusive, are specified in Table 6, which specifies a process for deriving sample values in the filtered output sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic from the output tensors outputTensor for different values of nnpfc_out_order_idc and a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors. When nnpfc_purpose is equal to 2 or 4, nnpfc_out_order_idc shall not be equal to 3. The value of nnpfc_out_order_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_out_order_idc greater than 3 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_out_order_idc.

Table 5 – Informative description of nnpfc_out_order_idc values

nnpfc_out_order_idc	Description
0	Only the luma matrix is present in the output tensor, thus the number of channels is 1.
1	Only the chroma matrices are present in the output tensor, thus the number of channels is 2.
2	The luma and chroma matrices are present in the output tensor, thus the number of channels is 3.
3	Four luma matrices and two chroma matrices are present in the output tensor, thus the number of channels is 6. This nnpfc_out_order_idc can only be used when the chroma format is 4:2:0.
4..255	reserved

Table 6 – Process for deriving sample values in the filtered output sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic from the output tensors outputTensor for a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors

nnpfc_out_order_idc	Process StoreOutputTensors( ) for deriving sample values in the filtered picture from the output tensors

0	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	yY = cTop * outPatchHeight / inpPatchHeight + yP
	xY = cLeft * outPatchWidth / inpPatchWidth + xP
	if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	if(nnpfc_component_last_flag == 0)
	FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	else
	FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	}

1	for(yP = 0; yP < outPatchCHeight; yP++)
	for(xP = 0; xP < outPatchCWidth; xP++) {
	xSrc = cLeft * horCScaling + xP
	ySrc = cTop * verCScaling + yP
	if (ySrc < nnpfc_pic_height_in_luma_samples / outSubHeightC &&
	xSrc < nnpfc_pic_width_in_luma_samples / outSubWidthC)
	if(nnpfc_component_last_flag == 0) {
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][0][yP][xP])
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][1][yP][xP])
	} else {
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][0])
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][1])
	}
	}

2	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	yY = cTop * outPatchHeight / inpPatchHeight + yP
	xY = cLeft * outPatchWidth / inpPatchWidth + xP
	yC = yY / outSubHeightC

	xC = xY / outSubWidthC
	yPc = (yP / outSubHeightC) * outSubHeightC
	xPc = (xP / outSubWidthC) * outSubWidthC
	if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][1][yPc][xPc])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][2][yPc][xPc])
	} else {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][1])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][2])
	}
	}

3	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	ySrc = cTop / 2 * outPatchHeight / inpPatchHeight + yP
	xSrc = cLeft / 2 * outPatchWidth / inpPatchWidth + xP
	if (ySrc < nnpfc_pic_height_in_luma_samples / 2 &&
	xSrc < nnpfc_pic_width_in_luma_samples / 2)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xSrc * 2][ySrc * 2] = OutY(outputTensor[0][0][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] = OutY(outputTensor[0][1][yP][xP])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] = OutY(outputTensor[0][2][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] = OutY(outputTensor[0][3][yP][xP])
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][4][yP][xP])
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][5][yP][xP])
	} else {
	FilteredYPic[xSrc * 2][ySrc * 2] = OutY(outputTensor[0][yP][xP][0])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] = OutY(outputTensor[0][yP][xP][1])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] = OutY(outputTensor[0][yP][xP][2])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] = OutY(outputTensor[0][yP][xP][3])
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][4])
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][5])
	}

	}
4..255	reserved

A base post-processing filter for a cropped decoded output picture picA is the filter that is identified by the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within a CLVS.

If there is another neural-network post-filter characteristics SEI message that has the same nnpfc_id value, has nnpfc_mode_idc equal to 1, has different content than the neural-network post-filter characteristics SEI message that defines the base post-processing filter, and pertains to the picture picA, the base post-processing filter is updated by decoding the ISO/IEC 15938-17 bitstream in that neural-network post-filter characteristics SEI message to obtain a post-processing filter PostProcessingFilter( ). Otherwise, the post-processing processing filter PostProcessingFilter( ) is assigned to be the same as the base post-processing filter.

The following process is used to filter the cropped decoded output picture with the post-processing filter PostProcessingFilter( ) to generate the filtered picture, which contains Y, Cb, and Cr sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic, respectively, as indicated by nnpfc_out_order_idc.

```
if( nnpfc_inp_order_idc == 0 )
    for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight )
      for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth ) {
          DeriveInputTensors( )
          outputTensor = PostProcessingFilter( inputTensor )
          StoreOutputTensors( )
      }
else if( nnpfc_inp_order_idc == 1 )
    for( cTop = 0; cTop < CroppedHeight / SubHeightC; cTop +=
inpPatchHeight )
        for( cLeft = 0; cLeft < CroppedWidth / SubWidthC; cLeft +=
inpPatchWidth ) {
          DeriveInputTensors( )
          outputTensor = PostProcessingFilter( inputTensor )
          StoreOutputTensors( )
      }
else if( nnpfc_inp_order_idc == 2 )
    for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight)          (85)
      for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth) {
          DeriveInputTensors( )
          outputTensor = PostProcessingFilter( inputTensor )
```

```
            StoreOutputTensors( )
        }
    else if( nnpfc_inp_order_idc == 3 )
        for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight * 2 )
            for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth * 2 ) {
                DeriveInputTensors( )
                outputTensor = PostProcessingFilter( inputTensor )
                StoreOutputTensors( )
            }
```

nnpfc_reserved_zero_bit shall be equal to 0.

nnpfc_uri_tag[i] contains a NULL-terminated UTF-8 character string specifying a tag URI. The UTF-8 character string contains a URI, with syntax and semantics as specified in IETF RFC 4151, uniquely identifying the format and associated information about the neural network used as the post-processing filter specified by nnrpf_uri[i] values.

NOTE 4 – nnrpf_uri_tag[i] elements represent a 'tag' URI, which allows uniquely identifying the format of neural network data specified by nnrpf_uri[i] values without needing a central registration authority.

nnpfc_uri[i] contains a NULL-terminated UTF-8 character string, as specified in IETF Internet Standard 63. The UTF-8 character string contains a URI, with syntax and semantics as specified in IETF Internet Standard 66, identifying the neural network information (e.g. data representation) used as the post-processing filter.

nnpfc_payload_byte[i] contains the i-th byte of a bitstream conforming to ISO/IEC 15938-17. The byte sequence nnpfc_payload_byte[i] for all present values of i shall be a complete bitstream that conforms to ISO/IEC 15938-17.

nnpfc_parameter_type_idc equal to 0 indicates that the neural network uses only integer parameters. nnpfc_parameter_type_flag equal to 1 indicates that the neural network may use floating point or integer parameters. nnpfc_parameter_type_idc equal to 2 indicates that the neural network uses only binary parameters. nnpfc_parameter_type_idc equal to 3 is reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved value of nnpfc_parameter_type_idc.

nnpfc_log2_parameter_bit_length_minus3 equal to 0, 1, 2, and 3 indicates that the neural network does not use parameters of bit length greater than 8, 16, 32, and 64, respectively. When nnpfc_parameter_type_idc is present and nnpfc_log2_parameter_bit_length_minus3 is not present the neural network does not use parameters of bit length greater than 1.

nnpfc_num_parameters_idc indicates the maximum number of neural network parameters for the post processing filter in units of a power of 2048. nnpfc_num_parameters_idc equal to

0 indicates that the maximum number of neural network parameters is not specified. The value nnpfc_num_parameters_idc shall be in the range of 0 to 52, inclusive. Values of nnpfc_num_parameters_idc greater than 52 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_num_parameters_idc.

If the value of nnpfc_num_parameters_idc is greater than zero, the variable maxNumParameters is derived as follows:

$$maxNumParameters = (2048 << nnpfc_num_parameters_idc) - 1 \quad (86)$$

It is a requirement of bitstream conformance that the number of neural network parameters of the post-processing filter shall be less than or equal to maxNumParameters.

nnpfc_num_kmac_operations_idc greater than 0 specifies that the maximum number of multiply-accumulate operations per sample of the post-processing filter is less than or equal to nnpfc_num_kmac_operations_idc * 1000. nnpfc_num_kmac_operations_idc equal to 0 specifies that the maximum number of multiply-accumulate operations of the network is not specified. The value of nnpfc_num_kmac_operations_idc shall be in the range of 0 to $2^{32} - 1$, inclusive.

Neural-network post-filter activation SEI message

Neural-network post-filter activation SEI message syntax

nn_post_filter_activation(payloadSize) {	**Descriptor**
nnpfa_id	ue(v)
}	

Neural-network post-filter activation SEI message semantics

This SEI message specifies the neural-network post-processing filter that may be used for post-processing filtering for the current picture.

The neural-network post-processing filter activation SEI message persists only for the current picture.

NOTE – There may be several neural-network post-processing filter activation SEI messages present for the same picture, for example, when the post-processing filters are meant for different purposes or filter different colour components.

nnpfa_id specifies that the neural-network post-processing filter specified by one or more neural-network post-processing filter characteristics SEI messages that pertain to the current

picture and have nnpfc_id equal to nnfpa_id may be used for post-processing filtering for the current picture.

4. Problems

The current design for the neural-network post-filter characteristics (NNPFC) SEI message has the following problems:

1) The NNPFC SEI message specifies that the purpose of a neural-network post-processing filter (NNPF) may be for visual quality improvement. However, visual quality improvement may have different interpretations, such as fidelity-constrained visual quality improvement, GAN-based (Generative Adversarial Network-based) visual quality improvement, film grain-based visual quality improvement, etc. Different video applications may prefer different types of visual quality improvements. For example, fidelity is important in surveillance scenario, but may not be so critical in some user-generated videos. Therefore, there is a need to be able to identify the type of visual quality improvement. Additionally, machine vision tasks are critical for industry use. It is also desirable to specify the purpose of NNPF with machine vision tasks.

2) The NNPFC SEI message specifies the auxiliary input to the neural network, which only includes quantization parameter-related data. However, there are other auxiliary inputs which may also be very helpful in terms of improving the performance of post-processing filtering. Therefore, there is a need to specify more auxiliary inputs.

3) The NNPFC SEI message specifies that the chroma matrices presented in the output tensor of NN filter include two channels. However, the capability of processing two chroma components separately can be useful. Therefore, there is a need to enable applying NN filter only on one of the chroma components.

4) The NNPFC SEI message specifies the process for deriving the input tensor inputTensor in Table 4. When nnpfc_inp_order_idc is equal to 3, nnpfc_component_last_flag is equal to 0, and nnpfc_auxiliary_inp_idc is equal to 0, no auxiliary input matrix is supposed to be present. Therefore, the assignment operation "inputTensor[0][6][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$" should be removed in this case.

5) The value of nnpfc_constant_patch_size_flag equal to 0 specifies that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. nnpfc_constant_patch_size_flag equal to 1 specifies that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. However, on the one hand, it is allowed for nnpfc_constant_patch_size_flag to be equal to 0, while on the other, regardless of the value of nnpfc_constant_patch_size_flag, the filtering process specified as part of the semantics of the NNPFC SEI message always use exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. In other words, the actual support of taking as input

any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 is missing.

6) The NNPFC SEI message specifies the bit depth of luma and chroma components, respectively. This means that NNPFC SEI message support different inputs with different bit depth for different color components. However, the bitdepth of output tensor of the NNPF is constrained to the same bitdepth.

7) Depending on nnpfc_constant_patch_size_flag, whether to take overlapsize into account is inconsistent.

5. Detailed Solutions

To solve the above problems, methods as summarized below are disclosed. The embodiments should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these embodiments can be applied individually or combined in any manner.

In the following description, the term "picture" may be replaced with any video unit, such as "slice".

1) To solve problem 1, one or more types of purposes are defined, including fine-grained visual quality improvements-oriented and machine vision tasks-oriented.

    a. In one example, one or more types of visual quality improvements are defined, and the visual quality improvement type of a neural-network post-processing filter with the visual quality improvement purpose is signalled in the NNPFC SEI message:

        i. In one example, a type of visual quality improvement is defined as objective-oriented/fidelity-oriented, targeting at increasing the fidelity of the reconstructed picture after applying the neural-network post-processing filter. The fidelity may be measured by PSNR (Peak Signal-to-Noise Ratio), Ms-SSIM (Multi-scale Structural Similarity), etc.

        ii. In one example, a type of visual quality improvement is defined as subjective-oriented, targeting at increasing the subjective visual quality of the reconstructed picture after applying the neural-network post-processing filter. The subjective visual quality may be measured by LPIPS (Learned Perceptual Image Patch Similarity), MOS (Mean Opinion Score), etc.

        iii. In one example, a type of visual quality improvement is defined as filter grain-oriented, with synthesizing filter grain on the reconstructed picture after applying the neural-network post-processing filter.

    b. In one example, one or more visual quality improvements related purposes are added as NNPFC purposes:

        i. In one example, image/video dehazing is added as a purpose.

        ii. In one example, image/video deraining is added as a purpose.

c. In one example, one or more machine vision tasks-oriented NNPFC purposes are added:

i. In one example, the purpose is to improve the performance of an enhancement task, which targets enhancing pictures to provide better input for other processing techniques. Edge enhancement may be defined as such an enhancement task.

ii. In one example, the purpose is to improve the performance of an analysis/detection/recognition task. Face recognition, object recognition, object tracking, object segmentation, etc. may be examples of such tasks.

iii. In one example, the purpose is to improve the performance of general machine vision tasks. The machine vision tasks could be any low-level image/video processing and/or high-level image/video understanding/recognition/detection, etc.

iv. In one example, the purpose is the machine vision tasks. The machine vision tasks could be any low-level image/video processing and/or high-level image/video understanding/recognition/detection, etc.

d. In one example, one or more types of visual information processing are defined, and the visual information processing type of a neural-network post-processing filter with the visual information processing purpose is signalled in the NNPFC SEI message:

i. In one example, visual information processing is defined as objective-oriented/fidelity-oriented/subjective-oriented/filter grain-oriented/machine vision task-oriented.

e. In one example, image/video style transfer is added as a NNPFC purpose.

f. In one example, image/video object removal is added as a NNPFC purpose.

g. In one example, image/video view synthesis is added as a purpose.

h. In one example, image/video caption is added as a purpose.

i. In one example, image/video annotation is added as a purpose.

2) To solve problem 2, more auxiliary inputs to the neural network post-processing filter are defined.

a. In one example, the auxiliary input includes prediction information such as prediction samples, prediction modes, etc.

b. In one example, the auxiliary input includes partitioning information such partitioning boundary.

c. In one example, the auxiliary input includes information from previously decoded pictures such as samples of collocated blocks or motion compensated blocks of current to-be-processed block in previously decoded pictures.

d. In one example, different color components share or are allowed to share the same auxiliary input.

e. In one example, different color components use or are allowed to use different auxiliary inputs.

f. In one example, two chroma components use or are allowed to use the same auxiliary input, which is different from the auxiliary input of the luma component.

3) To solve problem 3, the matrices presented in the input and/or output tensor of an NN filter may include luma, and/or cb, and/or cr components, or any combination of these components.

a. In one example, only a chroma matrix is present in the input and/or output tensor, the number of channels is 1, and the component is Cb.

b. In one example, only a chroma matrix is present in the input and/or output tensor, the number of channels is 1, and the component is Cr.

c. In one example, only a luma matrix and a chroma matrix are present in the input and/or output tensor, the number of channels is 2, and the components are luma and Cb.

d. In one example, only a luma matrix and a chroma matrix are present in the input and/or output tensor, the number of channels is 2, and the components are luma and Cr.

e. In one example, only four luma matrices and one chroma matrix are present in the input and/or output tensor, the number of channels is 5, and the components are luma and Cb. This can only be used when the chroma format is 4:2:0.

f. In one example, only four luma matrices and one chroma matrix are present in the input and/or output tensor, the number of channels is 5, and the components are luma and Cr. This can only be used when the chroma format is 4:2:0.

g. In one example, an indication is signalled to indicate whether the color components present in the output tensor will be used.

i. In one example, this indication is signalled in the NNPFA SEI message.

ii. In one example, the indication indicates that a color component presented in the output tensor will not be used, meaning the color component remains unchanged after applying the neural network post-processing filter.

iii. In one example, the indication indicates that a color component presented in the output tensor will be used.

1. In one example, the output tensor includes both Cb and Cr components.

a. In one example, the indication indicates that only Cb components in the output tensor will be used.

b. In one example, the indication indicates that only Cr components in the output tensor will be used.

c. In one example, the indication indicates that both Cb and Cr components in the output tensor will be used.

2. In one example, the output tensor includes the luma, Cb, and Cr components.

a. In one example, the indication indicates that only luma components in the output tensor will be used.

b. In one example, the indication indicates that only Cb components in the output tensor will be used.

c. In one example, the indication indicates that only Cr components in the output tensor will be used.

d. In one example, the indication indicates that both Cb and Cr components in the output tensor will be used.

e. In one example, , the indication indicates any combination of these three color components in the output tensor will be used.

3. In one example, the unuesd color components in the output tensor will be replaced with the corresponding color components in the input tensor.

h. In one example, one or more syntax elements are signalled to indicate whether the colour components presented in the output tensor will be used.

i. In one example, whether to signal such elements may depend on color format.

ii. In one example, an indication (e.g., in a first level) is signalled to indicate that all the components presented in the output tensor will be used by default.

1. In one example, the indication (e.g., in the first level) is equal to 1 which indicates that all the components presented in the output tensor will be used by default.

2. In one example, the indication (e.g., in the first level) is equal to 0 which indicates that partial components presented in the output tensor will be used and at least one component presented in the output tensor will be NOT used.

iii. In one example, multiple indications may be signalled to indicate one/more colour components presented in the output tensor will be used by default.

1. In one example, these indications may be signalled conditionally.

a. In one example, if the indication at the first level indicates the partial components presented in the output tensor, additional (e.g., three) indications (e.g., in a second level) are conditionally signalled to indicate whether Y, U, and

V color components present in the output tensor will be used, respectively.

b. In one example, the component X present in the output tensor may be used when the indication in second level for component X equals to 1.

i. In one example, X is Y or U(Cb) or V(Cr).

ii. In one example, X is R or G or B.

c. In one example, the component X present in the output tensor may be NOT used when the indication in second level for component X equals to 0.

i. In one example, X is Y or U(Cb) or V(Cr).

ii. In one example, X is R or G or B.

d. In one example, the three indications in second level may be signalled only when the indication in first level is equal to 0 which indicates that partial components presented in the output tensor will be used and at least one component presented in the output tensor will be NOT used.

e. In one example, the indication in second level for the last component X may be NOT signalled and be inferred to 0 when other two indications in second level for other two components are equal to 1.

f. In one example, the indication in second level for the last component X may be signalled and be constraint to 0 when other two indications in second level for other two components are equal to 1.

g. In one example, the indication in second level for the last component X may be NOT signalled and be inferred to 1 when other two indications in second level for other two components are equal to 0.

h. In one example, the indication in second level for the last component X may be signalled and be constraint to 1 when other two indications in second level for other two components are equal to 0.

iv. In one example, the unused color components in the output tensor will be replaced with the corresponding color components in the input tensor.

4) To solve problem 4, when nnpfc_inp_order_idc is equal to 3, nnpfc_component_last_flag is equal to 0, and nnpfc_auxiliary_inp_idc is equal to 0, no auxiliary input matrix is used.

5) To solve problem 5, one more of the following aspects are specified:

a. In one example, when nnpfc_constant_patch_size_flag is equal to 0, the patch size width, denoted by inpPatchWidth, and the patch size height, denoted by

inpPatchHeight, are provided by external means not specified in this document. The value of inpPatchWidth shall be a positive integer multiple of nnpfc_patch_width_minus1 + 1 and shall be less than or equal to CroppedWidth. The value of PatchSizeH shall be a positive integer multiple of nnpfc_patch_height_minus1 + 1 and shall be less than or equal to CroppedHeight.

i. An example of such external means is an API that passes the values of inpPatchWidth and inpPatchHeight to the decoder and render entity in a video application system, and the values may be configured by a user through a user interface of the application.

b. In one example, when nnpfc_constant_patch_size_flag is equal to 1, the value of inpPatchWidth is set equal to nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

c. In one example, regardless of the value of nnpfc_constant_patch_size_flag, the filtering process takes as input the patch size width inpPatchWidth and the patch size height inpPatchHeight as specified above.

6) To solve problem 6, one more of the following aspects are specified:

a. In one example, the upsampling bit depth for luma and chroma and/or bit depth of luma and chroma components of output tensor may be signalled separately.

i. Alternatively, furthermore, they may be conditionally signalled, e.g., according to color format.

b. In one example, an indication in the first level indicates whether the bit depth of luma and chroma components are the same is signalled.

i. In one example, furthermore, if same bit depth is used for both luma and chroma components, one element syntax in the second level is signalled to specify the bit depth of all the components of output tensor.

1. In one example, indication in the first level equals to 1.

ii. In one example, furthermore, if different bit depths are used for luma and chroma components, one or more element syntaxs in the second level are signalled to specify the bit depth of luma and chroma components of output tensor.

1. In one example, indication in the first level equals to 0.

2. In one example, two indications in the second level are signalled. The first indication specifies the bit depth of luma components of the output tensor. The second indication specifies the bit depth of chroma components of the output tensor.

3. In one example, three indications in the second level are signalled. The first indication specifies the bit depth of Y components of the output tensor. The second indication specifies the bit depth of Cb components of the output tensor. The third indication specifies the bit depth of Cr components of the output tensor.

7) To solve problem 7, one more of the following aspects are specified:

a. In one example, when nnpfc_constant_patch_size_flag is equal to 0, a syntax element that specifies the patch width minus 2 * overlapSize is signalled.

b. In one example, when nnpfc_constant_patch_size_flag is equal to 0, a syntax element that specifies the patch height minus 2* overlapSize is signalled.

c. In one example, regardless of the value of nnpfc_constant_patch_size_flag, a syntax element that specifies the patch width minus 2 * overlapSize is signalled.

d. In one example, regardless of the value of nnpfc_constant_patch_size_flag, a syntax element that specifies the patch height minus 2* overlapSize is signalled.

e. In one example, the syntax element nnpfc_overlap is signalled before the syntax element that specifies the patch width minus 2 * overlapSize and the syntax element that specifies the patch height minus 2 * overlapSize.

f. In one example, the syntax element nnpfc_overlap is signalled before the syntax element that specifies the patch width minus 1 and the syntax element that specifies the patch height minus 1.

8) To solve problem 7, one more of the following aspects are specified:

a. In one example, in the NNPFC SEI message, the syntax element nnpfc_overlap is moved to be before the syntax element nnpfc_constant_patch_size_flag.

b. In one example, in the NNPFC SEI message, the syntax elements nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 are only signalled when nnpfc_constant_patch_size_flag is equal to 1.

c. In one example, in the NNPFC SEI message, two additional syntax elements, indicating a horizontal common divisor and a vertical common divisor of all allowed values of the width and the height, respectively, of the extended patch (i.e., a patch plus the overlapping area) required for the input to the post-processing filter, are signalled, and one or more of the following aspects apply:

i. In one example, the two additional syntax elements specify the value of the horizontal common divisor minus 1 and the vertical common divisor minus 1, respectively.

ii. In one example, the two additional syntax elements specify the value of the horizontal common divisor minus 1 minus 2 * nnpfc_overlap and the vertical common divisor minus 1 minus 2 * nnpfc_overlap, respectively.

iii. In one example, the two additional syntax elements as described above are signalled in the NNPFC SEI message only when nnpfc_constant_patch_size_flag is equal to 0.

iv. In one example, it is specified that, nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts as input any patch size with width inpPatchWidth and height inpPatchHeight such that the width of the extended patch (i.e., a patch plus the overlapping area), which is equal to inpPatchWidth + 2 * nnpfc_overlap, is a positive

integer multiple of the horizontal common divisor, and the height of the extended patch, which is equal to inpPatchHeight + 2 * nnpfc_overlap, is a positive integer multiple of the vertical common divisor.

6. Embodiments

Below are some example embodiments for the case when a video unit is a picture for the embodiment items 1, 2, 3, 4, and their subitems summarized above in Section 5.

Most relevant parts that have been added or modified are underlined, and some of the deleted parts are highlighted in ~~strikethrough~~. There may be some other changes that are editorial in nature and thus not highlighted.

6.1. Embodiment 1

This embodiment is for the case when a video unit is a picture for the embodiment item 1 and all its subitems summarized above in Section 5.

Neural-network post-filter characteristics SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
nnpfc_id	ue(v)
nnpfc_mode_idc	ue(v)
nnpfc_purpose_and_formatting_flag	u(1)
if(nnpfc_purpose_and_formatting_flag) {	
nnpfc_purpose	ue(v)
if(nnpfc_purpose == 1)	
nnpfc_visual_quality_improvement_type	ue(v)
if(nnpfc_purpose == 2 || nnpfc_purpose == 4)	
nnpfc_out_sub_c_flag	u(1)
if(nnpfc_purpose == 3 || nnpfc_purpose == 4) {	
nnpfc_pic_width_in_luma_samples	ue(v)
nnpfc_pic_height_in_luma_samples	ue(v)
}	
...	
}	

...	
}	

Neural-network post-filter activation SEI message semantics

...

nnpfc_purpose indicates the purpose of post-processing filter as specified in Table 1. The value of nnpfc_purpose shall be in the range of 0 to $2^{32} - 2$, inclusive. Values of nnpfc_purpose that do not appear in Table 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_purpose.

When SubWidthC is equal to 1 and SubHeightC is equal to 1, nnpfc_purpose shall not be equal to 2 or 4.

nnpfc_visual_quality_improvement_type indicates the type of visual quality improvement as specified in Table 7. The value of nnpfc_visual_quality_improvement_type shall be in the range of 0 to 255, inclusive. Values of nnpfc_visual_quality_improvement_type that do not appear in Table 7 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification encountering a value of nnpfc_visual_quality_improvement_type greater than 3 shall ignore it.

Table 7 – Definition of nnpfc_visual_quality_improvement_type

Value	Interpretation
0	General visual quality improvement, targeting at increasing either the fidelity or the subjective visual quality of the reconstructed picture after applying the neural-network post-processing filter. The improvement may be measured by any objective or subjective metric.
1	Objective-oriented/fidelity-oriented visual quality improvement, targeting at increasing the fidelity of the reconstructed picture after applying the neural-network post-processing filter. The fidelity may be measured by PSNR, Ms-SSIM etc.
2	Subjective-oriented visual quality improvement, targeting at increasing the subjective visual quality of the reconstructed picture after applying the neural-network post-processing filter. The subjective visual quality may be measured by LPIPS or MOS.

3	Film grain-oriented visual quality improvement, with synthesizing filter grain on the reconstructed picture after applying the neural-network post-processing filter.

NOTE x – When a reserved value of nnpfc_visual_quality_improvement_type is taken into use in the future by ITU-T | ISO/IEC, the syntax of this SEI message could be extended with syntax elements whose presence is conditioned by nnpfc_visual_quality_improvement_type being equal to that value.

6.2. Embodiment 2

This embodiment is for the case when a video unit is a picture for the embodiment item 2, item 3, item4 and all its subitems summarized above in Section 5.

Neural-network post-filter activation SEI message semantics

This SEI message specifies a neural network that may be used as a post-processing filter. The use of specified post-processing filters for specific pictures is indicated with neural-network post-filter activation SEI messages.

Use of this SEI message requires the definition of the following variables:

- Cropped decoded output picture width and height in units of luma samples, denoted herein by CroppedWidth and CroppedHeight, respectively.
- Luma sample array CroppedYPic[x][y] and chroma sample arrays CroppedCbPic[x][y] and CroppedCrPic[x][y], when present, of the cropped decoded output picture for vertical coordinates y and horizontal coordinates x, where the top-left corner of the sample array has coordinates y equal to 0 and x equal to 0.
- Bit depth $BitDepth_Y$ for the luma sample array of the cropped decoded output picture.
- Bit depth $BitDepth_C$ for the chroma sample arrays, if any, of the cropped decoded output picture.
- A chroma format indicator, denoted herein by ChromaFormatIdc, as described in clause 7.3.
- When nnpfc_auxiliary_inp_idc is equal to 1, a quantization strength value StrengthControlVal.
- When nnpfc_auxiliary_inp_idc is equal to 2 or 3, the cropped decoded prediction picture with the same size as the cropped decoded output picture.
- When nnpfc_auxiliary_inp_idc is equal to 2 or 3, luma prediction array CroppedYPred[x][y] and chroma prediction arrays CroppedCbPred[x][y] and CroppedCrPred[x][y], when present, of the cropped decoded prediction picture for vertical coordinates y and horizontal coordinates x, where the top-left corner of the sample

array has coordinates y equal to 0 and x equal to 0.

When this SEI message specifies a neural network that may be used as a post-processing filter, the semantics specify the derivation of the luma sample array FilteredYPic[x][y] and chroma sample arrays FilteredCbPic[x][y] and FilteredCrPic[x][y], as indicated by the value of nnpfc_out_order_idc, that contain the output of the post-processing filter.

The variables SubWidthC and SubHeightC are derived from ChromaFormatIdc as specified by Table 2.

nnpfc_id contains an identifying number that may be used to identify a post-processing filter. The value of nnpfc_id shall be in the range of 0 to $2^{32} - 2$, inclusive.

Values of nnpfc_id from 256 to 511, inclusive, and from 2^{31} to $2^{32} - 2$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of nnpfc_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to $2^{32} - 2$, inclusive, shall ignore it.

nnpfc_mode_idc equal to 0 specifies that the post-processing filter associated with the nnpfc_id value is determined by external means not specified in this Specification.

nnpfc_mode_idc equal to 1 specifies that the post-processing filter associated with the nnpfc_id value is a neural network represented by the ISO/IEC 15938-17 bitstream contained in this SEI message.

nnpfc_mode_idc equal to 2 specifies that the post-processing filter associated with the nnpfc_id value is a neural network identified by a specified tag Uniform Resource Identifier (URI) (nnpfc_uri_tag[i]) and neural network information URI (nnpfc_uri[i]).

The value of nnpfc_mode_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_mode_idc greater than 2 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_mode_idc.

nnpfc_purpose_and_formatting_flag equal to 0 specifies that no syntax elements related to the filter purpose, input formatting, output formatting, and complexity are present. nnpfc_purpose_and_formatting_flag equal to 1 specifies that syntax elements related to the filter purpose, input formatting, output formatting, and complexity are present.

When nnpfc_mode_idc is equal to 1 and the current CLVS does not contain a preceding neural-network post-filter characteristics SEI message, in decoding order, that has the value of nnpfc_id equal to the value of nnpfc_id in this SEI message, nnpfc_purpose_and_formatting_flag shall be equal to 1.

When the current CLVS contains a preceding neural-network post-filter characteristics SEI message, in decoding order, that has the same value of nnpfc_id equal to the value of nnpfc_id in this SEI message, at least one of the following conditions shall apply:

- This SEI message has nnpfc_mode_idc equal to 1 and nnpfc_purpose_and_formatting_flag equal to 0 in order to provide a neural network update.

– This SEI message has the same content as the preceding neural-network post-filter characteristics SEI message.

When this SEI message is the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within the current CLVS, it specifies a base post-processing filter that pertains to the current decoded picture and all subsequent decoded pictures of the current layer, in output order, until the end of the current CLVS. When this SEI message is not the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within the current CLVS, this SEI message pertains to the current decoded picture and all subsequent decoded pictures of the current layer, in output order, until the end of the current CLVS or the next neural-network post-filter characteristics SEI message having that particular nnpfc_id value, in output order, within the current CLVS.

nnpfc_purpose indicates the purpose of post-processing filter as specified in Table 1. The value of nnpfc_purpose shall be in the range of 0 to $2^{32} - 2$, inclusive. Values of nnpfc_purpose that do not appear in Table 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_purpose.

When SubWidthC is equal to 1 and SubHeightC is equal to 1, nnpfc_purpose shall not be equal to 2 or 4.

nnpfc_out_sub_c_flag equal to 1 specifies that outSubWidthC is equal to 1 and outSubHeightC is equal to 1. nnpfc_out_sub_c_flag equal to 0 specifies that outSubWidthC is equal to 2 and outSubHeightC is equal to 1. When nnpfc_out_sub_c_flag is not present, outSubWidthC is inferred to be equal to SubWidthC and outSubHeightC is inferred to be equal to SubHeightC. If SubWidthC is equal to 2 and SubHeightC is equal to 1, nnpfc_out_sub_c_flag shall not be equal to 0.

nnpfc_pic_width_in_luma_samples and **nnpfc_pic_height_in_luma_samples** specify the width and height, respectively, of the luma sample array of the picture resulting by applying the post-processing filter identified by nnpfc_id to a cropped decoded output picture. When nnpfc_pic_width_in_luma_samples and nnpfc_pic_height_in_luma_samples are not present, they are inferred to be equal to CroppedWidth and CroppedHeight, respectively.

nnpfc_component_last_flag equal to 0 specifies that the second dimension in the input tensor inputTensor to the post-processing filter and the output tensor outputTensor resulting from the post-processing filter is used for the channel. nnpfc_component_last_flag equal to 1 specifies that the last dimension in the input tensor inputTensor to the post-processing filter and the output tensor outputTensor resulting from the post-processing filter is used for the channel.

NOTE 2 – The first dimension in the input tensor and in the output tensor is used for the batch index, which is a practice in some neural network frameworks. While the semantics of this SEI message use batch size equal to 1, it is up to the post-processing implementation to determine the batch size used as input to the neural network inference.

NOTE 3 – A colour component is an example of a channel.

nnpfc_inp_format_flag indicates the method of converting a sample value of the cropped decoded output picture to an input value to the post-processing filter. When nnpfc_inp_format_flag is equal to 0, the input values to the post-processing filter are real numbers and the functions InpY and InpC are specified as follows:

$$InpY(x) = x \div ((1 << BitDepth_Y) - 1) \quad (75)$$

$$InpC(x) = x \div ((1 << BitDepth_C) - 1) \quad (76)$$

When nnpfc_inp_format_flag is equal to 1, the input values to the post-processing filter are unsigned integer numbers and the functions InpY and InpC are specified as follows:

```
shift = BitDepthY − inpTensorBitDepth
if( inpTensorBitDepth >= BitDepthY)
    InpY( x ) = x << ( inpTensorBitDepth − BitDepthY )
else
    InpY( x ) = Clip3(0, ( 1 << inpTensorBitDepth ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )                                    (77)
shift = BitDepthC − inpTensorBitDepth
if( inpTensorBitDepth >= BitDepthC )
    InpC( x ) = x << ( inpTensorBitDepth − BitDepthC )
else
    InpC( x ) = Clip3(0, ( 1 << inpTensorBitDepth ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )
```

The variable inpTensorBitDepth is derived from the syntax element nnpfc_inp_tensor_bitdepth_minus8 as specified below.

nnpfc_inp_tensor_bitdepth_minus8 plus 8 specifies the bit depth of luma sample values in the input integer tensor. The value of inpTensorBitDepth is derived as follows:

$$inpTensorBitDepth = nnpfc_inp_tensor_bitdepth_minus8 + 8 \quad (78)$$

It is a requirement of bitstream conformance that the value of nnpfc_inp_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

nnpfc_auxiliary_inp_idc not equal to 0 specifies auxiliary input data is present in the input tensor of the neural-network post-filter. nnpfc_auxiliary_inp_idc equal to 0 indicates that auxiliary input data is not present in the input tensor. nnpfc_auxiliary_inp_idc equal to 1, 2, or 3 specifies that auxiliary input data is derived as specified in Table 4. The value of nnpfc_auxiliary_inp_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_auxiliary_inp_idc greater than ~~1~~ 3 are reserved for future specification by ITU-T |

ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_auxiliary_inp_idc.

nnpfc_separate_colour_description_present_flag equal to 1 indicates that a distinct combination of colour primaries, transfer characteristics, and matrix coefficients for the picture resulting from the post-processing filter is specified in the SEI message syntax structure. nnfpc_separate_colour_description_present_flag equal to 0 indicates that the combination of colour primaries, transfer characteristics, and matrix coefficients for the picture resulting from the post-processing filter is the same as indicated in VUI parameters for the CLVS.

nnpfc_colour_primaries has the same semantics as specified in clause 7.3 for the vui_colour_primaries syntax element, except as follows:

– nnpfc_colour_primaries specifies the colour primaries of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the colour primaries used for the CLVS.

– When nnpfc_colour_primaries is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_colour_primaries is inferred to be equal to vui_colour_primaries.

nnpfc_transfer_characteristics has the same semantics as specified in clause 7.3 for the vui_transfer_characteristics syntax element, except as follows:

– nnpfc_transfer_characteristics specifies the transfer characteristics of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the transfer characteristics used for the CLVS.

– When nnpfc_transfer_characteristics is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_transfer_characteristics is inferred to be equal to vui_transfer_characteristics.

nnpfc_matrix_coeffs has the same semantics as specified in clause 7.3 for the vui_matrix_coeffs syntax element, except as follows:

– nnpfc_matrix_coeffs specifies the matrix coefficients of the picture resulting from applying the neural-network post-filter specified in the SEI message, rather than the matrix coefficients used for the CLVS.

– When nnpfc_matrix_coeffs is not present in the neural-network post-filter characteristics SEI message, the value of nnpfc_matrix_coeffs is inferred to be equal to vui_matrix_coeffs.

– The values allowed for nnpfc_matrix_coeffs are not constrained by the chroma format of the decoded video pictures that is indicated by the value of ChromaFormatIdc for the semantics of the VUI parameters.

– When nnpfc_matrix_coeffs is equal to 0, nnpfc_out_order_idc shall not be equal to 1 or 3.

nnpfc_inp_order_idc indicates the method of ordering the sample arrays of a cropped decoded output picture as the input to the post-processing filter. Table 8 contains an informative

description of nnpfc_inp_order_idc values. The semantics of nnpfc_inp_order_idc in the range of 0 to ~~3~~ ***9***, inclusive, are specified in Table 9, which specifies a process for deriving the input tensors inputTensor for different values of nnpfc_inp_order_idc and a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors. When the chroma format of the cropped decoded output picture is not 4:2:0, nnpfc_inp_order_idc shall not be equal to ~~3~~ ***7, 8, or 9***. The value of nnpfc_inp_order_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_inp_order_idc greater than ~~3~~ ***9*** are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_inp_order_idc.

Table 8 – Informative description of nnpfc_inp_order_idc values

nnpfc_inp_order_idc	Description
0	When ~~If~~ nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one luma matrix ~~is present in the input tensor~~, thus the number of channels is 1. ~~Otherwise,~~ When nnpfc_auxiliary_inp_idc is ~~not~~ equal to ~~0~~ 1 or 2~~, and~~ the input tensor consists of one luma matrix and one auxiliary input matrix ~~are present~~, thus the number of channels is 2. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one luma matrix and two auxiliary input matrices, thus the number of channels is 3.
1	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one chroma matrix corresponding to the Cb component, thus the number of channels is 1. When nnpfc_auxiliary_inp_idc is equal to 1 or 2, the input tensor consists of one chroma matrix corresponding to the Cb component and one auxiliary input matrix, thus the number of channels is 2. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one chroma matrix corresponding to the Cb component and two auxiliary input matrices, thus the number of channels is 3.
2	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one chroma matrix corresponding to the Cr component, thus the number of channels is 1. When nnpfc_auxiliary_inp_idc is equal to 1 or 2, the input tensor consists of one chroma matrix corresponding to the Cr component and one auxiliary input matrix, thus the number of channels is 2. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one chroma matrix corresponding to the Cr component and two auxiliary input matrices, thus the number of channels is 3.
~~1~~ 3	When ~~If~~ nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of two chroma matrices, thus the number of channels is 2. ~~Otherwise,~~ When nnpfc_auxiliary_inp_idc is ~~not~~ equal to ~~0~~ 1~~, and~~ the input tensor consists of

	two chroma matrices and one auxiliary input matrix ~~are present~~, thus the number of channels is 3. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of two chroma matrices and two auxiliary input matrices, thus the number of channels is 4. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of two chroma matrices and three auxiliary input matrices, thus the number of channels is 5.
4	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one luma and one chroma matrix corresponding to the Cb component, thus the number of channels is 2. When nnpfc_auxiliary_inp_idc is equal to 1, the input tensor consists of one luma matrix, one chroma matrix corresponding to the Cb component and one auxiliary input matrix, thus the number of channels is 3. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of one luma matrix, one chroma matrix corresponding to the Cb component, and two auxiliary input matrices, thus the number of channels is 4. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one luma matrix, one chroma matrix corresponding to the Cb component, and three auxiliary input matrices, thus the number of channels is 5.
5	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one luma and one chroma matrix corresponding to the Cr component, thus the number of channels is 2. When nnpfc_auxiliary_inp_idc is equal to 1, the input tensor consists of one luma matrix, one chroma matrix corresponding to the Cr component and one auxiliary input matrix, thus the number of channels is 3. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of one luma matrix, one chroma matrix corresponding to the Cr component and two auxiliary input matrices, thus the number of channels is 4. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one luma matrix, one chroma matrix corresponding to cr component and three auxiliary input matrices, thus the number of channels is 5.
~~2~~ 6	When ~~If~~ nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of one luma and two chroma matrices ~~are present in the input tensor~~, thus the number of channels is 3. ~~Otherwise,~~ When nnpfc_auxiliary_inp_idc is ~~not~~ equal to ~~0 and~~ 1, the input tensor consists of one luma matrix, two chroma matrices and one auxiliary input matrix ~~are present~~, thus the number of channels is 4. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of one luma matrix, two chroma matrices and three auxiliary input matrices, thus the number of channels is 6. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of one luma matrix, two chroma matrices and four auxiliary input matrices, thus the number of channels is 7.

7	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of four luma matrices and one chroma matrix corresponding to the Cb component, thus the number of channels is 5. When nnpfc_auxiliary_inp_idc is equal to 1, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cb component, and one auxiliary input matrix, thus the number of channels is 6. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cb component and five auxiliary input matrices, thus the number of channels is 10. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cb component and six auxiliary input matrices, thus the number of channels is 11. The luma channels are derived in an interleaved manner as illustrated in Fig. 4. This nnpfc_inp_order_idc value can only be used when the chroma format is 4:2:0.
8	When nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of four luma matrices and one chroma matrix corresponding to the Cr component, thus the number of channels is 5. When nnpfc_auxiliary_inp_idc is equal to 1, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cr component, and one auxiliary input matrix, thus the number of channels is 6. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cr component and five auxiliary input matrices, thus the number of channels is 10. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of four luma matrices, one chroma matrix corresponding to the Cr component and six auxiliary input matrices, thus the number of channels is 11. The luma channels are derived in an interleaved manner as illustrated in Fig. 4. This nnpfc_inp_order_idc value can only be used when the chroma format is 4:2:0.
~~3~~9	When ~~If~~ nnpfc_auxiliary_inp_idc is equal to 0, the input tensor consists of four luma matrices and two chroma matrices ~~are present in the input tensor~~, thus the number of channels is 6. ~~Otherwise,~~ When nnpfc_auxiliary_inp_idc is ~~not~~ equal to 1, ~~0 and~~ the input tensor consists of four luma matrices, two chroma matrices, and one auxiliary input matrix ~~are present in the input tensor~~, thus the number of channels is 7. When nnpfc_auxiliary_inp_idc is equal to 2, the input tensor consists of four luma matrices, two chroma matrices, and six auxiliary input matrices, thus the number of channels is 12. When nnpfc_auxiliary_inp_idc is equal to 3, the input tensor consists of four luma matrices, two chroma matrices, and seven auxiliary input matrices, thus

	the number of channels is 13. The luma channels are derived in an interleaved manner as illustrated in Fig. 4. This nnpfc_inp_order_idc value can only be used when the chroma format is 4:2:0.
~~4..255~~ 10..255	reserved

A patch is a rectangular array of samples from a component (e.g., a luma or chroma component) of a picture.

nnpfc_constant_patch_size_flag equal to 0 specifies that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. When nnpfc_constant_patch_size_flag is equal to 0 the patch size width shall be less than or equal to CroppedWidth. When nnpfc_constant_patch_size_flag is equal to 0 the patch size height shall be less than or equal to CroppedHeight. nnpfc_constant_patch_size_flag equal to 1 specifies that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the horizontal sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_width_minus1 + 1) may be used as the horizontal sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the vertical sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_height_minus1 + 1) may be used as the vertical sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

nnpfc_overlap specifies the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

The variables inpPatchWidth, inpPatchHeight, outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:

```
inpPatchWidth = nnpfc_patch_width_minus1 + 1
inpPatchHeight = nnpfc_patch_height_minus1 + 1
outPatchWidth = ( nnpfc_pic_width_in_luma_samples * inpPatchWidth ) /
CroppedWidth
outPatchHeight = ( nnpfc_pic_height_in_luma_samples * inpPatchHeight ) /
CroppedHeight
```

```
horCScaling = SubWidthC / outSubWidthC
verCScaling = SubHeightC / outSubHeightC                    (79)
outPatchCWidth = outPatchWidth * horCScaling
outPatchCHeight = outPatchHeight * verCScaling
overlapSize = nnpfc_overlap
```

It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.

nnpfc_padding_type specifies the process of padding when referencing sample locations outside the boundaries of the cropped decoded output picture as described in Table 3. The value of nnpfc_padding_type shall be in the range of 0 to 15, inclusive.

nnpfc_luma_padding_val specifies the luma value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cb_padding_val specifies the Cb value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cr_padding_val specifies the Cr value to be used for padding when nnpfc_padding_type is equal to 4.

The function InpSampleVal(y, x, picHeight, picWidth, croppedPic) with inputs being a vertical sample location y, a horizontal sample location x, a picture height picHeight, a picture width picWidth, and sample array croppedPic returns the value of sampleVal derived as follows:

```
if( nnpfc_padding_type == 0 )
    if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
        sampleVal = 0
    else
        sampleVal = croppedPic[ x ][ y ]                    (80)
else if( nnpfc_padding_type == 1 )
    sampleVal = croppedPic[ Clip3( 0, picWidth − 1, x ) ][ Clip3( 0, picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 2 )
    sampleVal = croppedPic[ Reflect( picWidth − 1, x ) ][ Reflect( picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 3 )
    if(y >= 0 && y < picHeight)
        sampleVal = croppedPic[ Wrap( picWidth − 1, x ) ][ y ]
else if( nnpfc_padding_type == 4 )
```

```
if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
  sampleVal[ 0 ] = nnpfc_luma_padding_val
      sampleVal[ 1 ] = nnpfc_cb_padding_val
  sampleVal[ 2 ] = nnpfc_cr_padding_val
else
  sampleVal = croppedPic[ x ][ y ]
```

Table 9 – Process for deriving the input tensors inputTensor for a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors

nnpfc_inp_order_idc	Process DeriveInputTensors( ) for deriving input tensors

0	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpVal = InpY(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight, CroppedWidth, CroppedYPic))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpVal
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpVal
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	} else if (nnpfc_auxiliary_inp_idc == 2) {
	inpPred = InpY(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight, CroppedWidth, CroppedYPred))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpPred
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpPred
	} else if (nnpfc_auxiliary_inp_idc == 3) {
	inpPred = InpY(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight, CroppedWidth, CroppedYPred))

	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpPred
	}
	}
	}

1	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpCbVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpCbVal
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpCbVal
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	} else if (nnpfc_auxiliary_inp_idc == 2) {
	inpCbPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCbPred))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCbPred
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCbPred
	} else if (nnpfc_auxiliary_inp_idc == 3) {

	inpCbPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCbPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpCbPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpCbPred
	}
	}
	}

2	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpCrVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpCrVal
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpCrVal
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	} else if (nnpfc_auxiliary_inp_idc == 2) {
	inpCrPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCrPred))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] =

	inpCrPred
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCrPred
	} else if (nnpfc_auxiliary_inp_idc == 3) {
	inpCbPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCbPred))
	inpCrPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpCbPred
	inputTensor[0][4][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpCbPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][4] = inpCrPred
	}
	}
	}

~~1~~3	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpCbVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpCbVal

	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCrVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	} else if (nnpfc_auxiliary_inp_idc == 2) {
	inpCbPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCbPred))
	inpCrPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpCbPred
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpCbPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpCrPred
	}
	} else if (nnpfc_auxiliary_inp_idc == 3) {
	inpCrPred = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6Z}$
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] =

	inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpCrPred
	}
	}
	}

4	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yY = cTop + yP
	xY = cLeft + xP
	yC = yY / SubHeightC
	xC = xY / SubWidthC
	inpYVal = InpY(InpSampleVal(yY, xY, CroppedHeight, CroppedWidth, CroppedYPic))
	inpCbVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC, CroppedWidth / SubWidthC, CroppedCbPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpYVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCbVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpYVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCbVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	} else if (nnpfc_auxiliary_inp_idc == 2) {
	inpYPred = InpY(InpSampleVal(yY, xY, CroppedHeight, CroppedWidth, CroppedYPred))
	inpCbPred = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,

	CroppedWidth / SubWidthC, CroppedCbPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpYPred
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpCbPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpYPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpCbPred
	}
	} else if (nnpfc_auxiliary_inp_idc == 3) {
	inpYPred = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPred))
	inpCbPred = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpYPred
	inputTensor[0][4][yP + overlapSize][xP + overlapSize] = inpCbPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpYPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][4] = inpCbPred
	}
	}
	}

5	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yY = cTop + yP

```
    xY = cLeft + xP
    yC = yY / SubHeightC
    xC = xY / SubWidthC
    inpYVal = InpY( InpSampleVal( yY, xY, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpCrVal = InpC( InpSampleVal( yC, xC, CroppedHeight / SubHeightC,
        CroppedWidth / SubWidthC, CroppedCrPic ) )
    if( nnpfc_component_last_flag == 0 ) {
      inputTensor[ 0 ][ 0 ][ yP + overlapSize ][ xP + overlapSize ] = inpYVal
      inputTensor[ 0 ][ 1 ][ yP + overlapSize ][ xP + overlapSize ] = inpCrVal
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpYVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpCrVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] =
2^(StrengthControlVal – 42)/6
    } else if (nnpfc_auxiliary_inp_idc == 2) {
      inpYPred = InpY( InpSampleVal( yY, xY, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpCrPred = InpC( InpSampleVal( yC, xC, CroppedHeight /
SubHeightC,
        CroppedWidth / SubWidthC, CroppedCrPred ) )
      if( nnpfc_component_last_flag == 0 ) {
        inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] =
inpYPred
        inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCrPred
      } else {
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] =
inpYPred
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] =
inpCrPred
      }
```

	} else if (nnpfc_auxiliary_inp_idc == 3) {
	inpYPred = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPred))
	inpCrPred = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = inpYPred
	inputTensor[0][4][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = inpYPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][4] = inpCrPred
	}
	}
	}

26	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yY = cTop + yP
	xY = cLeft + xP
	yC = yY / SubHeightC
	xC = xY / SubWidthC
	inpYVal = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPic))
	inpCbVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpYVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCbVal

```
      inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] = inpCrVal
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpYVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpCbVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] = inpCrVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] =
2^(StrengthControlVal – 42)/6
    } else if (nnpfc_auxiliary_inp_idc == 2) {
      inpYPred = InpY( InpSampleVal( yY, xY, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpCbPred = InpC( InpSampleVal( yC, xC, CroppedHeight /
SubHeightC,
        CroppedWidth / SubWidthC, CroppedCbPred ) )
      inpCrPred = InpC( InpSampleVal( yC, xC, CroppedHeight /
SubHeightC,
        CroppedWidth / SubWidthC, CroppedCrPred ) )
      if( nnpfc_component_last_flag == 0 ) {
        inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] =
inpYPred
        inputTensor[ 0 ][ 4 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCbPred
        inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCrPred
      } else {
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] =
inpYPred
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 4 ] =
inpCbPred
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] =
inpCrPred
      }
    } else if (nnpfc_auxiliary_inp_idc == 3) {
```

	inpYPred = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPred))
	inpCbPred = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPred))
	inpCrPred = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][4][yP + overlapSize][xP + overlapSize] = inpYPred
	inputTensor[0][5][yP + overlapSize][xP + overlapSize] = inpCbPred
	inputTensor[0][6][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][4] = inpYPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][5] = inpCbPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][6] = inpCrPred
	}
	}
	}

7	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yTL = cTop + yP * 2
	xTL = cLeft + xP * 2
	yBR = yTL + 1
	xBR = xTL + 1
	yC = cTop / 2 + yP
	xC = cLeft / 2 + xP
	inpTLVal = InpY(InpSampleVal(yTL, xTL, CroppedHeight,

```
        CroppedWidth, CroppedYPic ) )
    inpTRVal = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpBLVal = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpBRVal = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpCbVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCbPic ) )
    if( nnpfc_component_last_flag == 0 ) {
      inputTensor[ 0 ][ 0 ][ yP + overlapSize ][ xP + overlapSize ] = inpTLVal
      inputTensor[ 0 ][ 1 ][ yP + overlapSize ][ xP + overlapSize ] = inpTRVal
      inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] = inpBLVal
      inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] = inpBRVal
      inputTensor[ 0 ][ 4 ][ yP + overlapSize ][ xP + overlapSize ] = inpCbVal
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpTLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpTRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] = inpBLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] = inpBRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 4 ] = inpCbVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] =
2^(StrengthControlVal – 42)/6
    } else if (nnpfc_auxiliary_inp_idc == 2) {
      inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpBLPred = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpBRPred = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
```

```
        inpCbPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
          CroppedWidth / 2, CroppedCbPred ) )
        if( nnpfc_component_last_flag == 0 ) {
          inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTLPred
          inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTRPred
          inputTensor[ 0 ][ 7 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBLPred
          inputTensor[ 0 ][ 8 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBRPred
          inputTensor[ 0 ][ 9 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCbPred
        } else {
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] =
inpTLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 6 ] =
inpTRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 7 ] =
inpBLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 8 ] =
inpBRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 9 ] =
inpCbPred
        }
      } else if (nnpfc_auxiliary_inp_idc == 3) {
        inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpBLPred = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpBRPred = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpCbPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
          CroppedWidth / 2, CroppedCbPred ) )
        if( nnpfc_component_last_flag == 0 ) {
          inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
```

	$2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][6][yP + overlapSize][xP + overlapSize] = inpTLPred
	inputTensor[0][7][yP + overlapSize][xP + overlapSize] = inpTRPred
	inputTensor[0][8][yP + overlapSize][xP + overlapSize] = inpBLPred
	inputTensor[0][9][yP + overlapSize][xP + overlapSize] = inpBRPred
	inputTensor[0][10][yP + overlapSize][xP + overlapSize] = inpCbPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][5] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][6] = inpTLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][7] = inpTRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][8] = inpBLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][9] = inpBRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][10] = inpCbPred
	}
	}
	}

8	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yTL = cTop + yP * 2
	xTL = cLeft + xP * 2
	yBR = yTL + 1
	xBR = xTL + 1
	yC = cTop / 2 + yP
	xC = cLeft / 2 + xP
	inpTLVal = InpY(InpSampleVal(yTL, xTL, CroppedHeight, CroppedWidth, CroppedYPic))
	inpTRVal = InpY(InpSampleVal(yTL, xBR, CroppedHeight,

```
        CroppedWidth, CroppedYPic ) )
    inpBLVal = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpBRVal = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpCrVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCrPic ) )
    if( nnpfc_component_last_flag == 0 ) {
      inputTensor[ 0 ][ 0 ][ yP + overlapSize ][ xP + overlapSize ] = inpTLVal
      inputTensor[ 0 ][ 1 ][ yP + overlapSize ][ xP + overlapSize ] = inpTRVal
      inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] = inpBLVal
      inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] = inpBRVal
      inputTensor[ 0 ][ 4 ][ yP + overlapSize ][ xP + overlapSize ] = inpCrVal
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpTLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpTRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] = inpBLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] = inpBRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 4 ] = inpCrVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] =
2^(StrengthControlVal – 42)/6
    } else if (nnpfc_auxiliary_inp_idc == 2) {
      inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpBLPred = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpBRPred = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpCrPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCrPred ) )
```

```
        if( nnpfc_component_last_flag == 0 ) {
          inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTLPred
          inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTRPred
          inputTensor[ 0 ][ 7 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBLPred
          inputTensor[ 0 ][ 8 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBRPred
          inputTensor[ 0 ][ 9 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCrPred
        } else {
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] =
inpTLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 6 ] =
inpTRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 7 ] =
inpBLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 8 ] =
inpBRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 9 ] =
inpCrPred
        }
      } else if (nnpfc_auxiliary_inp_idc == 3) {
        inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpBLPred = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpBRPred = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpCrPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
          CroppedWidth / 2, CroppedCrPred ) )
        if( nnpfc_component_last_flag == 0 ) {
          inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
          inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
```

	inpTLPred
	inputTensor[0][7][yP + overlapSize][xP + overlapSize] = inpTRPred
	inputTensor[0][8][yP + overlapSize][xP + overlapSize] = inpBLPred
	inputTensor[0][9][yP + overlapSize][xP + overlapSize] = inpBRPred
	inputTensor[0][10][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][5] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][6] = inpTLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][7] = inpTRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][8] = inpBLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][9] = inpBRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][10] = inpCrPred
	}
	}
	}

39	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yTL = cTop + yP * 2
	xTL = cLeft + xP * 2
	yBR = yTL + 1
	xBR = xTL + 1
	yC = cTop / 2 + yP
	xC = cLeft / 2 + xP
	inpTLVal = InpY(InpSampleVal(yTL, xTL, CroppedHeight, CroppedWidth, CroppedYPic))
	inpTRVal = InpY(InpSampleVal(yTL, xBR, CroppedHeight, CroppedWidth, CroppedYPic))
	inpBLVal = InpY(InpSampleVal(yBR, xTL, CroppedHeight,

```
        CroppedWidth, CroppedYPic ) )
    inpBRVal = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpCbVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCbPic ) )
    inpCrVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCrPic ) )
    if( nnpfc_component_last_flag == 0 ) {
      inputTensor[ 0 ][ 0 ][ yP + overlapSize ][ xP + overlapSize ] = inpTLVal
      inputTensor[ 0 ][ 1 ][ yP + overlapSize ][ xP + overlapSize ] = inpTRVal
      inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] = inpBLVal
      inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] = inpBRVal
      inputTensor[ 0 ][ 4 ][ yP + overlapSize ][ xP + overlapSize ] = inpCbVal
      inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] = inpCrVal
      inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpTLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpTRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] = inpBLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] = inpBRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 4 ] = inpCbVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] = inpCrVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 6 ] =
2^(StrengthControlVal – 42)/6
    } else if (nnpfc_auxiliary_inp_idc == 2) {
      inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
      inpBLPred = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPred ) )
```

```
        inpBRPred = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpCbPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
          CroppedWidth / 2, CroppedCbPred ) )
        inpCrPred = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
          CroppedWidth / 2, CroppedCrPred ) )
        if( nnpfc_component_last_flag == 0 ) {
          inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTLPred
          inputTensor[ 0 ][ 7 ][ yP + overlapSize ][ xP + overlapSize ] =
inpTRPred
          inputTensor[ 0 ][ 8 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBLPred
          inputTensor[ 0 ][ 9 ][ yP + overlapSize ][ xP + overlapSize ] =
inpBRPred
          inputTensor[ 0 ][ 10 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCbPred
          inputTensor[ 0 ][ 11 ][ yP + overlapSize ][ xP + overlapSize ] =
inpCrPred
        } else {
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 6 ] =
inpTLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 7 ] =
inpTRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 8 ] =
inpBLPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 9 ] =
inpBRPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 10 ] =
inpCbPred
          inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 11 ] =
inpCrPred
        }
      } else if (nnpfc_auxiliary_inp_idc == 3) {
        inpTLPred = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
        inpTRPred = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
          CroppedWidth, CroppedYPred ) )
```

	inpBLPred = InpY(InpSampleVal(yBR, xTL, CroppedHeight, CroppedWidth, CroppedYPred))
	inpBRPred = InpY(InpSampleVal(yBR, xBR, CroppedHeight, CroppedWidth, CroppedYPred))
	inpCbPred = InpC(InpSampleVal(yC, xC, CroppedHeight / 2, CroppedWidth / 2, CroppedCbPred))
	inpCrPred = InpC(InpSampleVal(yC, xC, CroppedHeight / 2, CroppedWidth / 2, CroppedCrPred))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][6][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][7][yP + overlapSize][xP + overlapSize] = inpTLPred
	inputTensor[0][8][yP + overlapSize][xP + overlapSize] = inpTRPred
	inputTensor[0][9][yP + overlapSize][xP + overlapSize] = inpBLPred
	inputTensor[0][10][yP + overlapSize][xP + overlapSize] = inpBRPred
	inputTensor[0][11][yP + overlapSize][xP + overlapSize] = inpCbPred
	inputTensor[0][12][yP + overlapSize][xP + overlapSize] = inpCrPred
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][6] = $2^{(StrengthControlVal - 42)/6}$
	inputTensor[0][yP + overlapSize][xP + overlapSize][7] = inpTLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][8] = inpTRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][9] = inpBLPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][10] = inpBRPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][11] = inpCbPred
	inputTensor[0][yP + overlapSize][xP + overlapSize][12] = inpCrPred

	} } }
4..255	reserved

nnpfc_complexity_idc greater than 0 specifies that one or more syntax elements that indicate the complexity of the post-processing filter associated with the nnpfc_id may be present. nnpfc_complexity_idc equal to 0 specifies that no syntax element that indicates the complexity of the post-processing filter associated with the nnpfc_id is present. The value nnpfc_complexity_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_complexity_idc greater than 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_complexity_idc.

nnpfc_out_format_flag equal to 0 indicates that the sample values output by the post-processing filter are real numbers and the functions OutY and OutC for converting luma sample values and chroma sample values, respectively, output by the post-processing, to integer values at bit depths $BitDepth_Y$ and $BitDepth_C$, respectively, are specified as follows:

$$OutY(x) = Clip3(0, (1 << BitDepth_Y) - 1, Round(x * ((1 << BitDepth_Y) - 1))) \quad (81)$$

$$OutC(x) = Clip3(0, (1 << BitDepth_C) - 1, Round(x * ((1 << BitDepth_C) - 1))) \quad (82)$$

nnpfc_out_format_flag equal to 1 indicates that the sample values output by the post-processing filter are unsigned integer numbers and the functions OutY and OutC are specified as follows:

```
shift = outTensorBitDepth − BitDepthY
if( shift > 0 )
    OutY( x ) = Clip3( 0, ( 1 << BitDepthY ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )
else
    OutY( x ) = x << ( BitDepthY − outTensorBitDepth )                    (83)
shift = outTensorBitDepth − BitDepthC
if( shift > 0 )
    OutC( x ) = Clip3( 0, ( 1 << BitDepthC ) − 1, ( x + ( 1 << ( shift −
1 ) ) ) >> shift )
else
```

$$OutC(x) = x << (BitDepth_C - outTensorBitDepth)$$

The variable outTensorBitDepth is derived from the syntax element nnpfc_out_tensor_bitdepth_minus8 as described below.

nnpfc_out_tensor_bitdepth_minus8 plus 8 specifies the bit depth of sample values in the output integer tensor. The value of outTensorBitDepth is derived as follows:

$$outTensorBitDepth = nnpfc_out_tensor_bitdepth_minus8 + 8 \quad (84)$$

It is a requirement of bitstream conformance that the value of nnpfc_out_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

nnpfc_out_order_idc indicates the output order of samples resulting from the post-processing filter. Table 10 contains an informative description of nnpfc_out_order_idc values. The semantics of nnpfc_out_order_idc in the range of 0 to ~~3~~ ***9***, inclusive, are specified in Table 11, which specifies a process for deriving sample values in the filtered output sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic from the output tensors outputTensor for different values of nnpfc_out_order_idc and a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors. When nnpfc_purpose is equal to 2 or 4, nnpfc_out_order_idc shall not be equal to ~~3~~ 7, 8, or 9. The value of nnpfc_out_order_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_out_order_idc greater than ~~3~~ 9 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_out_order_idc.

Table 10 – Informative description of nnpfc_out_order_idc values

nnpfc_out_order_idc	**Description**
0	Only the luma matrix is present in the output tensor, thus the number of channels is 1.
1	Only the chroma matrix corresponding to the Cb component is present in the output tensor, thus the number of channels is 1.
2	Only the chroma matrix corresponding to the Cr component is present in the output tensor, thus the number of channels is 1.
~~1~~3	Only the chroma matrices are present in the output tensor, thus the number of channels is 2.
4	Only the luma and the chroma matrix corresponding to the Cb component are present in the output tensor, thus the number of channels is 2.

5	Only the luma and the chroma matrix corresponding to the Cr component are present in the output tensor, thus the number of channels is 2.
~~2~~6	~~The~~ Only the luma and chroma matrices are present in the output tensor, thus the number of channels is 3.
7	Only four luma matrices and one chroma matrix corresponding to the Cb component are present in the output tensor, thus the number of channels is 5. This nnpfc_out_order_idc value can only be used when the chroma format is 4:2:0.
8	Only four luma matrices and one chroma matrix corresponding to the Cr component are present in the output tensor, thus the number of channels is 5. This nnpfc_out_order_idc value can only be used when the chroma format is 4:2:0.
~~3~~9	~~Four~~ Only four luma matrices and two chroma matrices are present in the output tensor, thus the number of channels is 6. This nnpfc_out_order_idc value can only be used when the chroma format is 4:2:0.
4..255	reserved

Table 11 – Process for deriving sample values in the filtered output sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic from the output tensors outputTensor for a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors

nnpfc_out_order_idc	Process StoreOutputTensors( ) for deriving sample values in the filtered picture from the output tensors

0	for(yP = 0; yP < outPatchHeight; yP++)
	  for(xP = 0; xP < outPatchWidth; xP++) {
	  yY = cTop * outPatchHeight / inpPatchHeight + yP
	  xY = cLeft * outPatchWidth / inpPatchWidth + xP
	  if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	  if(nnpfc_component_last_flag == 0)
	  FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	  else
	  FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	  }

1	for(yP = 0; yP < outPatchCHeight; yP++) for(xP = 0; xP < outPatchCWidth; xP++) {

	xSrc = cLeft * horCScaling + xP
	ySrc = cTop * verCScaling + yP
	if (ySrc < nnpfc_pic_height_in_luma_samples / outSubHeightC && xSrc < nnpfc_pic_width_in_luma_samples / outSubWidthC)
	if(nnpfc_component_last_flag == 0)
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][0][yP][xP])
	else
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][0])
	}

2	for(yP = 0; yP < outPatchCHeight; yP++)
	for(xP = 0; xP < outPatchCWidth; xP++) {
	xSrc = cLeft * horCScaling + xP
	ySrc = cTop * verCScaling + yP
	if (ySrc < nnpfc_pic_height_in_luma_samples / outSubHeightC && xSrc < nnpfc_pic_width_in_luma_samples / outSubWidthC)
	if(nnpfc_component_last_flag == 0)
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][0][yP][xP])
	else
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][0])
	}

~~1~~3	for(yP = 0; yP < outPatchCHeight; yP++)
	for(xP = 0; xP < outPatchCWidth; xP++) {
	xSrc = cLeft * horCScaling + xP
	ySrc = cTop * verCScaling + yP
	if (ySrc < nnpfc_pic_height_in_luma_samples / outSubHeightC && xSrc < nnpfc_pic_width_in_luma_samples / outSubWidthC)
	if(nnpfc_component_last_flag == 0) {
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][0][yP][xP])
	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][1][yP][xP])
	} else {
	FilteredCbPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][0])

	FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][1]) } }

4	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	yY = cTop * outPatchHeight / inpPatchHeight + yP
	xY = cLeft * outPatchWidth / inpPatchWidth + xP
	yC = yY / outSubHeightC
	xC = xY / outSubWidthC
	yPc = (yP / outSubHeightC) * outSubHeightC
	xPc = (xP / outSubWidthC) * outSubWidthC
	if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][1][yPc][xPc])
	} else {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][1])
	}
	}

5	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	yY = cTop * outPatchHeight / inpPatchHeight + yP
	xY = cLeft * outPatchWidth / inpPatchWidth + xP
	yC = yY / outSubHeightC
	xC = xY / outSubWidthC
	yPc = (yP / outSubHeightC) * outSubHeightC
	xPc = (xP / outSubWidthC) * outSubWidthC
	if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][1][yPc][xPc])
	} else {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][1])

	~~}~~ ~~}~~

~~26~~	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	yY = cTop * outPatchHeight / inpPatchHeight + yP
	xY = cLeft * outPatchWidth / inpPatchWidth + xP
	yC = yY / outSubHeightC
	xC = xY / outSubWidthC
	yPc = (yP / outSubHeightC) * outSubHeightC
	xPc = (xP / outSubWidthC) * outSubWidthC
	if (yY < nnpfc_pic_height_in_luma_samples && xY < nnpfc_pic_width_in_luma_samples)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][0][yP][xP])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][1][yPc][xPc])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][2][yPc][xPc])
	} else {
	FilteredYPic[xY][yY] = OutY(outputTensor[0][yP][xP][0])
	FilteredCbPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][1])
	FilteredCrPic[xC][yC] = OutC(outputTensor[0][yPc][xPc][2])
	}
	}

7	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	ySrc = cTop / 2 * outPatchHeight / inpPatchHeight + yP
	xSrc = cLeft / 2 * outPatchWidth / inpPatchWidth + xP
	if (ySrc < nnpfc_pic_height_in_luma_samples / 2 &&
	xSrc < nnpfc_pic_width_in_luma_samples / 2)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][0][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][1][yP][xP])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][2][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] =
	OutY(outputTensor[0][3][yP][xP])
	FilteredCbPic[xSrc][ySrc] =

	OutC(outputTensor[0][4][yP][xP])
	} else {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][0])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][1])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][yP][xP][2])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] =
	OutY(outputTensor[0][yP][xP][3])
	FilteredCbPic[xSrc][ySrc] =
	OutC(outputTensor[0][yP][xP][4])
	}
	}

8	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	ySrc = cTop / 2 * outPatchHeight / inpPatchHeight + yP
	xSrc = cLeft / 2 * outPatchWidth / inpPatchWidth + xP
	if (ySrc < nnpfc_pic_height_in_luma_samples / 2 &&
	xSrc < nnpfc_pic_width_in_luma_samples / 2)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][0][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][1][yP][xP])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][2][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] =
	OutY(outputTensor[0][3][yP][xP])
	FilteredCrPic[xSrc][ySrc] =
	OutC(outputTensor[0][4][yP][xP])
	} else {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][0])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][1])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][yP][xP][2])

	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] = OutY(outputTensor[0][yP][xP][3]) FilteredCrPic[xSrc][ySrc] = OutC(outputTensor[0][yP][xP][4]) } }

39	for(yP = 0; yP < outPatchHeight; yP++)
	for(xP = 0; xP < outPatchWidth; xP++) {
	ySrc = cTop / 2 * outPatchHeight / inpPatchHeight + yP
	xSrc = cLeft / 2 * outPatchWidth / inpPatchWidth + xP
	if (ySrc < nnpfc_pic_height_in_luma_samples / 2 &&
	xSrc < nnpfc_pic_width_in_luma_samples / 2)
	if(nnpfc_component_last_flag == 0) {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][0][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][1][yP][xP])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][2][yP][xP])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] =
	OutY(outputTensor[0][3][yP][xP])
	FilteredCbPic[xSrc][ySrc] =
	OutC(outputTensor[0][4][yP][xP])
	FilteredCrPic[xSrc][ySrc] =
	OutC(outputTensor[0][5][yP][xP])
	} else {
	FilteredYPic[xSrc * 2][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][0])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2] =
	OutY(outputTensor[0][yP][xP][1])
	FilteredYPic[xSrc * 2][ySrc * 2 + 1] =
	OutY(outputTensor[0][yP][xP][2])
	FilteredYPic[xSrc * 2 + 1][ySrc * 2 + 1] =
	OutY(outputTensor[0][yP][xP][3])
	FilteredCbPic[xSrc][ySrc] =
	OutC(outputTensor[0][yP][xP][4])
	FilteredCrPic[xSrc][ySrc] =
	OutC(outputTensor[0][yP][xP][5])

	} }
4..255	reserved

A base post-processing filter for a cropped decoded output picture picA is the filter that is identified by the first neural-network post-filter characteristics SEI message, in decoding order, that has a particular nnpfc_id value within a CLVS.

If there is another neural-network post-filter characteristics SEI message that has the same nnpfc_id value, has nnpfc_mode_idc equal to 1, has different content than the neural-network post-filter characteristics SEI message that defines the base post-processing filter, and pertains to the picture picA, the base post-processing filter is updated by decoding the ISO/IEC 15938-17 bitstream in that neural-network post-filter characteristics SEI message to obtain a post-processing filter PostProcessingFilter( ). Otherwise, the post-processing processing filter PostProcessingFilter( ) is assigned to be the same as the base post-processing filter.

The following process is used to filter the cropped decoded output picture with the post-processing filter PostProcessingFilter( ) to generate the filtered picture, which contains Y, Cb, and Cr sample arrays FilteredYPic, FilteredCbPic, and FilteredCrPic, respectively, as indicated by nnpfc_out_order_idc.

```
if( nnpfc_inp_order_idc == 0 )
    for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight )
      for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth ) {
          DeriveInputTensors( )
          outputTensor = PostProcessingFilter( inputTensor )
          StoreOutputTensors( )
      }
else if( nnpfc_inp_order_idc == 1 )
    for( cTop = 0; cTop < CroppedHeight / SubHeightC; cTop +=
inpPatchHeight )
      for( cLeft = 0; cLeft < CroppedWidth / SubWidthC; cLeft +=
inpPatchWidth ) {
          DeriveInputTensors( )
          outputTensor = PostProcessingFilter( inputTensor )
          StoreOutputTensors( )
      }
else if( nnpfc_inp_order_idc == 2 )
    for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight)          (85)
      for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth) {
          DeriveInputTensors( )
```

```
                outputTensor = PostProcessingFilter( inputTensor )
                StoreOutputTensors( )
            }
        else if( nnpfc_inp_order_idc == 3 )
            for( cTop = 0; cTop < CroppedHeight; cTop += inpPatchHeight * 2 )
              for( cLeft = 0; cLeft < CroppedWidth; cLeft += inpPatchWidth * 2 ) {
                DeriveInputTensors( )
                outputTensor = PostProcessingFilter( inputTensor )
                StoreOutputTensors( )
            }
```

6.3. Embodiment 3

This embodiment is for the case when a video unit is a picture for the embodiment item 4 summarized above in Section 5.

Neural-network post-filter characteristics SEI message semantics

...

nnpfc_inp_order_idc indicates the method of ordering the sample arrays of a cropped decoded output picture as the input to the post-processing filter. Table 2 contains an informative description of nnpfc_inp_order_idc values. The semantics of nnpfc_inp_order_idc in the range of 0 to 3, inclusive, are specified in Table 12, which specifies a process for deriving the input tensors inputTensor for different values of nnpfc_inp_order_idc and a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors. When the chroma format of the cropped decoded output picture is not 4:2:0, nnpfc_inp_order_idc shall not be equal to 3. The value of nnpfc_inp_order_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_inp_order_idc greater than 3 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_inp_order_idc.

A patch is a rectangular array of samples from a component (e.g., a luma or chroma component) of a picture.

nnpfc_constant_patch_size_flag equal to 0 specifies that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. When nnpfc_constant_patch_size_flag is equal to 0 the patch size width shall be less than or equal to CroppedWidth. When nnpfc_constant_patch_size_flag is equal to 0 the patch size height shall be less than or equal to CroppedHeight. nnpfc_constant_patch_size_flag equal to 1 specifies that the post-processing filter accepts exactly the patch size indicated by

nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the horizontal sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_width_minus1 + 1) may be used as the horizontal sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the vertical sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_height_minus1 + 1) may be used as the vertical sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

nnpfc_overlap specifies the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

The variables inpPatchWidth, inpPatchHeight, outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:

```
inpPatchWidth = nnpfc_patch_width_minus1 + 1
inpPatchHeight = nnpfc_patch_height_minus1 + 1
outPatchWidth = ( nnpfc_pic_width_in_luma_samples * inpPatchWidth ) /
CroppedWidth
outPatchHeight = ( nnpfc_pic_height_in_luma_samples * inpPatchHeight ) /
CroppedHeight
horCScaling = SubWidthC / outSubWidthC
verCScaling = SubHeightC / outSubHeightC                    (79)
outPatchCWidth = outPatchWidth * horCScaling
outPatchCHeight = outPatchHeight * verCScaling
overlapSize = nnpfc_overlap
```

It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.

nnpfc_padding_type specifies the process of padding when referencing sample locations outside the boundaries of the cropped decoded output picture as described in Table 3. The value of nnpfc_padding_type shall be in the range of 0 to 15, inclusive.

nnpfc_luma_padding_val specifies the luma value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cb_padding_val specifies the Cb value to be used for padding when nnpfc_padding_type is equal to 4.

nnpfc_cr_padding_val specifies the Cr value to be used for padding when nnpfc_padding_type is equal to 4.

The function InpSampleVal(y, x, picHeight, picWidth, croppedPic) with inputs being a vertical sample location y, a horizontal sample location x, a picture height picHeight, a picture width picWidth, and sample array croppedPic returns the value of sampleVal derived as follows:

```
if( nnpfc_padding_type == 0 )
    if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
      sampleVal = 0
    else
      sampleVal = croppedPic[ x ][ y ]                                    (80)
else if( nnpfc_padding_type == 1 )
    sampleVal = croppedPic[ Clip3( 0, picWidth − 1, x ) ][ Clip3( 0, picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 2 )
    sampleVal = croppedPic[ Reflect( picWidth − 1, x ) ][ Reflect( picHeight − 1,
y ) ]
else if( nnpfc_padding_type == 3 )
    if(y >= 0 && y < picHeight)
      sampleVal = croppedPic[ Wrap( picWidth − 1, x ) ][ y ]
else if( nnpfc_padding_type == 4 )
    if( y < 0 || x < 0 || y >= picHeight || x >= picWidth )
      sampleVal[ 0 ] = nnpfc_luma_padding_val
          sampleVal[ 1 ] = nnpfc_cb_padding_val
      sampleVal[ 2 ] = nnpfc_cr_padding_val
    else
      sampleVal = croppedPic[ x ][ y ]
```

Table 12 – Process for deriving the input tensors inputTensor for a given vertical sample coordinate cTop and a horizontal sample coordinate cLeft specifying the top-left sample location for the patch of samples included in the input tensors

nnpfc_inp_order_idc	Process DeriveInputTensors( ) for deriving input tensors
0	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++) for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {

	inpVal = InpY(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight,
	CroppedWidth, CroppedYPic))
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpVal
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpVal
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

1	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	inpCbVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(cTop + yP, cLeft + xP, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpCbVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCrVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = $2^{(StrengthControlVal - 42)/6}$
	}

	}

2	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yY = cTop + yP
	xY = cLeft + xP
	yC = yY / SubHeightC
	xC = xY / SubWidthC
	inpYVal = InpY(InpSampleVal(yY, xY, CroppedHeight,
	CroppedWidth, CroppedYPic))
	inpCbVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCbPic))
	inpCrVal = InpC(InpSampleVal(yC, xC, CroppedHeight / SubHeightC,
	CroppedWidth / SubWidthC, CroppedCrPic))
	if(nnpfc_component_last_flag == 0) {
	inputTensor[0][0][yP + overlapSize][xP + overlapSize] = inpYVal
	inputTensor[0][1][yP + overlapSize][xP + overlapSize] = inpCbVal
	inputTensor[0][2][yP + overlapSize][xP + overlapSize] = inpCrVal
	} else {
	inputTensor[0][yP + overlapSize][xP + overlapSize][0] = inpYVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][1] = inpCbVal
	inputTensor[0][yP + overlapSize][xP + overlapSize][2] = inpCrVal
	}
	if(nnpfc_auxiliary_inp_idc == 1) {
	if(nnpfc_component_last_flag == 0)
	inputTensor[0][3][yP + overlapSize][xP + overlapSize] = $2^{(StrengthControlVal - 42)/6}$
	else
	inputTensor[0][yP + overlapSize][xP + overlapSize][3] = $2^{(StrengthControlVal - 42)/6}$
	}
	}

3	for(yP = −overlapSize; yP < inpPatchHeight + overlapSize; yP++)
	for(xP = −overlapSize; xP < inpPatchWidth + overlapSize; xP++) {
	yTL = cTop + yP * 2
	xTL = cLeft + xP * 2
	yBR = yTL + 1
	xBR = xTL + 1
	yC = cTop / 2 + yP

```
    xC = cLeft / 2 + xP
    inpTLVal = InpY( InpSampleVal( yTL, xTL, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpTRVal = InpY( InpSampleVal( yTL, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpBLVal = InpY( InpSampleVal( yBR, xTL, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpBRVal = InpY( InpSampleVal( yBR, xBR, CroppedHeight,
        CroppedWidth, CroppedYPic ) )
    inpCbVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCbPic ) )
    inpCrVal = InpC( InpSampleVal( yC, xC, CroppedHeight / 2,
        CroppedWidth / 2, CroppedCrPic ) )
    if( nnpfc_component_last_flag == 0 ) {
      inputTensor[ 0 ][ 0 ][ yP + overlapSize ][ xP + overlapSize ] = inpTLVal
      inputTensor[ 0 ][ 1 ][ yP + overlapSize ][ xP + overlapSize ] = inpTRVal
      inputTensor[ 0 ][ 2 ][ yP + overlapSize ][ xP + overlapSize ] = inpBLVal
      inputTensor[ 0 ][ 3 ][ yP + overlapSize ][ xP + overlapSize ] = inpBRVal
      inputTensor[ 0 ][ 4 ][ yP + overlapSize ][ xP + overlapSize ] = inpCbVal
      inputTensor[ 0 ][ 5 ][ yP + overlapSize ][ xP + overlapSize ] = inpCrVal
      inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] = 2^(StrengthControlVal - 42)/6
    } else {
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 0 ] = inpTLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 1 ] = inpTRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 2 ] = inpBLVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 3 ] = inpBRVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 4 ] = inpCbVal
      inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 5 ] = inpCrVal
    }
    if(nnpfc_auxiliary_inp_idc == 1) {
      if( nnpfc_component_last_flag == 0 )
        inputTensor[ 0 ][ 6 ][ yP + overlapSize ][ xP + overlapSize ] = 2^(StrengthControlVal – 42)/6
      else
        inputTensor[ 0 ][ yP + overlapSize ][ xP + overlapSize ][ 6 ] = 2^(StrengthControlVal – 42)/6
    }
```

	}
4..255	reserved

nnpfc_complexity_idc greater than 0 specifies that one or more syntax elements that indicate the complexity of the post-processing filter associated with the nnpfc_id may be present. nnpfc_complexity_idc equal to 0 specifies that no syntax element that indicates the complexity of the post-processing filter associated with the nnpfc_id is present. The value nnpfc_complexity_idc shall be in the range of 0 to 255, inclusive. Values of nnpfc_complexity_idc greater than 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_complexity_idc.

6.4. Embodiment 4

This embodiment is for the case when a video unit is a picture for the embodiment item 5 and its subitems summarized above in Section 5.

Neural-network post-filter characteristics SEI message semantics

...

nnpfc_constant_patch_size_flag equal to 0 specifies that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. When nnpfc_constant_patch_size_flag is equal to 0 the patch size width shall be less than or equal to CroppedWidth. When nnpfc_constant_patch_size_flag is equal to 0 the patch size height shall be less than or equal to CroppedHeight. nnpfc_constant_patch_size_flag equal to 1 specifies that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the horizontal sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of (nnpfc_patch_width_minus1 + 1) may be used as the horizontal sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, specifies the vertical sample counts of the patch size required for the input to the post-processing filter. When nnpfc_constant_patch_size_flag is equal to 0, any positive integer multiple of

(nnpfc_patch_height_minus1 + 1) may be used as the vertical sample counts of the patch size used for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

Let the variables inpPatchWidth and inpPatchHeight be the patch size width and the patch size height, respectively.

If nnpfc_constant_patch_size_flag is equal to 0, the following applies:

– The values of inpPatchWidth and inpPatchHeight are provided by external means not specified in this document. For example, such external means may be an API that passes the values of inpPatchWidth and inpPatchHeight to the decoder and render entity in a video application system, and the values may be configured by a user through a user interface of the application.

– The value of inpPatchWidth shall be a positive integer multiple of nnpfc_patch_width_minus1 + 1 and shall be less than or equal to CroppedWidth. The value of inpPatchHeight shall be a positive integer multiple of nnpfc_patch_height_minus1 + 1 and shall be less than or equal to CroppedHeight.

Otherwise (nnpfc_constant_patch_size_flag is equal to 1), the value of inpPatchWidth is set equal to nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

nnpfc_overlap specifies the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

The variables ~~inpPatchWidth, inpPatchHeight,~~ outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:

~~inpPatchWidth = nnpfc_patch_width_minus1 + 1~~
~~inpPatchHeight = nnpfc_patch_height_minus1 + 1~~
outPatchWidth = (nnpfc_pic_width_in_luma_samples * inpPatchWidth) / CroppedWidth
outPatchHeight = (nnpfc_pic_height_in_luma_samples * inpPatchHeight) / CroppedHeight
horCScaling = SubWidthC / outSubWidthC
verCScaling = SubHeightC / outSubHeightC
(79)
outPatchCWidth = outPatchWidth * horCScaling
outPatchCHeight = outPatchHeight * verCScaling
overlapSize = nnpfc_overlap

It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.

6.5 Embodiment 5

This embodiment is for the case when a video unit is a picture for the invention item 1 and all its subitems summarized above in Section 5.

Neural-network post-filter characteristics SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
nnpfc_id	ue(v)
nnpfc_mode_idc	ue(v)
nnpfc_purpose_and_formatting_flag	u(1)
if(nnpfc_purpose_and_formatting_flag) {	
nnpfc_purpose	ue(v)
if(nnpfc_purpose == 2 || nnpfc_purpose == 4)	
nnpfc_out_sub_c_flag	u(1)
if(nnpfc_purpose == 3 || nnpfc_purpose == 4) {	
nnpfc_pic_width_in_luma_samples	ue(v)
nnpfc_pic_height_in_luma_samples	ue(v)
if(nnpfc_purpose == 5)	
nnpfc_machine_vision_task_type	ue(v)
}	
...	
}	
...	

Neural-network post-filter activation SEI message semantics

...

nnpfc_purpose indicates the purpose of post-processing filter as specified in Table 20. The value of nnpfc_purpose shall be in the range of 0 to $2^{32} - 2$, inclusive. Values of nnpfc_purpose that do not appear in Table 13 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_purpose.

Table 13 – Definition of nnpfc_purpose

Value	Interpretation
0	Unknown or unspecified
1	Visual quality improvement
2	Chroma upsampling from the 4:2:0 chroma format to the 4:2:2 or 4:4:4 chroma format, or from the 4:2:2 chroma format to the 4:4:4 chroma format
3	Increasing the width or height of the cropped decoded output picture without changing the chroma format
4	Increasing the width or height of the cropped decoded output picture and upsampling the chroma format
5	Machine vision tasks
6	Style transfer
7	Object removal task

nnpfc_machine_vision_task_type indicates the type of machine vision task as specified in Table 14. The value of nnpfc_machine_vision_task_type shall be in the range of 0 to 255, inclusive. Values of nnpfc_machine_vision_task_type that do not appear in Table 21-x are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification encountering a value of nnpfc_machine_vision_task_type greater than 3 shall ignore it.

Table 14 – Definition of nnpfc_machine_vision_task_type

Value	Interpretation
0	Enhancement task, targeting at enhancing pictures to provide better input for other processing techniques.
1	Analysis task, targeting at changing the representation of an image into something that is more meaningful and easier to analyze.

NOTE x – When a reserved value of nnpfc_machine_vision_task_type is taken into use in the future by ITU-T | ISO/IEC, the syntax of this SEI message could be extended with syntax elements whose presence is conditioned by nnpfc_machine_vision_task_type being equal to that value.

6.6 Embodiment 6

This embodiment is for the case when a video unit is a picture for the invention item 3 and all its subitems summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_activation(payloadSize) {	**Descriptor**
nnpfa_id	ue(v)

nnpfa_output_component	ue(v)
}	

Neural-network post-filter activation SEI message semantics

This SEI message specifies the neural-network post-processing filter that may be used for post-processing filtering for the current picture.

The neural-network post-processing filter activation SEI message persists only for the current picture.

NOTE – There may be several neural-network post-processing filter activation SEI messages present for the same picture, for example, when the post-processing filters are meant for different purposes or filter different colour components.

nnpfa_id specifies that the neural-network post-processing filter specified by one or more neural-network post-processing filter characteristics SEI messages that pertain to the current picture and have nnpfc_id equal to nnfpa_id may be used for post-processing filtering for the current picture.

nnpfa_output_component indicates the adopted color component of output tensor as specified in Table 15. The value of nnpfa_output_component shall be in the range of 0 to 6, inclusive.

Table 15 – **Definition of nnpfa_output_component**

Value	Interpretation
0	All the components present in the output tensor will be used.
1	Both Cb and Cr components are used in the output tensor
2	Only Cb components are used in the output tensor
3	Only Cr components are used in the output tensor
4	Only luma components are used in the output tensor
5	Both luma components and Cb components are used in the output tensor
6	Both luma components and Cr components are used in the output tensor

6.7 Embodiment 7

This embodiment is for the case when a video unit is a picture for the invention item 1 and all its subitems summarized above in Section 5.

Neural-network post-filter characteristics SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
nnpfc_id	ue(v)
nnpfc_mode_idc	ue(v)
nnpfc_purpose_and_formatting_flag	u(1)

if(nnpfc_purpose_and_formatting_flag) {	
nnpfc_purpose	ue(v)
if(nnpfc_purpose == 1)	
nnpfc_visual_information_processing_type	ue(v)
if(nnpfc_purpose == 2 || nnpfc_purpose == 4)	
nnpfc_out_sub_c_flag	u(1)
if(nnpfc_purpose == 3 || nnpfc_purpose == 4) {	
nnpfc_pic_width_in_luma_samples	ue(v)
nnpfc_pic_height_in_luma_samples	ue(v)
}	
...	
}	
...	
}	

Neural-network post-filter activation SEI message semantics

...

nnpfc_purpose indicates the purpose of post-processing filter as specified in Table 20. The value of nnpfc_purpose shall be in the range of 0 to $2^{32} - 2$, inclusive. Values of nnpfc_purpose that do not appear in Table 16 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification shall ignore SEI messages that contain reserved values of nnpfc_purpose.

Table 16 – Definition of nnpfc_purpose

Value	Interpretation
0	Unknown or unspecified
1	~~Visual quality improvement~~ Visual_information_processing
2	Chroma upsampling from the 4:2:0 chroma format to the 4:2:2 or 4:4:4 chroma format, or from the 4:2:2 chroma format to the 4:4:4 chroma format
3	Increasing the width or height of the cropped decoded output picture without changing the chroma format
4	Increasing the width or height of the cropped decoded output picture and upsampling the chroma format

NOTE – When a reserved value of nnpfc_purpose is taken into use in the future by ITU-T | ISO/IEC,

the syntax of this SEI message could be extended with syntax elements whose presence is conditioned by nnpfc_purpose being equal to that value.

When SubWidthC is equal to 1 and SubHeightC is equal to 1, nnpfc_purpose shall not be equal to 2 or 4.

nnpfc_visual_information_processing_type indicates the type of visual quality improvement as specified in Table 17. The value of nnpfc_visual_information_processing_type shall be in the range of 0 to 255, inclusive. Values of nnpfc_visual_information_processing_type that do not appear in Table 21-x are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders conforming to this version of this Specification encountering a value of nnpfc_visual_information_processing_type greater than 3 shall ignore it.

Table 17 – Definition of nnpfc_visual_information_processing_type

Value	Interpretation
0	General visual quality improvement, targeting at increasing either the fidelity or the subjective visual quality of the reconstructed picture after applying the neural-network post-processing filter. The improvement may be measured by any objective or subjective metric.
1	Objective-oriented/fidelity-oriented visual quality improvement, targeting at increasing the fidelity of the reconstructed picture after applying the neural-network post-processing filter. The fidelity may be measured by PSNR, Ms-SSIM etc.
2	Subjective-oriented visual quality improvement, targeting at increasing the subjective visual quality of the reconstructed picture after applying the neural-network post-processing filter. The subjective visual quality may be measured by LPIPS or MOS. The dehazing and deraining tasks could be treated as subjective-oriented visual quality improvement.
3	Film grain-oriented visual quality improvement, with synthesizing filter grain on the reconstructed picture after applying the neural-network post-processing filter.
4	Machine vision-oriented processing and the purpose is to improve the performance of general machine vision tasks, such as image/video understanding/recognition/detect/segmentation.
5	Change the appearance or visual style of input image/video. The style transfer task is included in this type.

NOTE x – When a reserved value of nnpfc_visual_information_processing_type is taken into use in the future by ITU-T | ISO/IEC, the syntax of this SEI message could be extended with syntax elements whose presence is conditioned by nnpfc_visual_information_processing_type being equal to that value.

6.8 Embodiment 8

This embodiment is for the case when a video unit is a picture for the invention item 3 and all its subitems summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_activation(payloadSize) {	**Descriptor**
nnpfa_id	ue(v)
nnpfa_default_output_flag	u(1)
if(!nnpfa_default_output_flag) {	
nnpfa_luma_output_flag	u(1)
nnpfa_Cb_output_flag	u(1)
nnpfa_Cr_output_flag	u(1)
}	
}	

Neural-network post-filter activation SEI message semantics

This SEI message specifies the neural-network post-processing filter that may be used for post-processing filtering for the current picture.

The neural-network post-processing filter activation SEI message persists only for the current picture.

NOTE – There may be several neural-network post-processing filter activation SEI messages present for the same picture, for example, when the post-processing filters are meant for different purposes or filter different colour components.

nnpfa_id specifies that the neural-network post-processing filter specified by one or more neural-network post-processing filter characteristics SEI messages that pertain to the current picture and have nnpfc_id equal to nnfpa_id may be used for post-processing filtering for the current picture.

nnpfa_default_output_flag equal to 1 specifies that all the color components present in the output tensor will be used. nnpfa_default_output_flag equal to 0 indicates that partial color components present in the output tensor will be used.

nnpfa_luma_output_flag equal to 1 specifies that the luma components of the output tensor will be used if it is presented in the output tensor. nnpfa_luma_output_flag equal to 0 specifies that the luma components present in the output tensor will not be used.

nnpfa_Cb_output_flag equal to 1 specifies that the Cb components of the output tensor will be used if it is presented in the output tensor. nnpfa_Cb_output_flag equal to 0 specifies that

the Cb components present in the output tensor will not be used.

nnpfa_Cr_output_flag equal to 1 specifies that the Cr components of the output tensor will be used if it is presented in the output tensor. nnpfa_Cr_output_flag equal to 0 specifies that the Cr components present in the output tensor will not be used.

When nnpfa_default_output_flag equal to 0, nnpfa_luma_output_flag, nnpfa_Cb_output_flag, and nnpfa_Cr_output_flag shall not all equal to 1 at the same NNPFA SEI message. When nnpfa_default_output_flag equal to 0, nnpfa_luma_output_flag, nnpfa_Cb_output_flag, and nnpfa_Cr_output_flag shall not all equal to 0 at the same NNPFA SEI message.

6.9. Embodiment 9

This embodiment is for the case when a video unit is a picture for the invention item 6 and its subitems summarized above in Section 5.

Neural-network post-filter characteristics SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
…	
nnpfc_out_format_idc	ue(v)
if(nnpfc_out_format_idc == 1)	
~~**nnpfc_out_tensor_bitdepth_minus8**~~	~~ue(v)~~
nnpfc_luma_out_tensor_bitdepth_minus8	ue(v)
nnpfc_chroma_out_tensor_bitdepth_minus8	ue(v)
…	
}	

Neural-network post-filter characteristics SEI message semantics

...

nnpfc_out_format_idc equal to 0 indicates that the sample values output by the post-processing filter are real numbers where the value range of 0 to 1, inclusive, maps linearly to the unsigned integer value range of 0 to (1 << bitDepth) – 1, inclusive, for any desired bit depth bitDepth for subsequent post-processing or displaying.

~~nnpfc_out_format_flag equal to 1 indicates that the sample values output by the post-processing filter are unsigned integer numbers in the range of 0 to (1 << (nnpfc_out_tensor_bitdepth_minus8 + 8)) – 1, inclusive.~~

nnpfc_out_format_flag equal to 1 indicates that the luma sample values output by the post-processing filter are unsigned integer numbers in the range of 0 to

(1 << (nnpfc_luma_out_tensor_bitdepth_minus8 + 8)) − 1, inclusive, and the chroma sample values output by the post-processing filter are unsigned integer numbers in the range of 0 to (1 << (nnpfc_luma_out_tensor_bitdepth_minus8 + 8)) − 1, inclusive.

Values of nnpfc_out_format_idc greater than 1 are reserved for future specification by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders conforming to this edition of this document shall ignore NNPFC SEI messages that contain reserved values of nnpfc_out_format_idc.

~~**nnpfc_out_tensor_bitdepth_minus8** plus 8 specifies the bit depth of sample values in the output integer tensor. The value of nnpfc_out_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.~~

nnpfc_luma_out_tensor_bitdepth_minus8 plus 8 specifies the bit depth of sample values in the output integer tensor. The value of nnpfc_out_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

nnpfc_chroma_out_tensor_bitdepth_minus8 plus 8 specifies the bit depth of sample values in the output integer tensor. The value of nnpfc_out_tensor_bitdepth_minus8 shall be in the range of 0 to 24, inclusive.

…

6.10. Embodiment 10

This embodiment is for the case when a video unit is a picture for the invention items 7.a and 7.b summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
…	ue(v)
nnpfc_constant_patch_size_flag	u(1)
if(nnpfc_constant_patch_size_flag) {	
nnpfc_patch_width_minus_two_overlapsize	ue(v)
nnpfc_patch_height_minus_two_overlapsize	ue(v)
}	
else {	
nnpfc_patch_width_minus1	ue(v)
nnpfc_patch_height_minus1	ue(v)
}	

…	

…

Neural-network post-filter activation SEI message semantics

…

nnpfc_constant_patch_size_flag equal to 1 indicates that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus_two_overlapsize ~~nnpfc_patch_width_minus1~~ and nnpfc_patch_height_minus_two_overlapsize ~~nnpfc_patch_height_minus1~~ as input. nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input.

nnpfc_patch_width_minus_two_overlapsize + 2 * overlapSize, when nnpfc_constant_patch_size_flag equal to 1, indicates the horizontal sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_width_minus_two_overlapsize shall be in the range of 0 to Min(32767, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus_two_overlapsize + 2 * overlapSize, when nnpfc_constant_patch_size_flag equal to 1, indicates the vertical sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_height_minus_two_overlapsize shall be in the range of 0 to Min(32767, CroppedHeight − 1), inclusive.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 0 ~~+~~, multipling a positive integer indicates the horizontal sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 0 ~~+~~, multipling a positive integer indicates the vertical sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

Let the variables inpPatchWidth and inpPatchHeight be the patch size width and the patch size height, respectively.

If nnpfc_constant_patch_size_flag is equal to 0, the following applies:

- The values of inpPatchWidth and inpPatchHeight are either provided by external means not specified in this document or set by the post-processor itself.
- The value of inpPatchWidth + 2 * overlapSize shall be a positive integer multiple of nnpfc_patch_width_minus1 + 1 and shall be less than or equal to CroppedWidth. The value of inpPatchHeight + 2 * overlapSize shall be a positive integer multiple of nnpfc_patch_height_minus1 + 1 and shall be less than or equal to CroppedHeight.

Otherwise (nnpfc_constant_patch_size_flag is equal to 1), the value of inpPatchWidth is set

equal to nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

…

6.11. Embodiment 11

This embodiment is for the case when a video unit is a picture for the invention items 7.c to 7.e summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
…	ue(v)
nnpfc_constant_patch_size_flag	u(1)
~~**nnpfc_patch_width_minus1**~~	~~ue(v)~~
~~**nnpfc_patch_height_minus1**~~	~~ue(v)~~
nnpfc_overlap	ue(v)
nnpfc_delta_patch_width	ue(v)
nnpfc_delta_patch_height	ue(v)
…	

…

Neural-network post-filter activation SEI message semantics

…

nnpfc_overlap indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

The variables outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:

outPatchWidth = (nnpfc_pic_width_in_luma_samples * inpPatchWidth) / CroppedWidth (84)

outPatchHeight = (nnpfc_pic_height_in_luma_samples * inpPatchHeight) / CroppedHeight (85)

horCScaling = SubWidthC / outSubWidthC (86)

verCScaling = SubHeightC / outSubHeightC (87)

outPatchCWidth = outPatchWidth * horCScaling (88)

outPatchCHeight = outPatchHeight * verCScaling (89)

overlapSize = nnpfc_overlap (90)

It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.

nnpfc_constant_patch_size_flag equal to 1 indicates that the post-processing filter accepts exactly the patch size indicated by nnpfc_delta_patch_width and nnpfc_delta_patch_height ~~nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1~~ as input.

nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts any patch size that is a positive integer multiple of the patch size indicated by nnpfc_delta_patch_width and nnpfc_delta_patch_height ~~nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1~~ as input.

~~**nnpfc_patch_width_minus1 + 1,** **nnpfc_delta_patch_width** + 2 * overlapSize, when nnpfc_constant_patch_size_flag equal to 1, indicates the horizontal sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_delta_patch_width ~~nnpfc_patch_width_minus1~~ shall be in the range of 0 to Min(32766 − 2 * overlapSize + 1, CroppedWidth − 2 * overlapSize~~ − 1~~), inclusive.

~~**nnpfc_patch_height_minus1** + 1,~~ **nnpfc_delta_patch_height** + 2 * overlapSize, when nnpfc_constant_patch_size_flag equal to 1, indicates the vertical sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_delta_patch_height ~~nnpfc_patch_height_minus1~~ shall be in the range of 0 to Min(32766 − 2 * overlapSize + 1, CroppedHeight − 2 * overlapSize~~ − 1~~), inclusive.

Let the variables inpPatchWidth and inpPatchHeight be the patch size width and the patch size height, respectively.

If nnpfc_constant_patch_size_flag is equal to 0, the following applies:

– The values of inpPatchWidth and inpPatchHeight are either provided by external means not specified in this document or set by the post-processor itself.

– The value of inpPatchWidth + 2 * overlapSize shall be a positive integer multiple of ~~nnpfc_patch_width_minus1 + 1~~ nnpfc_delta_patch_width + 2 * overlapSize and shall be less than or equal to CroppedWidth. The value of inpPatchHeight + 2 * overlapSize shall

be a positive integer multiple of ~~nnpfc_patch_height_minus1 + 1~~ nnpfc_delta_patch_height + 2 * overlapSize and shall be less than or equal to CroppedHeight.

Otherwise (nnpfc_constant_patch_size_flag is equal to 1), the value of inpPatchWidth is set equal to ~~nnpfc_patch_width_minus1 + 1~~ nnpfc_delta_patch_width + 2 * overlapSize and the value of inpPatchHeight is set equal to ~~nnpfc_patch_height_minus1 + 1~~ nnpfc_delta_patch_height + 2 * overlapSize.

~~**nnpfc_overlap** indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.~~

~~The variables outPatchWidth, outPatchHeight, horCScaling, verCScaling, outPatchCWidth, outPatchCHeight, and overlapSize are derived as follows:~~

~~outPatchWidth = (nnpfc_pic_width_in_luma_samples * inpPatchWidth) / CroppedWidth (84)~~

~~outPatchHeight = (nnpfc_pic_height_in_luma_samples * inpPatchHeight) / CroppedHeight (85)~~

~~horCScaling = SubWidthC / outSubWidthC (86)~~

~~verCScaling = SubHeightC / outSubHeightC (87)~~

~~outPatchCWidth = outPatchWidth * horCScaling (88)~~

~~outPatchCHeight = outPatchHeight * verCScaling (89)~~

~~overlapSize = nnpfc_overlap (90)~~

~~It is a requirement of bitstream conformance that outPatchWidth * CroppedWidth shall be equal to nnpfc_pic_width_in_luma_samples * inpPatchWidth and outPatchHeight * CroppedHeight shall be equal to nnpfc_pic_height_in_luma_samples * inpPatchHeight.~~

nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

...

6.12. Embodiment 12

This embodiment is for the invention item 8 and all its subitems summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
…	ue(v)
nnpfc_overlap	ue(v)
nnpfc_constant_patch_size_flag	u(1)
if(nnpfc_constant_patch_size_flag) {	
nnpfc_patch_width_minus1	ue(v)
nnpfc_patch_height_minus1	ue(v)
} else {	
nnpfc_extended_patch_width_cd_minus1	ue(v)
nnpfc_extended_patch_height_cd_minus1	ue(v)
}	
~~**nnpfc_overlap**~~	~~ue(v)~~
…	

…

Neural-network post-filter activation SEI message semantics

…

nnpfc_overlap indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

nnpfc_constant_patch_size_flag equal to 1 indicates that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts as input any patch size with width inpPatchWidth and height inpPatchHeight such that the width of the extended patch (i.e., a patch plus the overlapping area), which is equal to inpPatchWidth + 2 * nnpfc_overlap, is a positive integer multiple of ~~the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input~~ nnpfc_extended_patch_width_cd_minus1 + 1, and the height of the extended patch,

which is equal to inpPatchHeight + 2 * nnpfc_overlap, is a positive integer multiple of nnpfc_extended_patch_height_cd_minus1 + 1.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, indicates the horizontal sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, indicates the vertical sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

nnpfc_extended_patch_width_cd_minus1 plus 1, when nnpfc_constant_patch_size_flag equal to 0, indicates a common divisor of all allowed values of the width of the extended patch required for the input to the post-processing filter. The value of nnpfc_extended_patch_width_cd_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

Alternatively, the value of nnpfc_extended_patch_width_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(32766, CroppedWidth − 1), inclusive. Alternatively, the value of nnpfc_extended_patch_width_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(32766, CroppedWidth + 2 * nnpfc_overlap − 1), inclusive. Alternatively, the value of nnpfc_extended_patch_width_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(2 * nnpfc_overlap + 32766, CroppedWidth + 2 * nnpfc_overlap − 1), inclusive.

nnpfc_extended_patch_height_lcd_minus1 plus 1, when nnpfc_constant_patch_size_flag equal to 0, indicates a common divisor of all allowed values of the height of the extended patch required for the input to the post-processing filter. The value of nnpfc_extended_patch_height_cd_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

Alternatively, the value of nnpfc_extended_patch_height_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(32766, CroppedHeight − 1), inclusive. Alternatively, the value of nnpfc_extended_patch_height_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(32766, CroppedHeight + 2 * nnpfc_overlap − 1), inclusive. Alternatively, the value of nnpfc_extended_patch_height_cd_minus1 shall be in the range of 2 * nnpfc_overlap to Min(2 * nnpfc_overlap + 32766, CroppedHeight + 2 * nnpfc_overlap − 1), inclusive.

Let the variables inpPatchWidth and inpPatchHeight be the patch size width and the patch size height, respectively.

If nnpfc_constant_patch_size_flag is equal to 0, the following applies:

– The values of inpPatchWidth and inpPatchHeight are either provided by external means not specified in this document or set by the post-processor itself.

– The value of inpPatchWidth + 2 * nnpfc_overlap shall be a positive integer multiple of ~~nnpfc_patch_width_minus1 + 1~~ nnpfc_extended_patch_width_cd_minus1 + 1 and inpPatchWidth2 shall be less than or equal to CroppedWidth. The value of inpPatchHeight + 2 * nnpfc_overlap shall be a positive integer multiple of ~~nnpfc_patch_height_minus1 + 1~~ nnpfc_extended_patch_height_cd_minus1 + 1 and inpPatchHeight shall be less than or equal to CroppedHeight.

Otherwise (nnpfc_constant_patch_size_flag is equal to 1), the value of inpPatchWidth is set equal to nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

~~**nnpfc_overlap** indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.~~

...

6.13. Embodiment 13

This embodiment is for the invention item 8 and all its subitems summarized above in Section 5.

Neural-network post-filter activation SEI message syntax

nn_post_filter_characteristics(payloadSize) {	**Descriptor**
…	ue(v)
nnpfc_overlap	ue(v)
nnpfc_constant_patch_size_flag	u(1)
if(nnpfc_constant_patch_size_flag) {	
nnpfc_patch_width_minus1	ue(v)
nnpfc_patch_height_minus1	ue(v)
} else {	
nnpfc_extended_patch_width_cd_delta_minus1	ue(v)
nnpfc_extended_patch_height_cd_delta_minus1	ue(v)

~~}~~	
~~**nnpfc_overlap**~~	~~ue(v)~~
…	

…

Neural-network post-filter activation SEI message semantics

…

nnpfc_overlap indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.

nnpfc_constant_patch_size_flag equal to 1 indicates that the post-processing filter accepts exactly the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input. nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts as input any patch size with width inpPatchWidth and height inpPatchHeight such that the width of the extended patch (i.e., a patch plus the overlapping area), which is equal to inpPatchWidth + 2 * nnpfc_overlap, is a positive integer multiple of ~~the patch size indicated by nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 as input~~ nnpfc_extended_patch_width_cd_delta_minus1 + 1+ 2 * nnpfc_overlap, and the height of the extended patch, which is equal to inpPatchHeight + 2 * nnpfc_overlap, is a positive integer multiple of nnpfc_extended_patch_height_cd_delta_minus1 +1 + 2 * nnpfc_overlap.

nnpfc_patch_width_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, indicates the horizontal sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_width_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_patch_height_minus1 + 1, when nnpfc_constant_patch_size_flag equal to 1, indicates the vertical sample counts of the patch size required for the input to the post-processing filter. The value of nnpfc_patch_height_minus1 shall be in the range of 0 to Min(32766, CroppedHeight − 1), inclusive.

nnpfc_extended_patch_width_cd_delta_minus1 plus 1 plus 2 * nnpfc_overlap, when nnpfc_constant_patch_size_flag equal to 0, indicates a common divisor of all allowed values of the width of the extended patch required for the input to the post-processing filter. The value of nnpfc_extended_patch_width_delta_cd_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

nnpfc_extended_patch_height_cd_delta_minus1 plus 1 plus 2 * nnpfc_overlap, when nnpfc_constant_patch_size_flag equal to 0, indicates a common divisor of all allowed values

of the height of the extended patch required for the input to the post-processing filter. The value of nnpfc_extended_patch_height_cd_delta_minus1 shall be in the range of 0 to Min(32766, CroppedWidth − 1), inclusive.

Let the variables inpPatchWidth and inpPatchHeight be the patch size width and the patch size height, respectively.

If nnpfc_constant_patch_size_flag is equal to 0, the following applies:

- The values of inpPatchWidth and inpPatchHeight are either provided by external means not specified in this document or set by the post-processor itself.
- The value of inpPatchWidth + 2 * nnpfc_overlap shall be a positive integer multiple of ~~nnpfc_patch_width_minus1 + 1~~ nnpfc_extended_patch_width_cd_delta_minus1 + 1 + 2 * nnpfc_overlap and inpPatchWidth2 shall be less than or equal to CroppedWidth. The value of inpPatchHeight + 2 * nnpfc_overlap shall be a positive integer multiple of ~~nnpfc_patch_height_minus1 + 1~~ nnpfc_extended_patch_height_cd_minus1 + 1 + 2 * nnpfc_overlap and inpPatchHeight shall be less than or equal to CroppedHeight.

Otherwise (nnpfc_constant_patch_size_flag is equal to 1), the value of inpPatchWidth is set equal to nnpfc_patch_width_minus1 + 1 and the value of inpPatchHeight is set equal to nnpfc_patch_height_minus1 + 1.

~~**nnpfc_overlap** indicates the overlapping horizontal and vertical sample counts of adjacent input tensors of the post-processing filter. The value of nnpfc_overlap shall be in the range of 0 to 16383, inclusive.~~

...

[0061] Fig. 5 illustrates a flowchart of a method 500 for video processing in accordance with embodiments of the present disclosure. At block 510, a conversion between a current video unit of a video and a bitstream of the video is performed. As used herein, the current video block may be a current picture or a current slice. A neural network filter is applied to the current video unit. A patch size of the neural network filter is associated with an overlapping size of samples of the neural network filter. A first indication of the patch size and a second indication of the overlapping size are included in the bitstream. In some embodiments, the conversion between the current video unit and the bitstream may include encoding the current video unit into the bitstream. Alternatively, or in addition, the conversion may include decoding the current video unit from the bitstream.

[0062] The method 500 enables applying the neural network filter with a patch size indicated in the bitstream. In this way, it can improve the coding efficiency and coding effectiveness of video coding.

[0063] In some embodiments, the neural network filter comprises a neural network post-processing filter.

[0064] In some embodiments, the first and second indications are included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream.

[0065] In some embodiments, the second indication is before the first indication. For example, in the NNPFC SEI message, the syntax element nnpfc_overlap is moved to be before the syntax element nnpfc_constant_patch_size_flag.

[0066] In some embodiments, the first indication is equal to a first value, and a third indication for a width of the patch size and a fourth indication for a height of the patch size is included in the bitstream. For example, the first value may be 1. For example, in the NNPFC SEI message, the syntax elements nnpfc_patch_width_minus1 and nnpfc_patch_height_minus1 are only signalled when nnpfc_constant_patch_size_flag is equal to 1.

[0067] In some embodiments, a fifth indication of a horizontal common divisor and a sixth indication of a vertical common divisor for a plurality of allowed value of width and height of an extended patch for an input of the neural network filter is included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream, the extended patch comprising a patch and an overlapping area. For example, in the NNPFC SEI message, two additional syntax elements, indicating a horizontal common divisor and a vertical common divisor of all allowed values of the width and the height, respectively, of the extended patch (i.e., a patch plus the overlapping area) required for the input to the post-processing filter, are signalled.

[0068] In some embodiments, the fifth indication comprises a value of the horizontal common divisor minus 1, and the sixth indication comprises a value of the vertical common divisor minus 1.

[0069] In some embodiments, the fifth indication comprises a value of the horizontal common divisor minus 1 minus 2 times the overlapping size, and the sixth indication

comprises a value of the vertical common divisor minus 1 minus 2 times the overlapping size.

[0070] In some embodiments, the fifth indication and the sixth indication are included in the NNPFC SEI message based on the first indication being equal to a second value.

[0071] In some embodiments, the second value is 0.

[0072] In some embodiments, the first indication being equal to the second value indicates that the neural network filter accepts as input with a patch size with a first width and a first height, a second width of an extended patch is equal to a sum of the first width and two times the overlapping size, and a second height of the extended patch is equal to a sum of the first height and two times the overlapping size, and wherein the second width is a positive integer multiple of the horizontal common divisor, and the second height is a positive integer multiple of the vertical common divisor. For example, it is specified that, nnpfc_constant_patch_size_flag equal to 0 indicates that the post-processing filter accepts as input any patch size with width inpPatchWidth and height inpPatchHeight such that the width of the extended patch (i.e., a patch plus the overlapping area), which is equal to inpPatchWidth + 2 * nnpfc_overlap, is a positive integer multiple of the horizontal common divisor, and the height of the extended patch, which is equal to inpPatchHeight + 2 * nnpfc_overlap, is a positive integer multiple of the vertical common divisor.

[0073] According to further embodiments of the present disclosure, a non-transitory computer-readable recording medium is provided. The non-transitory computer-readable recording medium stores a bitstream of a video which is generated by a method performed by an apparatus for video processing. In the method, the bitstream is generated. A neural network filter is applied to a current video unit of the video. A patch size of the neural network filter is associated with an overlapping size of samples of the neural network filter. A first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0074] According to still further embodiments of the present disclosure, a method for storing bitstream of a video is provided. In the method, the bitstream is generated. The bitstream is stored in a non-transitory computer-readable recording medium. A neural network filter is applied to a current video unit of the video. A patch size of the neural network filter is associated with an overlapping size of samples of the neural network

filter. A first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0075] Implementations of the present disclosure can be described in view of the following clauses, the features of which can be combined in any reasonable manner.

[0076] Clause 1. A method for video processing, comprising: performing a conversion between a current video unit of a video and a bitstream of the video, wherein a neural network filter is applied to the current video unit, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0077] Clause 2. The method of clause 1, wherein the second indication is before the first indication.

[0078] Clause 3. The method of clause 1 or 2, wherein the first indication is equal to a first value, and a third indication for a width of the patch size and a fourth indication for a height of the patch size is included in the bitstream.

[0079] Clause 4. The method of clause 3, wherein the first value is 1.

[0080] Clause 5. The method of any of clauses 1-4, wherein the neural network filter comprises a neural network post-processing filter.

[0081] Clause 6. The method of any of clause 1-5, wherein the first and second indications are included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream.

[0082] Clause 7. The method of any of clauses 1-6, wherein a fifth indication of a horizontal common divisor and a sixth indication of a vertical common divisor for a plurality of allowed value of width and height of an extended patch for an input of the neural network filter is included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream, the extended patch comprising a patch and an overlapping area.

[0083] Clause 8. The method of clause 7, wherein the fifth indication comprises a value of the horizontal common divisor minus 1, and the sixth indication comprises a value of the vertical common divisor minus 1.

[0084] Clause 9. The method of clause 7, wherein the fifth indication comprises a value of the horizontal common divisor minus 1 minus 2 times the overlapping size, and the sixth indication comprises a value of the vertical common divisor minus 1 minus 2 times the overlapping size.

[0085] Clause 10. The method of any of clauses 7-9, wherein the fifth indication and the sixth indication are included in the NNPFC SEI message based on the first indication being equal to a second value.

[0086] Clause 11. The method of clause 10, wherein the second value is 0.

[0087] Clause 12. The method of clause 10 or 11, wherein the first indication being equal to the second value indicates that the neural network filter accepts as input with a patch size with a first width and a first height, a second width of an extended patch is equal to a sum of the first width and two times the overlapping size, and a second height of the extended patch is equal to a sum of the first height and two times the overlapping size, and wherein the second width is a positive integer multiple of the horizontal common divisor, and the second height is a positive integer multiple of the vertical common divisor.

[0088] Clause 13. The method of any of clauses 1-12, wherein the conversion includes encoding the current video unit into the bitstream.

[0089] Clause 14. The method of any of clauses 1-12, wherein the conversion includes decoding the current video unit from the bitstream.

[0090] Clause 15. An apparatus for video processing comprising a processor and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor, cause the processor to perform a method in accordance with any of clauses 1-14.

[0091] Clause 16. A non-transitory computer-readable storage medium storing instructions that cause a processor to perform a method in accordance with any of clauses 1-14.

[0092] Clause 17. A non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by an apparatus for video processing, wherein the method comprises: generating the bitstream, wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network

filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

[0093] Clause 18. A method for storing a bitstream of a video, comprising: generating the bitstream; and storing the bitstream in a non-transitory computer-readable recording medium, wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

Example Device

[0094] Fig. 6 illustrates a block diagram of a computing device 600 in which various embodiments of the present disclosure can be implemented. The computing device 600 may be implemented as or included in the source device 110 (or the video encoder 114 or 200) or the destination device 120 (or the video decoder 124 or 300).

[0095] It would be appreciated that the computing device 600 shown in Fig. 6 is merely for purpose of illustration, without suggesting any limitation to the functions and scopes of the embodiments of the present disclosure in any manner.

[0096] As shown in Fig. 6, the computing device 600 includes a general-purpose computing device 600. The computing device 600 may at least comprise one or more processors or processing units 610, a memory 620, a storage unit 630, one or more communication units 640, one or more input devices 650, and one or more output devices 660.

[0097] In some embodiments, the computing device 600 may be implemented as any user terminal or server terminal having the computing capability. The server terminal may be a server, a large-scale computing device or the like that is provided by a service provider. The user terminal may for example be any type of mobile terminal, fixed terminal, or portable terminal, including a mobile phone, station, unit, device, multimedia computer, multimedia tablet, Internet node, communicator, desktop computer, laptop computer, notebook computer, netbook computer, tablet computer, personal communication system (PCS) device, personal navigation device, personal digital assistant (PDA), audio/video player, digital camera/video camera, positioning device, television receiver, radio broadcast receiver, E-book device, gaming device, or any

combination thereof, including the accessories and peripherals of these devices, or any combination thereof. It would be contemplated that the computing device 600 can support any type of interface to a user (such as "wearable" circuitry and the like).

[0098] The processing unit 610 may be a physical or virtual processor and can implement various processes based on programs stored in the memory 620. In a multi-processor system, multiple processing units execute computer executable instructions in parallel so as to improve the parallel processing capability of the computing device 600. The processing unit 610 may also be referred to as a central processing unit (CPU), a microprocessor, a controller or a microcontroller.

[0099] The computing device 600 typically includes various computer storage medium. Such medium can be any medium accessible by the computing device 600, including, but not limited to, volatile and non-volatile medium, or detachable and non-detachable medium. The memory 620 can be a volatile memory (for example, a register, cache, Random Access Memory (RAM)), a non-volatile memory (such as a Read-Only Memory (ROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), or a flash memory), or any combination thereof. The storage unit 630 may be any detachable or non-detachable medium and may include a machine-readable medium such as a memory, flash memory drive, magnetic disk or another other media, which can be used for storing information and/or data and can be accessed in the computing device 600.

[0100] The computing device 600 may further include additional detachable/non-detachable, volatile/non-volatile memory medium. Although not shown in Fig. 6, it is possible to provide a magnetic disk drive for reading from and/or writing into a detachable and non-volatile magnetic disk and an optical disk drive for reading from and/or writing into a detachable non-volatile optical disk. In such cases, each drive may be connected to a bus (not shown) via one or more data medium interfaces.

[0101] The communication unit 640 communicates with a further computing device via the communication medium. In addition, the functions of the components in the computing device 600 can be implemented by a single computing cluster or multiple computing machines that can communicate via communication connections. Therefore, the computing device 600 can operate in a networked environment using a logical connection with one or more other servers, networked personal computers (PCs) or further general network nodes.

[0102] The input device 650 may be one or more of a variety of input devices, such as a mouse, keyboard, tracking ball, voice-input device, and the like. The output device 660 may be one or more of a variety of output devices, such as a display, loudspeaker, printer, and the like. By means of the communication unit 640, the computing device 600 can further communicate with one or more external devices (not shown) such as the storage devices and display device, with one or more devices enabling the user to interact with the computing device 600, or any devices (such as a network card, a modem and the like) enabling the computing device 600 to communicate with one or more other computing devices, if required. Such communication can be performed via input/output (I/O) interfaces (not shown).

[0103] In some embodiments, instead of being integrated in a single device, some or all components of the computing device 600 may also be arranged in cloud computing architecture. In the cloud computing architecture, the components may be provided remotely and work together to implement the functionalities described in the present disclosure. In some embodiments, cloud computing provides computing, software, data access and storage service, which will not require end users to be aware of the physical locations or configurations of the systems or hardware providing these services. In various embodiments, the cloud computing provides the services via a wide area network (such as Internet) using suitable protocols. For example, a cloud computing provider provides applications over the wide area network, which can be accessed through a web browser or any other computing components. The software or components of the cloud computing architecture and corresponding data may be stored on a server at a remote position. The computing resources in the cloud computing environment may be merged or distributed at locations in a remote data center. Cloud computing infrastructures may provide the services through a shared data center, though they behave as a single access point for the users. Therefore, the cloud computing architectures may be used to provide the components and functionalities described herein from a service provider at a remote location. Alternatively, they may be provided from a conventional server or installed directly or otherwise on a client device.

[0104] The computing device 600 may be used to implement video encoding/decoding in embodiments of the present disclosure. The memory 620 may include one or more video coding modules 625 having one or more program instructions. These modules are accessible and executable by the processing unit 610 to perform the functionalities of the

various embodiments described herein.

[0105] In the example embodiments of performing video encoding, the input device 650 may receive video data as an input 670 to be encoded. The video data may be processed, for example, by the video coding module 625, to generate an encoded bitstream. The encoded bitstream may be provided via the output device 660 as an output 680.

[0106] In the example embodiments of performing video decoding, the input device 650 may receive an encoded bitstream as the input 670. The encoded bitstream may be processed, for example, by the video coding module 625, to generate decoded video data. The decoded video data may be provided via the output device 660 as the output 680.

[0107] While this disclosure has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the present application as defined by the appended claims. Such variations are intended to be covered by the scope of this present application. As such, the foregoing description of embodiments of the present application is not intended to be limiting.

I/We Claim:

1. A method for video processing, comprising:

performing a conversion between a current video unit of a video and a bitstream of the video,

wherein a neural network filter is applied to the current video unit, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and

wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

2. The method of claim 1, wherein the second indication is before the first indication.

3. The method of claim 1 or 2, wherein the first indication is equal to a first value, and a third indication for a width of the patch size and a fourth indication for a height of the patch size is included in the bitstream.

4. The method of claim 3, wherein the first value is 1.

5. The method of any of claims 1-4, wherein the neural network filter comprises a neural network post-processing filter.

6. The method of any of claim 1-5, wherein the first and second indications are included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream.

7. The method of any of claims 1-6, wherein a fifth indication of a horizontal common divisor and a sixth indication of a vertical common divisor for a plurality of allowed value of width and height of an extended patch for an input of the neural network filter is included in a neural network post-filter characteristics (NNPFC) supplemental enhancement information (SEI) message in the bitstream, the extended patch comprising a patch and an overlapping area.

8. The method of claim 7, wherein the fifth indication comprises a value of the horizontal common divisor minus 1, and the sixth indication comprises a value of the vertical common divisor minus 1.

9. The method of claim 7, wherein the fifth indication comprises a value of the horizontal common divisor minus 1 minus 2 times the overlapping size, and the sixth indication comprises a value of the vertical common divisor minus 1 minus 2 times the overlapping size.

10. The method of any of claims 7-9, wherein the fifth indication and the sixth indication are included in the NNPFC SEI message based on the first indication being equal to a second value.

11. The method of claim 10, wherein the second value is 0.

12. The method of claim 10 or 11, wherein the first indication being equal to the second value indicates that the neural network filter accepts as input with a patch size with a first width and a first height, a second width of an extended patch is equal to a sum of the first width and two times the overlapping size, and a second height of the extended patch is equal to a sum of the first height and two times the overlapping size, and

wherein the second width is a positive integer multiple of the horizontal common divisor, and the second height is a positive integer multiple of the vertical common divisor.

13. The method of any of claims 1-12, wherein the conversion includes encoding the current video unit into the bitstream.

14. The method of any of claims 1-12, wherein the conversion includes decoding the current video unit from the bitstream.

15. An apparatus for video processing comprising a processor and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor, cause the processor to perform a method in accordance with any of claims 1-14.

16. A non-transitory computer-readable storage medium storing instructions that cause a processor to perform a method in accordance with any of claims 1-14.

17. A non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by an apparatus for video processing, wherein the method comprises:

generating the bitstream,

wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and

wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

18. A method for storing a bitstream of a video, comprising:

generating the bitstream; and

storing the bitstream in a non-transitory computer-readable recording medium,

wherein a neural network filter is applied to a current video unit of the video, a patch size of the neural network filter being associated with an overlapping size of samples of the neural network filter, and

wherein a first indication of the patch size and a second indication of the overlapping size are included in the bitstream.

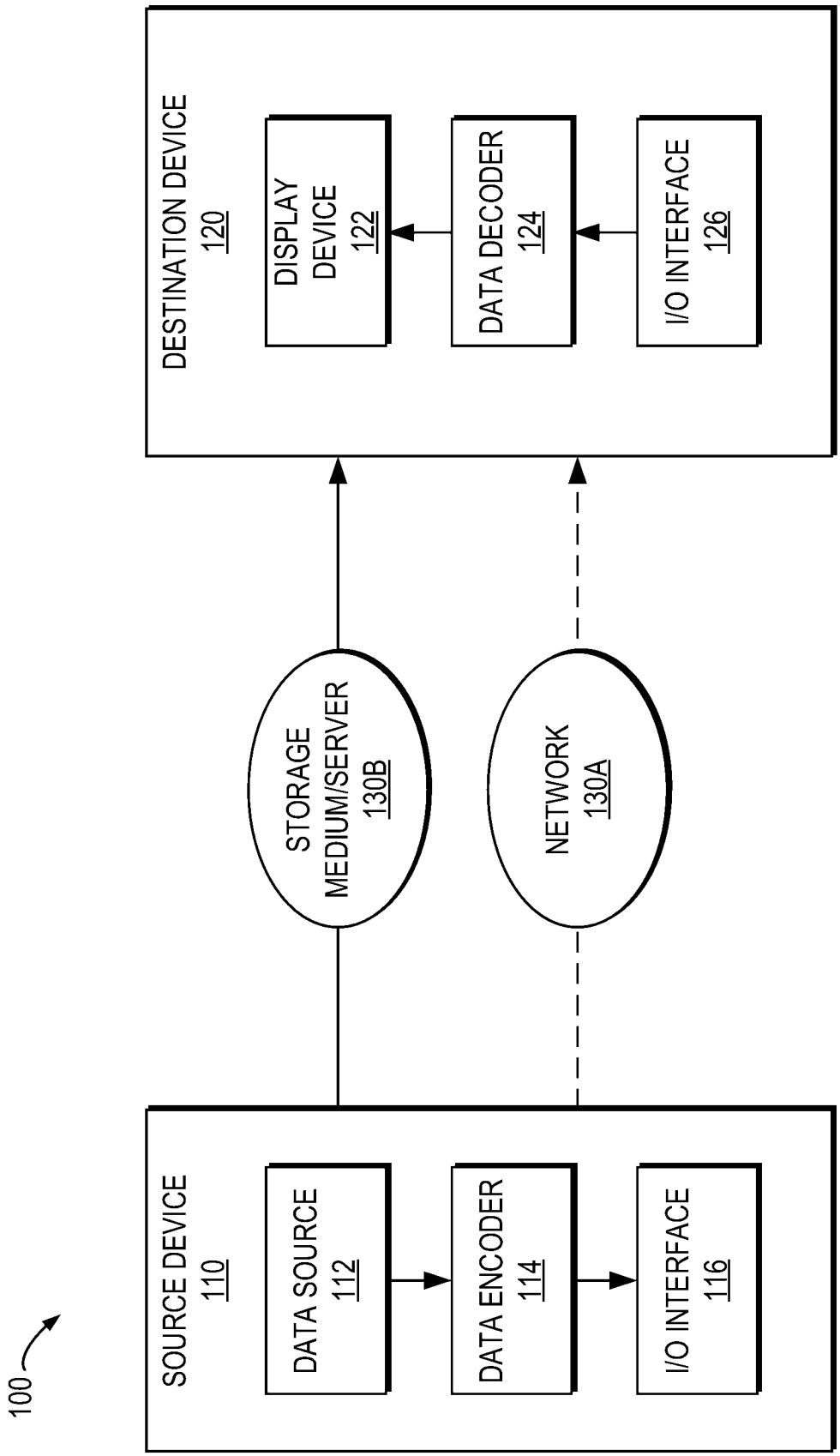
100
SOURCE DEVICE
110
DATA SOURCE
112
DATA ENCODER
114
I/O INTERFACE
116
STORAGE
MEDIUM/SERVER
130B
NETWORK
130A
DESTINATION DEVICE
120
DISPLAY
DEVICE
122
DATA DECODER
124
I/O INTERFACE
126

Fig. 1

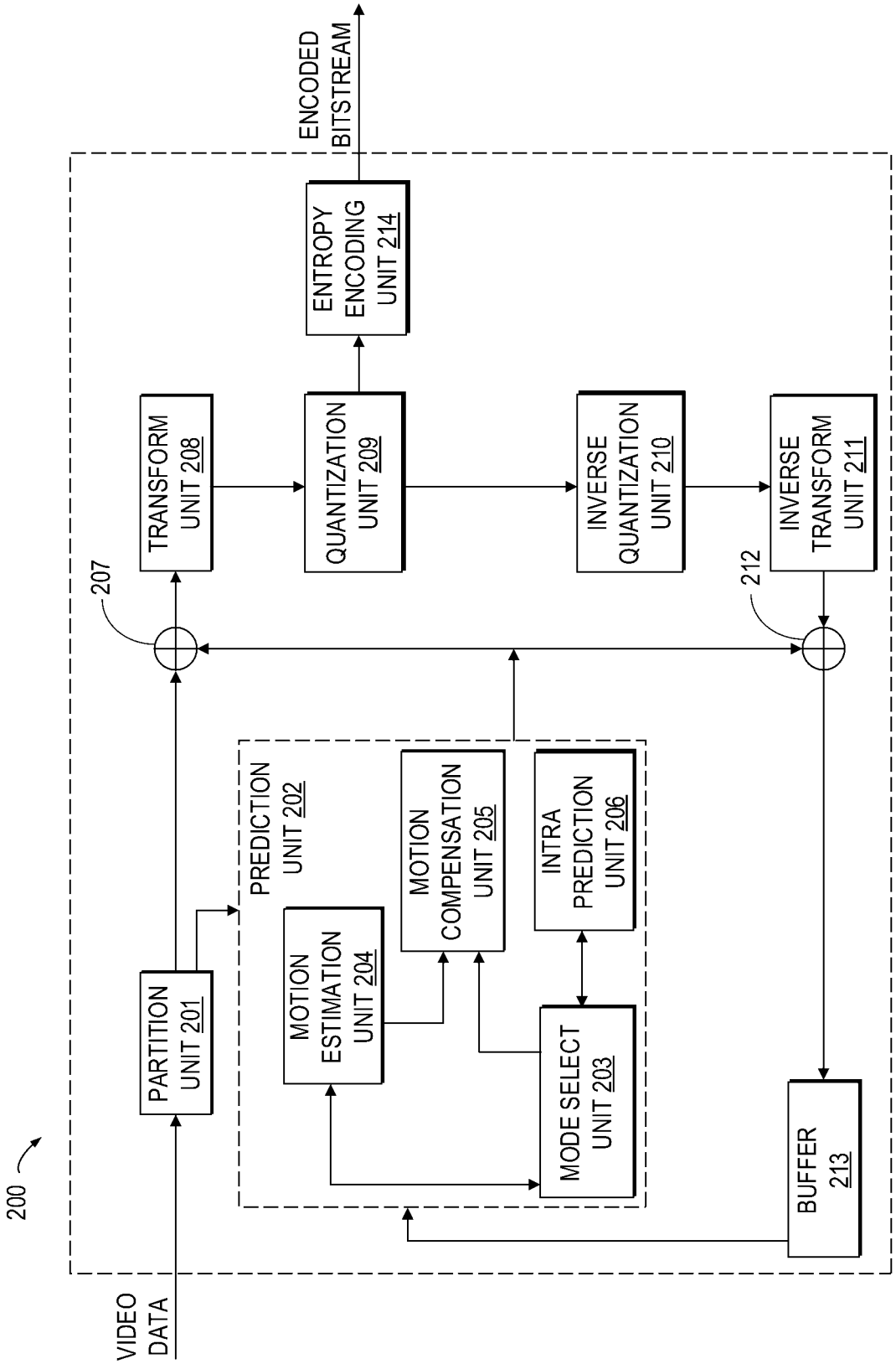
200
VIDEO DATA
PARTITION UNIT 201
PREDICTION UNIT 202
MOTION ESTIMATION UNIT 204
MOTION COMPENSATION UNIT 205
INTRA PREDICTION UNIT 206
MODE SELECT UNIT 203
207
TRANSFORM UNIT 208
QUANTIZATION UNIT 209
ENTROPY ENCODING UNIT 214
ENCODED BITSTREAM
INVERSE QUANTIZATION UNIT 210
INVERSE TRANSFORM UNIT 211
212
BUFFER 213

Fig. 2

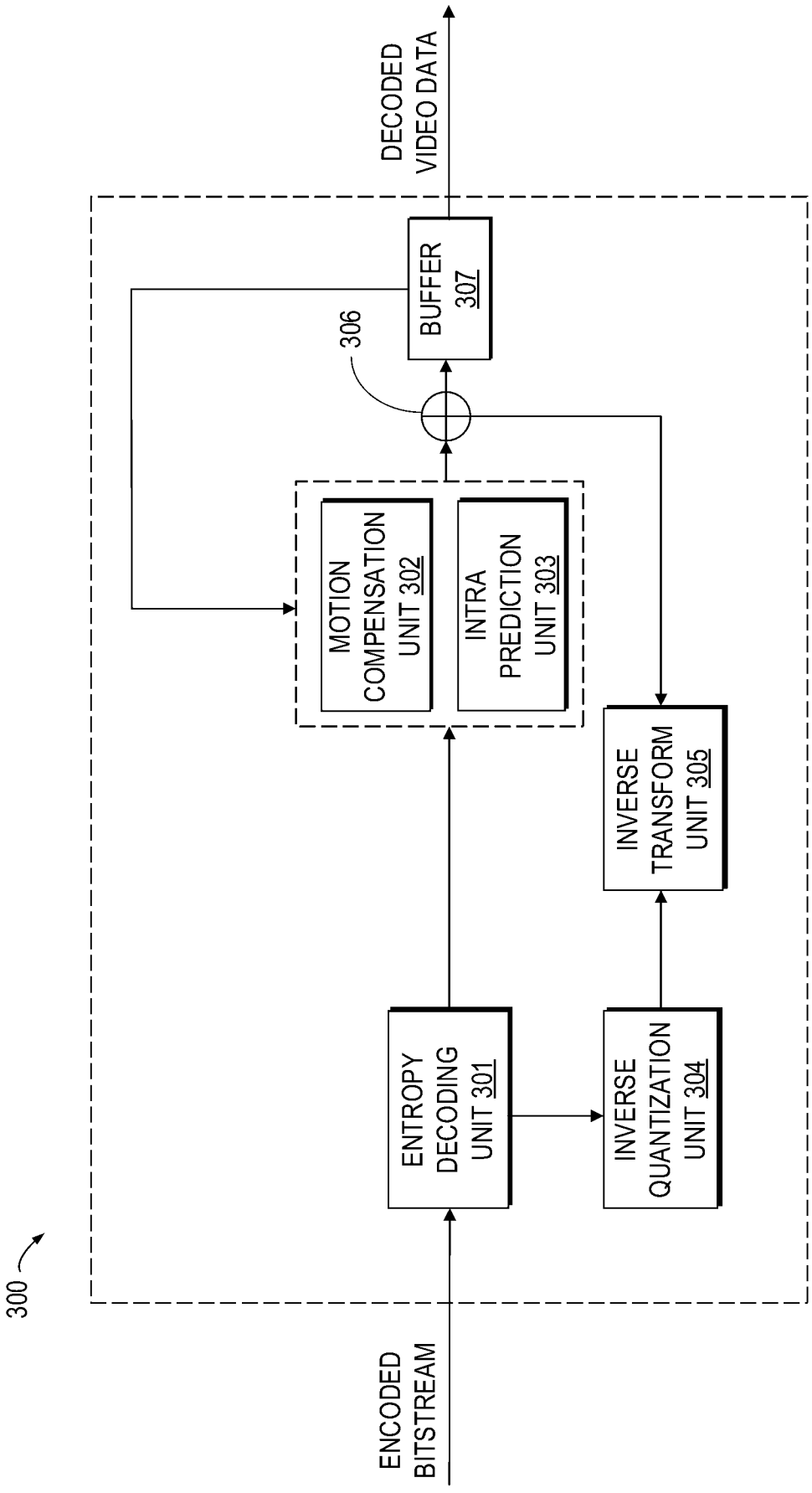
300
ENCODED BITSTREAM
ENTROPY DECODING UNIT 301
INVERSE QUANTIZATION UNIT 304
INVERSE TRANSFORM UNIT 305
MOTION COMPENSATION UNIT 302
INTRA PREDICTION UNIT 303
306
BUFFER 307
DECODED VIDEO DATA

Fig. 3

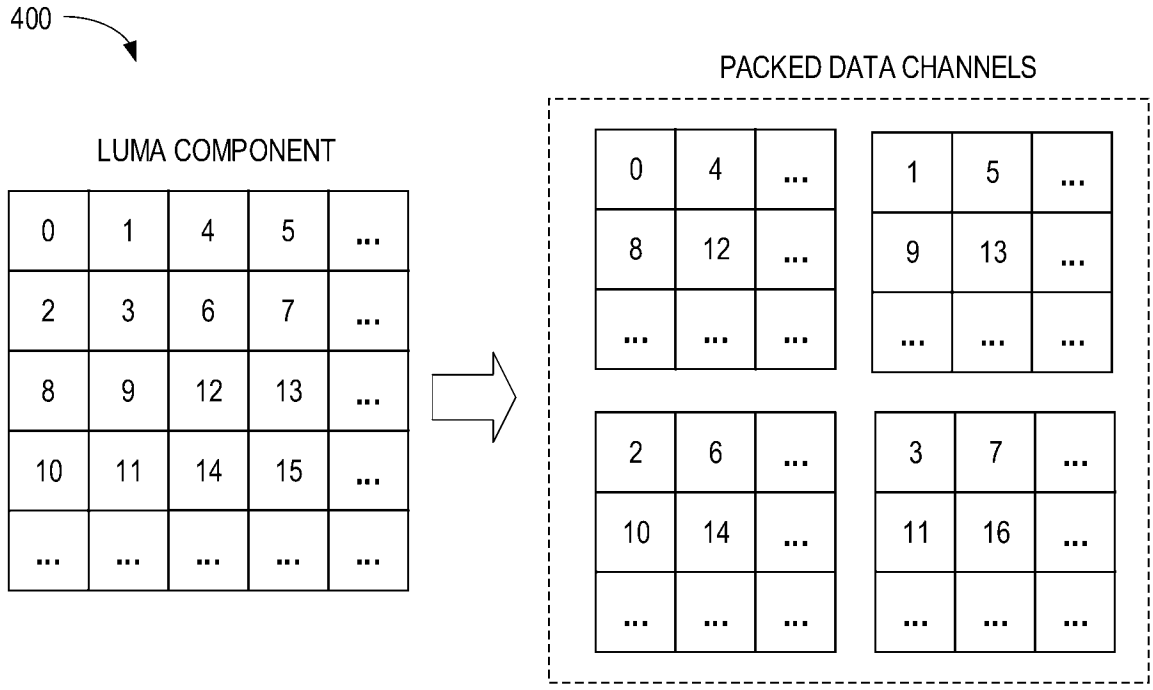
400
LUMA COMPONENT
0 1 4 5 ...
2 3 6 7 ...
8 9 12 13 ...
10 11 14 15 ...
... ...
PACKED DATA CHANNELS
0 4 ... 8 12 ... ...
1 5 ... 9 13 ... ...
2 6 ... 10 14 ... ...
3 7 ... 11 16 ... ...

Fig. 4

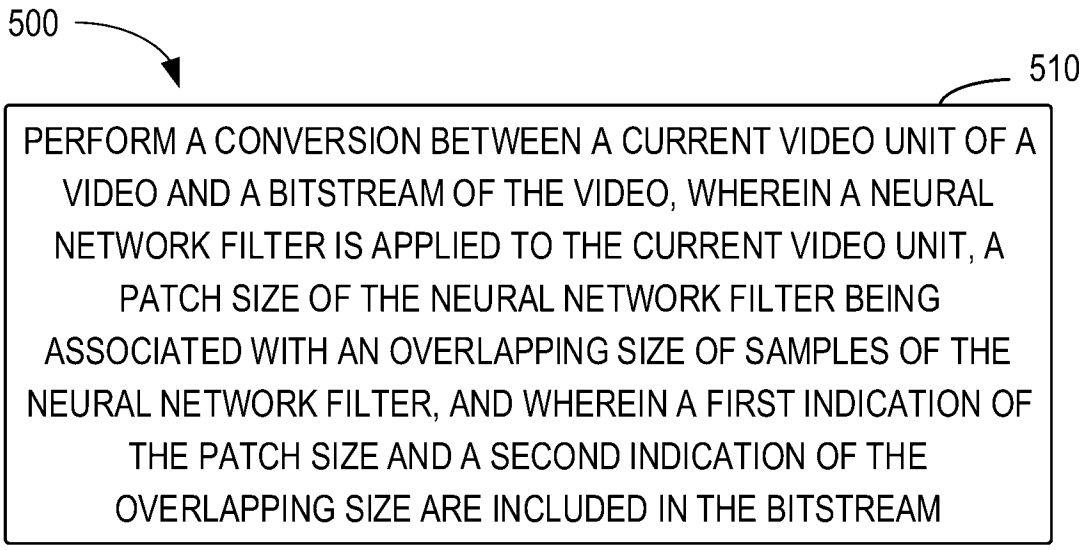
500
510
PERFORM A CONVERSION BETWEEN A CURRENT VIDEO UNIT OF A VIDEO AND A BITSTREAM OF THE VIDEO, WHEREIN A NEURAL NETWORK FILTER IS APPLIED TO THE CURRENT VIDEO UNIT, A PATCH SIZE OF THE NEURAL NETWORK FILTER BEING ASSOCIATED WITH AN OVERLAPPING SIZE OF SAMPLES OF THE NEURAL NETWORK FILTER, AND WHEREIN A FIRST INDICATION OF THE PATCH SIZE AND A SECOND INDICATION OF THE OVERLAPPING SIZE ARE INCLUDED IN THE BITSTREAM

Fig. 5

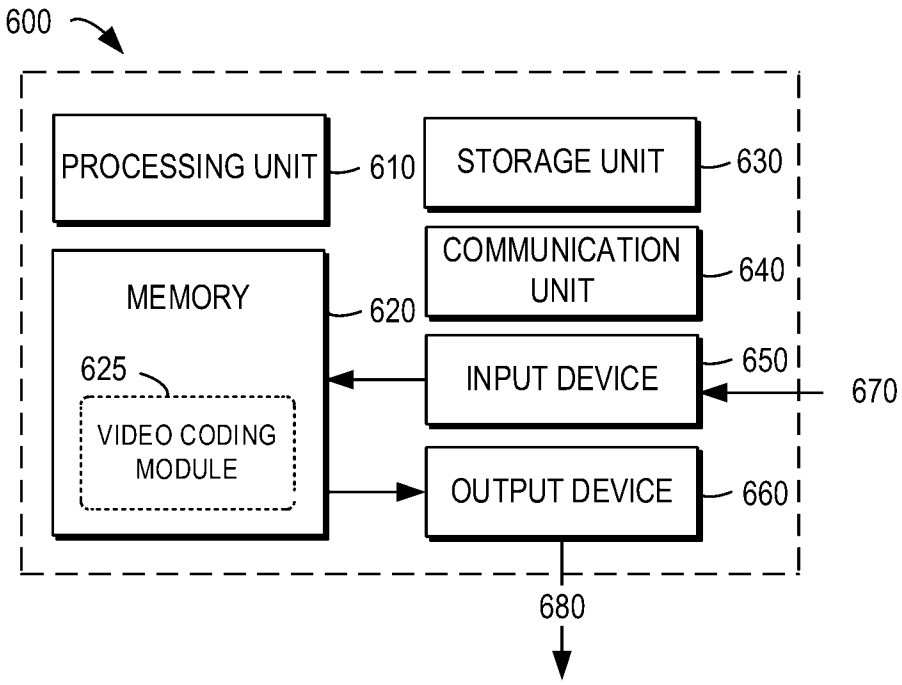
600
PROCESSING UNIT
610
STORAGE UNIT
630
COMMUNICATION UNIT
640
MEMORY
620
625
VIDEO CODING MODULE
INPUT DEVICE
650
670
OUTPUT DEVICE
660
680

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2024/072173

A. CLASSIFICATION OF SUBJECT MATTER

H04N19/184(2014.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC:H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

CNTXT;ENTXT;DWPI;CNKI;IEEE;JVET:video,image,picture,convers+,transf+,bitstream,neural,network,filter,patch,size, overlap+, sampl+, indicat+,SEI,NNPFC, character+, horizontal ,vertical

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2022050558 A1 (SAMSUNG ELECTRONICS CO., LTD.) 10 March 2022 (2022-03-10) description pages 4-10,14-15	1-18
A	WO 2022055165 A1 (LG ELECTRONICS INC.) 17 March 2022 (2022-03-17) the whole document	1-18
A	CN 110866876 A (NORTHWESTERN POLYTECHNICAL UNIVERSITY) 06 March 2020 (2020-03-06) the whole document	1-18
A	US 11216932 B1 (MINDS AI TECHNOLOGIES LTD) 04 January 2022 (2022-01-04) the whole document	1-18
A	NORKIN, Andrey. "AHG17: Alternative film grain characteristics SEI message" *Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11 18th Meeting: by teleconference, Document: JVET-R0384-v3,* 24 April 2020 (2020-04-24), the whole document	1-18

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 March 2024	**19 March 2024**
Name and mailing address of the ISA/CN **CHINA NATIONAL INTELLECTUAL PROPERTY ADMINISTRATION** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China**	Authorized officer **WU,Qian** Telephone No. **(+86) 010-53961822**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2024/072173

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
WO	2022050558	A1	10 March 2022	KR	20220029212	A	08 March 2022
				US	2023205923	A1	29 June 2023
WO	2022055165	A1	17 March 2022	US	2023419557	A1	28 December 2023
CN	110866876	A	06 March 2020		None		
US	11216932	B1	04 January 2022	WO	2022200854	A1	29 September 2022
				TW	202239281	A	01 October 2022
				US	2022309637	A1	29 September 2022
				US	11694318	B2	04 July 2023