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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 October 2024 (24.10.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/217423 A1

(51) International Patent Classification:
H04N 19/70 (2014.01) *H04N 19/587* (2014.01)

(21) International Application Number:
PCT/CN2024/088134

(22) International Filing Date:
16 April 2024 (16.04.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PCT/CN2023/088800
17 April 2023 (17.04.2023) CN

(71) Applicant: **DOUYIN VISION CO., LTD.** [CN/CN]; Room B-0035, 2/F, No. 3 Building, No. 30, Shixing Road, Shijingshan District, Beijing 100041 (CN).

(72) Inventors: **WANG, Meng**; Jinritoutiao Post office, China Satellite Communications Tower, No. 63, Zhichun Road, Haidian District, Beijing 100080 (CN). **WU, Yaojun**; Jinritoutiao Post office, China Satellite Communications Tower, No. 63, Zhichun Road, Haidian District, Beijing 100080 (CN).

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

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

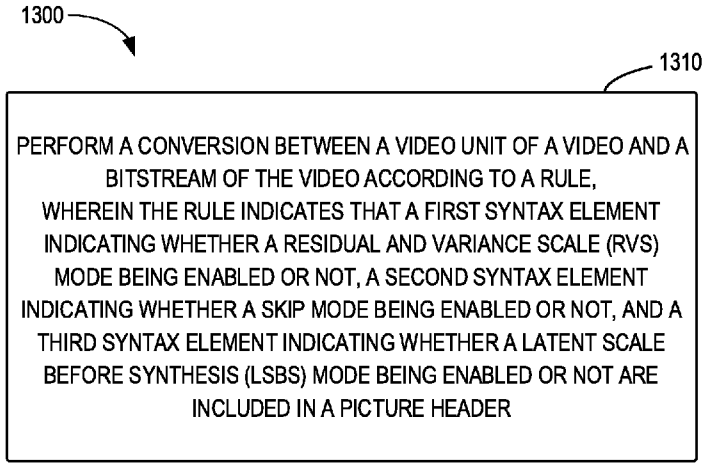


FIG. 13

(57) Abstract: Embodiments of the present disclosure provide a solution for video processing. A method for video processing is proposed. The method comprises: performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

WO 2024/217423 A1

[Continued on next page]

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-based image and video compression method with syntax elements design for mask and scale tools.

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 video unit of a video and a bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header. In this way, it can simplify the synthesis transform module while maintaining the reconstruction capability.

[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 of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

[0008] In a fifth aspect, a method for storing a bitstream of a video is proposed. The method comprises: generating the bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header; and storing the bitstream in a non-transitory computer-readable medium.

[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 is a schematic diagram illustrating an example transform coding scheme;

[0015] FIG. 5 illustrates example latent representations of an image;

[0016] FIG. 6 is a schematic diagram illustrating an example autoencoder implementing a hyperprior model;

[0017] FIG. 7 is a schematic diagram illustrating an example combined model configured to jointly optimize a context model along with a hyperprior and the autoencoder;

[0018] FIG. 8 illustrates an example encoding process;

[0019] FIG. 9 illustrates an example decoding process;

[0020] FIG. 10 illustrates an example encoder and decoder with wavelet-based transform;

[0021] FIG. 11 illustrates an example output of a forward wavelet-based transform;

[0022] FIG. 12 illustrates an example partitioning of the output of a forward wavelet-based transform;

[0023] FIG. 13 illustrates a flowchart of a method for video processing in accordance with embodiments of the present disclosure; and

[0024] FIG. 14 illustrates a block diagram of a computing device in which various embodiments of the present disclosure can be implemented.

[0025] Throughout the drawings, the same or similar reference numerals usually refer to the same or similar elements.

DETAILED DESCRIPTION

[0026] 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.

[0027] 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.

[0028] 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.

[0029] 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.

[0030] 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

[0031] 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.

[0032] 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.

[0033] 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.

[0034] 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.

[0035] 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.

[0036] 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.

[0037] 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.

[0038] In some embodiments, the video encoder 200 may include a partition unit 201, a predication 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.

[0039] In other examples, the video encoder 200 may include more, fewer, or different functional components. In an example, the predication unit 202 may include an intra block copy (IBC) unit. The IBC unit may perform predication in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[0040] 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.

[0041] 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.

[0042] 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 predication (CIIP) mode in which the predication is based on an inter predication signal and an intra predication 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-predication.

[0043] 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.

[0044] 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.

[0045] 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.

[0046] 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.

[0047] 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.

[0048] 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.

[0049] 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.

[0050] 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 predication (AMVP) and merge mode signaling.

[0051] 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.

[0052] 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.

[0053] 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.

[0054] 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.

[0055] 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.

[0056] 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 predication unit 202 to produce a reconstructed video block associated with the current video block for storage in the buffer 213.

[0057] After the reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed to reduce video blocking artifacts in the video block.

[0058] 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.

[0059] 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.

[0060] 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.

[0061] 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.

[0062] 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.

[0063] 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.

[0064] 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.

[0065] 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.

[0066] 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.

[0067] 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 predication and also produces decoded video for presentation on a display device.

[0068] 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

[0069] This present disclosure is related to a neural network-based image and video compression approach where an autoregressive neural network is utilized. The examples target a high efficiency synthesis transform for the decoder, therefore enhancing the quality of reconstruction images with moderate computational complexity. The present disclosure is applicable to both luma and chroma components.

2. Introduction

[0070] Deep learning is developing in a variety of areas, such as in computer vision and image processing. Inspired by the successful application of deep learning technology to computer vision areas, neural image/video compression technologies are being studied for application to image/video compression techniques. The neural network is designed based on interdisciplinary research of neuroscience and mathematics. The neural network has shown strong capabilities in the context of non-linear transform and classification. An example neural network-based image compression algorithm achieves comparable R-D performance with Versatile Video Coding (VVC), which is a video coding standard developed by the Joint Video Experts Team (JVET) with experts from motion picture experts group (MPEG) and Video coding experts group (VCEG). Neural network-based video compression is an actively developing research area resulting in continuous improvement of the performance of neural image compression. However, neural network-based video coding is still a largely undeveloped discipline due to the inherent difficulty of the problems addressed by neural networks.

2.1 Image/Video Compression

[0071] Image/video compression usually refers to a computing technology that compresses video images into binary code to facilitate storage and transmission. The binary codes may or may not support losslessly reconstructing the original image/video. Coding without data loss is known as lossless compression and coding while allowing for targeted loss of data in known as lossy compression, respectively. Most coding systems employ lossy compression since lossless reconstruction is not necessary in most scenarios. Usually the performance of image/video compression algorithms is evaluated based on a resulting compression ratio and reconstruction quality. Compression ratio is directly related to the number of binary codes resulting from compression, with fewer binary codes resulting in better compression. Reconstruction quality is measured by comparing the

reconstructed image/video with the original image/video, with greater similarity resulting in better reconstruction quality.

[0072] Image/video compression techniques can be divided into video coding methods and neural-network-based video compression methods. Video coding schemes adopt transform-based solutions, in which statistical dependency in latent variables, such as discrete cosine transform (DCT) and wavelet coefficients, is employed to carefully hand-engineer entropy codes to model the dependencies in the quantized regime. Neural network-based video compression can be grouped into neural network-based coding tools and end-to-end neural network-based video compression. The former is embedded into existing video codecs as coding tools and only serves as part of the framework, while the latter is a separate framework developed based on neural networks without depending on video codecs.

[0073] A series of video coding standards have been developed to accommodate the increasing demands of visual content transmission. The international organization for standardization (ISO)/ International Electrotechnical Commission (IEC) has two expert groups, namely Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MPEG). International Telecommunication Union (ITU) telecommunication standardization sector (ITU-T) also has a Video Coding Experts Group (VCEG), which is for standardization of image/video coding technology. The influential video coding standards published by these organizations include Joint Photographic Experts Group (JPEG), JPEG 2000, H.262, H.264/ advanced video coding (AVC) and H.265/High Efficiency Video Coding (HEVC). The Joint Video Experts Team (JVET), formed by MPEG and VCEG, developed the Versatile Video Coding (VVC) standard. An average of 50% bitrate reduction is reported by VVC under the same visual quality compared with HEVC.

[0074] Neural network-based image/video compression/coding is also under development. Example neural network coding network architectures are relatively shallow, and the performance of such networks is not satisfactory. Neural network-based methods benefit from the abundance of data and the support of powerful computing resources, and are therefore better exploited in a variety of applications. Neural network-based image/video compression has shown promising improvements and is confirmed to be feasible. Nevertheless, this technology is far from mature and a lot of challenges should be addressed.

2.2 Neural Networks

[0075] Neural networks, also known as artificial neural networks (ANN), are computational models used in machine learning technology. Neural networks are usually composed of multiple processing layers, and each layer is composed of multiple simple but non-linear basic computational units. One benefit of such deep networks is a capacity for processing data with multiple levels of abstraction and converting data into different kinds of representations. Representations created by neural networks are not manually designed. Instead, the deep network including the processing layers is learned from massive data using a general machine learning procedure. Deep learning eliminates the necessity of handcrafted representations. Thus, deep learning is regarded useful especially for processing natively unstructured data, such as acoustic and visual signals. The processing of such data has been a longstanding difficulty in the artificial intelligence field.

2.3 Neural Networks For Image Compression

[0076] Neural networks for image compression can be classified in two categories, including pixel probability models and auto-encoder models. Pixel probability models employ a predictive coding strategy. Auto-encoder models employ a transform-based solution. Sometimes, these two methods are combined together.

2.3.1 Pixel Probability Modeling

[0077] According to Shannon's information theory, the optimal method for lossless coding can reach the minimal coding rate, which is denoted as $-\log_2 p(x)$ where $p(x)$ is the probability of symbol x. Arithmetic coding is a lossless coding method that is believed to be among the optimal methods. Given a probability distribution $p(x)$, arithmetic coding causes the coding rate to be as close as possible to a theoretical limit $-\log_2 p(x)$ without considering the rounding error. Therefore, the remaining problem is to determine the probability, which is very challenging for natural image/video due to the curse of dimensionality. The curse of dimensionality refers to the problem that increasing dimensions causes data sets to become sparse, and hence rapidly increasing amounts of data is needed to effectively analyze and organize data as the number of dimensions increases.

[0078] Following the predictive coding strategy, one way to model $p(x)$ is to predict

pixel probabilities one by one in a raster scan order based on previous observations, where x is an image, can be expressed as follows:

$$p(x) = p(x_1)p(x_2|x_1) \dots p(x_i|x_1, \dots, x_{i-1}) \dots p(x_{m\times n}|x_1, \dots, x_{m\times n-1}) \quad (1)$$

where m and n are the height and width of the image, respectively. The previous observation is also known as the context of the current pixel. When the image is large, estimation of the conditional probability can be difficult. Thereby, a simplified method is to limit the range of the context of the current pixel as follows:

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1) \dots p(x_i|x_{i-k}, \dots, x_{i-1}) \dots p(x_{m\times n}|x_{m\times n-k}, \dots, x_{m\times n-1}) \quad (2)$$

where k is a pre-defined constant controlling the range of the context.

[0079] It should be noted that the condition may also take the sample values of other color components into consideration. For example, when coding the red (R), green (G), and blue (B) (RGB) color component, the R sample is dependent on previously coded pixels (including R,G, and/or B samples), the current G sample may be coded according to previously coded pixels and the current R sample. Further, when coding the current B sample, the previously coded pixels and the current R and G samples may also be taken into consideration.

[0080] Neural networks may be designed for computer vision tasks, and may also be effective in regression and classification problems. Therefore, neural networks may be used to estimate the probability of $p(x_i)$ given a context $x_1, x_2, \dots, x_{i-1}$. In an example neural network design, the pixel probability is employed for binary images according to $x_i \in \{-1, +1\}$. The neural autoregressive distribution estimator (NADE) is designed for pixel probability modeling. NADE is a feed-forward network with a single hidden layer. In another example, the feed-forward network may include connections skipping the hidden layer. Further, the parameters may also be shared. Example designs perform experiments on the binarized MNIST dataset. In an example, NADE is extended to a real-valued NADE (RNADE) model, where the probability $p(x_i|x_1, \dots, x_{i-1})$ is derived with a mixture of Gaussians. The RNADE model feed-forward network also has a single hidden layer, but the hidden layer employs rescaling to avoid saturation and uses a rectified linear unit (ReLU) instead of sigmoid. In another example, NADE and RNADE are improved by using reorganizing the order of the pixels and with deeper neural networks.

[0081] Designing advanced neural networks plays an important role in improving pixel

probability modeling. In an example neural network, a multi-dimensional long short-term memory (LSTM) is used. The LSTM works together with mixtures of conditional Gaussian scale mixtures for probability modeling. LSTM is a special kind of recurrent neural networks (RNNs) and may be employed to model sequential data. The spatial variant of LSTM may also be used for images later. Several different neural networks may be employed, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs), such as Pixel RNN (PixelRNN) and Pixel CNN (PixelCNN), respectively. In PixelRNN, two variants of LSTM, denoted as row LSTM and diagonal bidirectional LSTM (BiLSTM) are employed. Diagonal BiLSTM is specifically designed for images. PixelRNN incorporates residual connections to help train deep neural networks with up to twelve layers. In PixelCNN, masked convolutions are used to adjust for the shape of the context. PixelRNN and PixelCNN are more dedicated to natural images. For example, PixelRNN and PixelCNN consider pixels as discrete values (e.g., 0, 1, ..., 255) and predict a multinomial distribution over the discrete values. Further, PixelRNN and PixelCNN deal with color images in RGB color space. In addition, PixelRNN and PixelCNN work well on the large-scale image dataset image network (ImageNet). In an example, a Gated PixelCNN is used to improve the PixelCNN. Gated PixelCNN achieves comparable performance with PixelRNN, but with much less complexity. In an example, a PixelCNN++ is employed with the following improvements upon PixelCNN: a discretized logistic mixture likelihood is used rather than a 256-way multinomial distribution; down-sampling is used to capture structures at multiple resolutions; additional short-cut connections are introduced to speed up training; dropout is adopted for regularization; and RGB is combined for one pixel. In another example, PixelSNAIL combines casual convolutions with self-attention.

[0082] Most of the above methods directly model the probability distribution in the pixel domain. Some designs also model the probability distribution as conditional based upon explicit or latent representations. Such a model can be expressed as:

$$p(x|h) = \prod_{i=1}^{m\times n} p(x_i|x_1, \ldots, x_{i-1}, h) \quad (3)$$

where h is the additional condition and $p(x) = p(h)p(x|h)$ indicates the modeling is split into an unconditional model and a conditional model. The additional condition can be image label information or high-level representations.

2.3.2 Auto-encoder

[0083] An Auto-encoder is now described. The auto-encoder is trained for dimensionality reduction and include an encoding component and a decoding component. The encoding component converts the high-dimension input signal to low-dimension representations. The low-dimension representations may have reduced spatial size, but a greater number of channels. The decoding component recovers the high-dimension input from the low-dimension representation. The auto-encoder enables automated learning of representations and eliminates the need of hand-crafted features, which is also believed to be one of the most important advantages of neural networks.

[0084] FIG. 4 is a schematic diagram illustrating an example transform coding scheme 400. The original image x is transformed by the analysis network g_a to achieve the latent representation y. The latent representation y is quantized (q) and compressed into bits. The number of bits R is used to measure the coding rate. The quantized latent representation $\hat{y}$ is then inversely transformed by a synthesis network g_s to obtain the reconstructed image $\hat{x}$. The distortion (D) is calculated in a perceptual space by transforming x and $\hat{x}$ with the function g_p, resulting in z and $\hat{z}$, which are compared to obtain D.

[0085] An auto-encoder network can be applied to lossy image compression. The learned latent representation can be encoded from the well-trained neural networks. However, adapting the auto-encoder to image compression is not trivial since the original auto-encoder is not optimized for compression, and is thereby not efficient for direct use as a trained auto-encoder. In addition, other major challenges exist. First, the low-dimension representation should be quantized before being encoded. However, the quantization is not differentiable, which is required in backpropagation while training the neural networks. Second, the objective under a compression scenario is different since both the distortion and the rate need to be take into consideration. Estimating the rate is challenging. Third, a practical image coding scheme should support variable rate, scalability, encoding/decoding speed, and interoperability. In response to these challenges, various schemes are under development.

[0086] An example auto-encoder for image compression using the example transform coding scheme 400 can be regarded as a transform coding strategy. The original image x is transformed with the analysis network $y = g_a(x)$, where y is the latent representation to be quantized and coded. The synthesis network inversely transforms the quantized

latent representation $\hat{y}$ back to obtain the reconstructed image $\hat{x} = g_s(\hat{y})$. The framework is trained with the rate-distortion loss function, $\mathcal{L} = D + \lambda R$, where D is the distortion between x and $\hat{x}$, R is the rate calculated or estimated from the quantized representation $\hat{y}$, and λ is the Lagrange multiplier. D can be calculated in either pixel domain or perceptual domain. Most example systems follow this prototype and the differences between such systems might only be the network structure or loss function.

[0087] In terms of network structure, RNNs and CNNs are the most widely used architectures. In the RNNs relevant category, an example general framework for variable rate image compression uses RNN. The example uses binary quantization to generate codes and does not consider rate during training. The framework provides a scalable coding functionality, where RNN with convolutional and deconvolution layers performs well. Another example offers an improved version by upgrading the encoder with a neural network similar to PixelRNN to compress the binary codes. The performance is better than JPEG on a Kodak image dataset using multi-scale structural similarity (MS-SSIM) evaluation metric. Another example further improves the RNN-based solution by introducing hidden-state priming. In addition, an SSIM-weighted loss function is also designed, and a spatially adaptive bitrates mechanism is included. This example achieves better results than better portable graphics (BPG) on the Kodak image dataset using MS-SSIM as evaluation metric. Another example system supports spatially adaptive bitrates by training stop-code tolerant RNNs.

[0088] Another example proposes a general framework for rate-distortion optimized image compression. The example system uses multiary quantization to generate integer codes and considers the rate during training. The loss is the joint rate-distortion cost, which can be mean square error (MSE) or other metrics. The example system adds random uniform noise to stimulate the quantization during training and uses the differential entropy of the noisy codes as a proxy for the rate. The example system uses generalized divisive normalization (GDN) as the network structure, which includes a linear mapping followed by a nonlinear parametric normalization. The effectiveness of GDN on image coding is verified. Another example system includes improved version that uses three convolutional layers each followed by a down-sampling layer and a GDN layer as the forward transform. Accordingly, this example version uses three layers of inverse GDN each followed by an up-sampling layer and convolution layer to stimulate the inverse transform. In addition, an arithmetic coding method is devised to compress the integer

codes. The performance is reportedly better than JPEG and JPEG 2000 on Kodak dataset in terms of MSE. Another example improves the method by devising a scale hyper-prior into the auto-encoder. The system transforms the latent representation y with a subnet h_a to $z = h_a(y)$ and z is quantized and transmitted as side information. Accordingly, the inverse transform is implemented with a subnet h_s that decodes from the quantized side information $\hat{z}$ to the standard deviation of the quantized $\hat{y}$, which is further used during the arithmetic coding of $\hat{y}$. On the Kodak image set, this method is slightly worse than BGP in terms of peak signal to noise ratio (PSNR). Another example system further exploits the structures in the residue space by introducing an autoregressive model to estimate both the standard deviation and the mean. This example uses a Gaussian mixture model to further remove redundancy in the residue. The performance is on par with VVC on the Kodak image set using PSNR as evaluation metric.

2.3.3 Hyper Prior Model

[0089] FIG. 5 illustrates example latent representations of an image. FIG. 5 includes an image 501 from the Kodak dataset, via isualization of the latent 502 representation y of the image 501, a standard deviations $\boldsymbol{\sigma}$ 503 of the latent 502, and latents y 504 after a hyper prior network is introduced. A hyper prior network includes a hyper encoder and decoder. In the transform coding approach to image compression, as shown in FIG. 4, the encoder subnetwork transforms the image vector x using a parametric analysis transform $g_a(x, \emptyset_g)$ into a latent representation $\boldsymbol{y}$, which is then quantized to form $\hat{\boldsymbol{y}}$. Because $\hat{\boldsymbol{y}}$ is discrete-valued, $\hat{\boldsymbol{y}}$ can be losslessly compressed using entropy coding techniques such as arithmetic coding and transmitted as a sequence of bits.

[0090] As evident from the latent 502 and the standard deviations $\boldsymbol{\sigma}$ 503 of FIG. 5, there are significant spatial dependencies among the elements of $\hat{\boldsymbol{y}}$. Notably, their scales (standard deviations $\boldsymbol{\sigma}$ 503) appear to be coupled spatially. An additional set of random variables $\hat{\boldsymbol{z}}$ may be introduced to capture the spatial dependencies and to further reduce the redundancies. In this case the image compression network is depicted in FIG. 6.

[0091] FIG. 6 is a schematic diagram 600 illustrating an example network architecture of an autoencoder implementing a hyperprior model. The upper side shows an image autoencoder network, and the lower side corresponds to the hyperprior subnetwork. The analysis and synthesis transforms are denoted as g_a and g_s, respectively. Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder,

respectively. The hyperprior model includes two subnetworks, hyper encoder (denoted with h_a) and hyper decoder (denoted with h_s). The hyper prior model generates a quantized hyper latent ($\hat{\boldsymbol{z}}$) which comprises information related to the probability distribution of the samples of the quantized latent $\hat{\boldsymbol{y}}$. $\hat{\boldsymbol{z}}$ is included in the bitstream and transmitted to the receiver (decoder) along with $\hat{\boldsymbol{y}}$.

[0092] In schematic diagram 600, the upper side of the models is the encoder g_a and decoder g_s as discussed above. The lower side is the additional hyper encoder h_a and hyper decoder h_s networks that are used to obtain $\hat{\boldsymbol{z}}$. In this architecture the encoder subjects the input image x to g_a, yielding the responses $\boldsymbol{y}$ with spatially varying standard deviations. The responses $\boldsymbol{y}$ are fed into h_a, summarizing the distribution of standard deviations in $\boldsymbol{z}$. $\boldsymbol{z}$ is then quantized ($\hat{\boldsymbol{z}}$), compressed, and transmitted as side information. The encoder then uses the quantized vector $\hat{\boldsymbol{z}}$ to estimate $\boldsymbol{\sigma}$, the spatial distribution of standard deviations, and uses $\boldsymbol{\sigma}$ to compress and transmit the quantized image representation $\hat{\boldsymbol{y}}$. The decoder first recovers $\hat{\boldsymbol{z}}$ from the compressed signal. The decoder then uses h_s to obtain $\boldsymbol{\sigma}$, which provides the decoder with the correct probability estimates to successfully recover $\hat{\boldsymbol{y}}$ as well. The decoder then feeds $\hat{\boldsymbol{y}}$ into g_s to obtain the reconstructed image.

[0093] When the hyper encoder and hyper decoder are added to the image compression network, the spatial redundancies of the quantized latent $\hat{\boldsymbol{y}}$ are reduced. The latents y 504 in FIG. 5 correspond to the quantized latent when the hyper encoder/decoder are used. Compared to the standard deviations $\boldsymbol{\sigma}$ 503, the spatial redundancies are significantly reduced as the samples of the quantized latent are less correlated.

2.3.4 Context Model

[0094] Although the hyperprior model improves the modelling of the probability distribution of the quantized latent $\hat{\boldsymbol{y}}$, additional improvement can be obtained by utilizing an autoregressive model that predicts quantized latents from their causal context, which may be known as a context model.

[0095] The term auto-regressive indicates that the output of a process is later used as an input to the process. For example, the context model subnetwork generates one sample of a latent, which is later used as input to obtain the next sample.

[0096] FIG. 7 is a schematic diagram 700 illustrating an example combined model

configured to jointly optimize a context model along with a hyperprior and the autoencoder. The combined model jointly optimizes an autoregressive component that estimates the probability distributions of latents from their causal context (Context Model) along with a hyperprior and the underlying autoencoder. Real-valued latent representations are quantized (Q) to create quantized latents ($\hat{\boldsymbol{y}}$) and quantized hyper-latents ($\hat{\boldsymbol{z}}$), which are compressed into a bitstream using an arithmetic encoder (AE) and decompressed by an arithmetic decoder (AD). The dashed region corresponds to the components that are executed by the receiver (e.g, a decoder) to recover an image from a compressed bitstream.

[0097] An example system utilizes a joint architecture where both a hyperprior model subnetwork (hyper encoder and hyper decoder) and a context model subnetwork are utilized. The hyperprior and the context model are combined to learn a probabilistic model over quantized latents $\hat{\boldsymbol{y}}$, which is then used for entropy coding. As depicted in schematic diagram 700, the outputs of the context subnetwork and hyper decoder subnetwork are combined by the subnetwork called Entropy Parameters, which generates the mean $\boldsymbol{\mu}$ and scale (or variance) $\boldsymbol{\sigma}$ parameters for a Gaussian probability model. The gaussian probability model is then used to encode the samples of the quantized latents into bitstream with the help of the arithmetic encoder (AE) module. In the decoder the gaussian probability model is utilized to obtain the quantized latents $\hat{\boldsymbol{y}}$ from the bitstream by arithmetic decoder (AD) module.

[0098] In an example, the latent samples are modeled as gaussian distribution or gaussian mixture models (not limited to). In the example according to the schematic diagram 700, the context model and hyper prior are jointly used to estimate the probability distribution of the latent samples. Since a gaussian distribution can be defined by a mean and a variance (aka sigma or scale), the joint model is used to estimate the mean and variance (denoted as $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$).

2.3.5 The encoding process using joint auto-regressive hyper prior model

[0099] The design in FIG 4. corresponds an example combined compression method. In this section and the next, the encoding and decoding processes are described separately.

[0100] FIG. 8 illustrates an example encoding process 800. The input image is first processed with an encoder subnetwork. The encoder transforms the input image into a transformed representation called latent, denoted by $\boldsymbol{y}$. $\boldsymbol{y}$ is then input to a quantizer block,

denoted by Q, to obtain the quantized latent ($\hat{\boldsymbol{y}}$). $\hat{\boldsymbol{y}}$ is then converted to a bitstream (bits1) using an arithmetic encoding module (denoted AE). The arithmetic encoding block converts each sample of the $\hat{\boldsymbol{y}}$ into a bitstream (bits1) one by one, in a sequential order.

[0101] The modules hyper encoder, context, hyper decoder, and entropy parameters subnetworks are used to estimate the probability distributions of the samples of the quantized latent $\hat{\boldsymbol{y}}$. the latent $\boldsymbol{y}$ is input to hyper encoder, which outputs the hyper latent (denoted by $\boldsymbol{z}$). The hyper latent is then quantized ($\hat{\boldsymbol{z}}$) and a second bitstream (bits2) is generated using arithmetic encoding (AE) module. The factorized entropy module generates the probability distribution, that is used to encode the quantized hyper latent into bitstream. The quantized hyper latent includes information about the probability distribution of the quantized latent ($\hat{\boldsymbol{y}}$).

[0102] The Entropy Parameters subnetwork generates the probability distribution estimations, that are used to encode the quantized latent $\hat{\boldsymbol{y}}$. The information that is generated by the Entropy Parameters typically include a mean $\boldsymbol{\mu}$ and scale (or variance) $\boldsymbol{\sigma}$ parameters, that are together used to obtain a gaussian probability distribution. A gaussian distribution of a random variable x is defined as $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ where the parameter $\boldsymbol{\mu}$ is the mean or expectation of the distribution (and also its median and mode), while the parameter $\boldsymbol{\sigma}$ is its standard deviation (or variance, or scale). In order to define a gaussian distribution, the mean and the variance need to be determined. The entropy parameters module are used to estimate the mean and the variance values.

[0103] The subnetwork hyper decoder generates part of the information that is used by the entropy parameters subnetwork, the other part of the information is generated by the autoregressive module called context module. The context module generates information about the probability distribution of a sample of the quantized latent, using the samples that are already encoded by the arithmetic encoding (AE) module. The quantized latent $\hat{\boldsymbol{y}}$ is typically a matrix composed of many samples. The samples can be indicated using indices, such as $\hat{\boldsymbol{y}}$[i,j,k] or $\hat{\boldsymbol{y}}$[i,j] depending on the dimensions of the matrix $\hat{\boldsymbol{y}}$. The samples $\hat{\boldsymbol{y}}$[i,j] are encoded by AE one by one, typically using a raster scan order. In a raster scan order the rows of a matrix are processed from top to bottom, where the samples in a row are processed from left to right. In such a scenario (where the raster scan order is used by the AE to encode the samples into bitstream), the context module generates the information pertaining to a sample $\hat{\boldsymbol{y}}$[i,j], using the samples encoded before, in raster scan

order. The information generated by the context module and the hyper decoder are combined by the entropy parameters module to generate the probability distributions that are used to encode the quantized latent $\hat{\boldsymbol{y}}$ into bitstream (bits1).

[0104] Finally, the first and the second bitstream are transmitted to the decoder as result of the encoding process. It is noted that the other names can be used for the modules described above.

[0105] In the above description, all of the elements in FIG. 8 are collectively called an encoder. The analysis transform that converts the input image into latent representation is also called an encoder (or auto-encoder).

2.3.6 The decoding process using joint auto-regressive hyper prior model

[0106] FIG. 9 illustrates an example decoding process 900. FIG. 9 depicts a decoding process separately.

[0107] In the decoding process, the decoder first receives the first bitstream (bits1) and the second bitstream (bits2) that are generated by a corresponding encoder. The bits2 is first decoded by the arithmetic decoding (AD) module by utilizing the probability distributions generated by the factorized entropy subnetwork. The factorized entropy module typically generates the probability distributions using a predetermined template, for example using predetermined mean and variance values in the case of gaussian distribution. The output of the arithmetic decoding process of the bits2 is $\hat{\boldsymbol{z}}$, which is the quantized hyper latent. The AD process reverts to AE process that was applied in the encoder. The processes of AE and AD are lossless, meaning that the quantized hyper latent $\hat{\boldsymbol{z}}$ that was generated by the encoder can be reconstructed at the decoder without any change.

[0108] After obtaining of $\hat{\boldsymbol{z}}$, it is processed by the hyper decoder, whose output is fed to entropy parameters module. The three subnetworks, context, hyper decoder and entropy parameters that are employed in the decoder are identical to the ones in the encoder. Therefore, the exact same probability distributions can be obtained in the decoder (as in encoder), which is essential for reconstructing the quantized latent $\hat{\boldsymbol{y}}$ without any loss. As a result, the identical version of the quantized latent $\hat{\boldsymbol{y}}$ that was obtained in the encoder can be obtained in the decoder.

[0109] After the probability distributions (e.g. the mean and variance parameters) are

obtained by the entropy parameters subnetwork, the arithmetic decoding module decodes the samples of the quantized latent one by one from the bitstream bits1. From a practical standpoint, autoregressive model (the context model) is inherently serial, and therefore cannot be sped up using techniques such as parallelization. Finally, the fully reconstructed quantized latent $\hat{\boldsymbol{y}}$ is input to the synthesis transform (denoted as decoder in FIG. 9) module to obtain the reconstructed image.

[0110] In the above description, the all of the elements in FIG. 9 are collectively called decoder. The synthesis transform that converts the quantized latent into reconstructed image is also called a decoder (or auto-decoder).

2.3.7 Wavelet based neural compression architecture

[0111] The analysis transform (denoted as encoder) in FIG. 8 and the synthesis transform (denoted as decoder) in FIG. 9 might be replaced by a wavelet based transform. FIG. 10 below shows an example such an implementation. In the figure first the input image is converted from an RGB color format to a YUV color format. This conversion process is optional, and can be missing in other implementations. If however such a conversion is applied at the input image, a back conversion (from YUV to RGB) is also applied before the output image is generated. Moreover there are 2 additional post processing modules (post-process 1 and 2) shown in the figure. These modules are also optional, hence might be missing in other implementations. The core of an encoder with wavelet-based transform is composed of a wavelet-based forward transform, a quantization module and an entropy coding module. After these 3 modules are applied to the input image, the bitstream is generated. The core of the decoding process is composed of entropy decoding, de-quantization process and an inverse wavelet-based transform operation. The decoding process convers the bitstream into output image. The encoding and decoding processes are depicted FIG. 10.

[0112] FIG. 10 illustrates an example encoder and decoder 1000 with wavelet-based transform.

[0113] After the wavelet-based forward transform is applied to the input image, in the output of the wavelet-based forward transform the image is split into its frequency components. The output of a 2-dimensional forward wavelet transform (depicted as iWave forward module in the figure above) might take the form depicted in FIG. 11. The input of the transform is an image of a castle. In the example, after the transform an output with

7 distinct regions are obtained. The number of distinct regions depend on the specific implementation of the transform and might different from 7. Potential number of regions are 4, 7, 10, 13, ...

[0114] FIG. 11 illustrates an example output 1100 of a forward wavelet-based transform.

[0115] In FIG. 11, the input image is transformed into 7 regions with 3 small images and 4 even smaller images. The transformation is based on the frequency components, the small image at the bottom right quarter comprises the high frequency components in both horizontal and vertical directions. The smallest image at the top-left corner on the other hand comprises the lowest frequency components both in the vertical and horizontal directions. The small image on the top-right quarter comprises the high frequency components in the horizontal direction and low frequency components in the vertical direction.

[0116] FIG. 12 illustrates an example partitioning 1200 of the output of a forward wavelet-based transform. FIG. 12 depicts a possible splitting of the latent representation after the 2D forward transform. The latent representation are the samples (latent samples, or quantized latent samples) that are obtained after the 2D forward transform. The latent samples are divided into 7 sections above, denoted as HH1, LH1, HL1, LL2, HL2, LH2 and HH2. The HH1 describes that the section comprises high frequency components in the vertical direction, high frequency components in the horizontal direction and that the splitting depth is 1. HL2 describes that the section comprises low frequency components in the vertical direction, high frequency components in the horizontal direction and that the splitting depth is 2.

[0117] After the latent samples are obtained at the encoder by the forward wavelet transform, they are transmitted to the decoder by using entropy coding. At the decoder, entropy decoding is applied to obtain the latent samples, which are then inverse transformed (by using iWave inverse module in FIG. 10) to obtain the reconstructed image.

2.4 Neural Networks for Video Compression

[0118] Similar to video coding technologies, neural image compression serves as the foundation of intra compression in neural network-based video compression. Thus, development of neural network-based video compression technology is behind development of neural network-based image compression because neural network-based

video compression technology is of greater complexity and hence needs far more effort to solve the corresponding challenges. Compared with image compression, video compression needs efficient methods to remove inter-picture redundancy. Inter-picture prediction is then a major step in these example systems. Motion estimation and compensation is widely adopted in video codecs, but is not generally implemented by trained neural networks.

[0119] Neural network-based video compression can be divided into two categories according to the targeted scenarios: random access and the low-latency. In random access case, the system allows decoding to be started from any point of the sequence, typically divides the entire sequence into multiple individual segments, and allows each segment to be decoded independently. In a low-latency case, the system aims to reduce decoding time, and thereby temporally previous frames can be used as reference frames to decode subsequent frames.

2.4.1 Low-latency

[0120] An example system employs a video compression scheme with trained neural networks. The system first splits the video sequence frames into blocks and each block is coded according to an intra coding mode or an inter coding mode. If intra coding is selected, there is an associated auto-encoder to compress the block. If inter coding is selected, motion estimation and compensation are performed and a trained neural network is used for residue compression. The outputs of auto-encoders are directly quantized and coded by the Huffman method.

[0121] Another neural network-based video coding scheme employs PixelMotionCNN. The frames are compressed in the temporal order, and each frame is split into blocks which are compressed in the raster scan order. Each frame is first extrapolated with the preceding two reconstructed frames. When a block is to be compressed, the extrapolated frame along with the context of the current block are fed into the PixelMotionCNN to derive a latent representation. Then the residues are compressed by a variable rate image scheme. This scheme performs on par with H.264.

[0122] Another example system employs an end-to-end neural network-based video compression framework, in which all the modules are implemented with neural networks. The scheme accepts a current frame and a prior reconstructed frame as inputs. An optical flow is derived with a pre-trained neural network as the motion information. The motion

information is warped with the reference frame followed by a neural network generating the motion compensated frame. The residues and the motion information are compressed with two separate neural auto-encoders. The whole framework is trained with a single rate-distortion loss function. The example system achieves better performance than H.264.

[0123] Another example system employs an advanced neural network-based video compression scheme. The system inherits and extends video coding schemes with neural networks with the following major features. First the system uses only one auto-encoder to compress motion information and residues. Second, the system uses motion compensation with multiple frames and multiple optical flows. Third, the system uses an on-line state that is learned and propagated through the following frames over time. This scheme achieves better performance in MS-SSIM than HEVC reference software.

[0124] Another example system uses an extended end-to-end neural network-based video compression framework. In this example, multiple frames are used as references. The example system is thereby able to provide more accurate prediction of a current frame by using multiple reference frames and associated motion information. In addition, a motion field prediction is deployed to remove motion redundancy along temporal channel. Postprocessing networks are also used to remove reconstruction artifacts from previous processes. The performance of this system is better than H.265 by a noticeable margin in terms of both PSNR and MS-SSIM.

[0125] Another example system uses scale-space flow to replace an optical flow by adding a scale parameter based on a framework. This example system may achieve better performance than H.264. Another example system uses a multi-resolution representation for optical flows based. Concretely, the motion estimation network produces multiple optical flows with different resolutions and let the network learn which one to choose under the loss function. The performance is slightly better than H.265.

2.4.2 Random Access

[0126] Another example system uses a neural network-based video compression scheme with frame interpolation. The key frames are first compressed with a neural image compressor and the remaining frames are compressed in a hierarchical order. The system performs motion compensation in the perceptual domain by deriving the feature maps at multiple spatial scales of the original frame and using motion to warp the feature maps. The results are used for the image compressor. The method is on par with H.264.

[0127] An example system uses a method for interpolation-based video compression. The interpolation model combines motion information compression and image synthesis. The same auto-encoder is used for image and residual. Another example system employs a neural network-based video compression method based on variational auto-encoders with a deterministic encoder. Concretely, the model includes an auto-encoder and an auto-regressive prior. Different from previous methods, this system accepts a group of pictures (GOP) as inputs and incorporates a three dimensional (3D) autoregressive prior by taking into account of the temporal correlation while coding the latent representations. This system provides comparative performance as H.265.

2.5 Preliminaries

[0128] Almost all the natural image and/or video is in digital format. A grayscale digital image can be represented by $x \in \mathbb{D}^{m \times n}$, where $\mathbb{D}$ is the set of values of a pixel, m is the image height, and n is the image width. For example, $\mathbb{D} = \{0, 1, 2, \ldots, 255\}$ is an example setting, and in this case $|\mathbb{D}| = 256 = 2^8$. Thus, the pixel can be represented by an 8-bit integer. An uncompressed grayscale digital image has 8 bits-per-pixel (bpp), while compressed bits are definitely less.

[0129] A color image is typically represented in multiple channels to record the color information. For example, in the RGB color space an image can be denoted by $x \in \mathbb{D}^{m \times n \times 3}$ with three separate channels storing Red, Green, and Blue information. Similar to the 8-bit grayscale image, an uncompressed 8-bit RGB image has 24 bpp. Digital images/videos can be represented in different color spaces. The neural network-based video compression schemes are mostly developed in RGB color space while the video codecs typically use a YUV color space to represent the video sequences. In YUV color space, an image is decomposed into three channels, namely luma (Y), blue difference choma (Cb) and red difference chroma (Cr). Y is the luminance component and Cb and Cr are the chroma components. The compression benefit to YUV occur because Cb and Cr are typically down sampled to achieve pre-compression since human vision system is less sensitive to chroma components.

[0130] A color video sequence is composed of multiple color images, also called frames, to record scenes at different timestamps. For example, in the RGB color space, a color video can be denoted by $X = \{x_0, x_1, \ldots, x_t, \ldots, x_{T-1}\}$ where T is the number of frames in a video sequence and $x \in \mathbb{D}^{m \times n}$. If $m = 1080$, $n = 1920$, $|\mathbb{D}| = 2^8$, and the video has 50

frames-per-second (fps), then the data rate of this uncompressed video is $1920 \times 1080 \times 8 \times 3 \times 50 = 2{,}488{,}320{,}000$ bits-per-second (bps). This results in about 2.32 gigabits per second (Gbps), which uses a lot storage and should be compressed before transmission over the internet.

[0131] Usually the lossless methods can achieve a compression ratio of about 1.5 to 3 for natural images, which is clearly below streaming requirements. Therefore, lossy compression is employed to achieve a better compression ratio, but at the cost of incurred distortion. The distortion can be measured by calculating the average squared difference between the original image and the reconstructed image, for example based on MSE. For a grayscale image, MSE can be calculated with the following equation.

$$MSE = \frac{\|x-\hat{x}\|^2}{m \times n} \tag{4}$$

[0132] Accordingly, the quality of the reconstructed image compared with the original image can be measured by peak signal-to-noise ratio (PSNR):

$$PSNR = 10 \times log_{10} \frac{(max(\mathbb{D}))^2}{MSE} \tag{5}$$

where $max(\mathbb{D})$ is the maximal value in $\mathbb{D}$, e.g., 255 for 8-bit grayscale images. There are other quality evaluation metrics such as structural similarity (SSIM) and multi-scale SSIM (MS-SSIM).

[0133] To compare different lossless compression schemes, the compression ratio given the resulting rate, or vice versa, can be compared. However, to compare different lossy compression methods, the comparison has to take into account both the rate and reconstructed quality. For example, this can be accomplished by calculating the relative rates at several different quality levels and then averaging the rates. The average relative rate is known as Bjontegaard's delta-rate (BD-rate). There are other aspects to evaluate image and/or video coding schemes, including encoding/decoding complexity, scalability, robustness, and so on.

3. Technical problems solved by disclosed technical solutions

3.1 The core problem

Mask and Scale process is common for three tools: Residual and Variance Scale (RVS), Skip Mode (Skip) and Latent Scale Before Synthesis (LSBS). The syntax elements are not concise enough. We design new syntax elements and decoding logics to improve the coding efficiency.

3.2 Technical Solutions

3.2.1 Mask generation

The masking generation is a core function which is used by all three aforementioned coding tools.

The input of the mask generation core function is the tensor with sigma samples σ (σ_Y and σ_{UV} in primary and secondary components coding pipelines) of size $[C, h_4, w_4]$. The output is a mask $mask[C, h_4, w_4]$ to be used by one or more of the aforementioned coding tools (those that are enabled).

To generate the mask, five syntax elements are used, namely $ThresholdRVS, ThreshodSkip, ThresholdLSBS$, $GreaterFlag$, and $Log2BlockSize$, that are included in the Picture Header.

Mask generation is illustrated in following process.

According to the $Log2BlockSize$, the $BlockSize$ is calculated as

- $BlockSize = 2^{Log2BlockSize}$.

$Threshold$ is set as $ThresholdRVS, ThreshodSkip$, or $ThresholdLSBS$ based on current mode.

First step is pooling. If the $BlockSize$ is greater than 1, a pooling operation is applied to the input sigma samples tensor first. The pooling operation is average pooling, with a kernel size equal to $BlockSize$ in horizontal and vertical dimension.

Pooling process generated pooled sigma tensor σ_p of size $[C, h_p, w_p]$, with $h_p = ceil(h_4/BlockSize), w_p = ceil(w_4/BlockSize)$. If the $BlockSize$ is equal to 1, the size of the pooled sigma samples tensor size is equal to size of the variance values tensor.

Afterwards each one of the pooled sigma samples are compared with the $Threshold$, and the comparison is stored in a pooled mask tensor mask_p. The pooled mask samples are obtained according to the following:

- For $c = c = 0..C-1, i = 0..h_p - 1$ and $j = 0..w_p - 1$

$$mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap Greater == False \\ False \ Otherwise \end{cases}.$$

Specifically, the $GreaterFlag$ of Skip mode is always inferred as true. After the pooled mask samples tensor mask_p is obtained, and if the *BlockSize* is greater than 1, an up-sampling operation is applied to mask_p to obtain the final mask samples tensor. The up-sampling operation is based on nearest neighbor. If the *BlockSize* is equal to 1, the up-sampling operation is skipped. If the *BlockSize* is greater than 1, a cropping operation is applied after up-sampling resulting in an output mask tensor with size $[C, h_4, w_4]$:

$mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

3.2.2 Residual and Variance Scale (RVS)

This module scales both the residual and the variance parameter used to create the entropy coding model. Residual and variance scaling work together and share the same scaling factors. The position of residual scaling is after Gain Unit on encoder side. The position of inverse residual scaling is right after inverse Gain Unit. Variance scaling is located after Hyper Scale Decoder. The process of RVS achieves adaptive quantization of residual samples based on their corresponding variance value.

Residual and Variance Scaling (RVS) uses several sets of control parameters, defined by $numRVSparams$ (signalled to the decoder in Picture Header). The first four sets parameters which are $ApplicationList[numRVSparams]$, $ThresholdRVS[numRVSparams]$, $GreaterFlag[numRVSparams]$, and $Log2BlockSize[numRVSparams]$ are used to generate the mask as described in section G.2. The fourth is a scale factor $Scale[numRVSparams]$.

At the decoder, the input of the RVS are the residual tensor $\hat{r}[C, h_4, w_4]$ after inverse gain unit function and variance $\sigma[C, h_4, w_4]$ and binary mask $mask[numRVSparams][\ C, h_4, w_4]$ generated as described in section G.2 for $numRVSparams$ sets of parameters. When $ApplicationList[numRVSparams]$ equals to 0, the RVS applies to luma component only. When $ApplicationList[numRVSparams]$ equals to 1, the RVS applies to chroma component only. When $ApplicationList[numRVSparams]$ equals to 2, the RVS applies to both luma and chroma components.

The process of RVS at the decoder is as follows:

- First the σ_{temp} and $\hat{r}_{temp}$ tensors are initialized to be equal to variance tensor σ and quantized residual tensor $\hat{r}$ respectively.
- For $idx = 0..numRVSparams - 1$ the following ordered steps are applied:
  - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with $Threshold[idx]$, $GreaterFlag[idx]$, $Log2BlockSize[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
  - Each sample of the modified residual tensor and modified sigma tensor are obtained as follows:

$$\hat{r}_{temp}[c,i,j] = \begin{cases} \hat{r}_{temp}[c,i,j] \cdot ScaleRVS[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \hat{r}_{temp}[c,i,j] & overwise \end{cases}$$

$$\sigma_{temp}[c,i,j] = \begin{cases} \sigma_{temp}[c,i,j] \cdot ScaleRVS[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \sigma_{temp}[c,i,j] & overwise \end{cases}$$

$c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$.

RVS process outputs modified variance tensor $\sigma[C, h_4, w_4]$ and modified residual tensor

$\hat{r}[C, h_4, w_4]$ which are set to σ_{temp} and $\hat{r}_{temp}$ respectively.

3.2.3 Skip Mode

The residual skip process uses several set of control parameters, defined by $numSkipparams$ (signalled to the decoder in Picture Header). Two sets of parameters $ThresholdSkip[numSkipparams]$ and $Log2BlockSize[numSkipparams]$ are used to define the mask as described in section G.2. When $ApplicationList[numSkipparams]$ equals to 0, the Skip mode applies to luma component only. When $ApplicationList[numSkipparams]$ equals to 1, the Skip mode applies to chroma component only. When $ApplicationList[numSkipparams]$ equals to 2, the Skip mode applies to both luma and chroma components.

At the decoder, the inputs of skip mode process are the 1D $\{s'_m\}$ after the entropy decoding process of steam#2 C.7, $mask$ computed as described is section G.2 using the variance tensor σ after the hyper scale decoding process.

The output of the lossless decoding process is a 1D array $\{s'_m\}$, whose size is equal to the total number of "1"s in the $maskAggregate$ tensor. In other words, the $maskAggregate$ tensor determines which samples of the residual tensor $\hat{r}$ are included in the bitstream. All of the other samples of the quantized residual tensor are inferred to be equal to zero. The process of residual skip mode at the decoder is as follows:

- Tensors $\hat{r}$ and $maskAggregate$ are initialized to be equal to all zeros and all ones respectively;
- The counter $k = 0$;
- Dimensions $[C, h_4, w_4]$ are set equal to number of channels, height and width of the sigma tensor σ ($C = C_p = 128$, $h_4 = h_{4,Y}, w_4 = w_{4,Y}$ for primary component, $C = C_s = 64$, $h_4 = h_{4,UV}, w_4 = w_{4,UV}$ for secondary component);
- For $idx = 0..numSkipparams - 1$ the following ordered steps are applied:
  - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with $ThresholdSkip[idx]$, $GreaterFlag[idx]$, $Log2BlockSize[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
  - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$
    $$maskAggregate[c, i, j] = maskAggregate[c, i, j] \cdot mask[idx][c, i, j]$$
  - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$
    $$\hat{r}[c, i, j] = maskAggregate[c, i, j] \cdot s_k; k = k + 1$$

The output of this process is the residual tensor $\hat{r}$.

3.2.4 Latent Scale Before Synthesis

Latent Scale Before Synthesis (LSBS) uses several sets of control parameters, defined by $numLSBSparams$ (signalled to the decoder in Picture Header). The first four sets parameters which are $ApplicationList[numLSBSparams]$, $ThresholdLSBS[numLSBSparams]$,

$GreaterFlag[numLSBSparams]$, and $Log2BlockSize[numLSBSparams]$ are used to generate the mask as described in section G.2. The fourth and fifth set of parameters are scale factor $ScaleLSBS1[numLSBSparams]$ and $ScaleLSBS2[numLSBSparams]$. LSBS process is applied consecutively $numLSBSparams$ times. When $ApplicationList[numLSBSparams]$ equals to 0, the LSBS mode applies to luma component only. When $ApplicationList[numLSBSparams]$ equals to 1, the LSBS mode applies to chroma component only. When $ApplicationList[numLSBSparams]$ equals to 2, the LSBS mode applies to both luma and chroma components.

At the decoder, the input of the LSBS process are the residual tensor $\hat{r}[C, h_4, w_4]$ after entropy decoding (and Skip Mode if applicable), the prediction tensor $\mu[C, h_4, w_4]$ after the prediction fusion process, latent tensor $\hat{y}[C, h_4, w_4]$, and binary mask generated using variance σ as described in section G.2.The process of LSBS at the decoder is as follows:

- Tensors $\hat{y}_{temp}, \hat{r}_{temp}, \mu_{temp}$ are initialized to be equal to latent samples tensor $\hat{y}$, residual tensor $\hat{r}$ and prediction tensor μ respectively.
- For $idx = 0..numLSBSparams - 1$ the following steps are applied:
  - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with $ThresholdLSBS[idx]$, $GreaterFlag[idx]$, $Log2BlockSize[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
  - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$

$$\hat{y}_{temp}[c,i,j] = \hat{y}_{temp}[c,i,j] + (\, mask[idx][c,i,j] > 0 ? \mu[c,i,j] \cdot ScaleLSBS1[idx] + \hat{r}[c,i,j] \cdot ScaleLSBS2[idx] : 0)$$

The output of this process is the modified latent tensor $\hat{y}$, which is set equal to $\hat{y}_{temp}$.

3.2.5 Parameters signalling

Following syntax elements are included into Picture Header:

rvs_enable_flag – 1-bit binary value specifying the on/off status of RVS mode. 0 indicates disabling RVS mode for luma and chroma components. 1 indicates enabling RVS mode for luma and chroma components.

skip_enable_flag – 1-bit binary value specifying the on/off of Skip mode. 0 indicates disabling skip mode for luma and chroma components. 1 indicates enabling skip mode for luma and chroma components.

lsbs_enable_falg – 1-bit binary value specifying the on/off of LSBS mode. 0 indicates disabling LSBS mode for luma and chroma components. 1 indicates enabling LSBS mode for luma and chroma components.

numRVSparams – 3-bit unsigned integer, the number of parameters sets used in the adaptive quantization process, controlling the quantization of the residuals.

numSkipParams – 3-bit unsigned integer specifying the number of parameters sets used in the block-based skipping process. If a first filter OR the second filter decides to skip a sample, that sample is skipped.

numLSBSparams – 3-bit unsigned integer specifying the number of parameters sets used in the latent domain masking and scaling, determine scaling at the decoder after $\hat{y}$ is reconstructed.

applicationList – 2-bit unsigned integer. 0 indicates parameter set is applied to luma component, 1 indicates parameter set is applied to chroma component, 2 indicates parameter set applied to both components.

ScaleRVS – 8-bit unsigned integer specifying the value of the multiplier to be used in processing samples of RVS mode.

ScaleLSBS – 10-bit unsigned integer specifying the value of the multiplier to be used in processing samples of LSBS mode.

ThresholdRVS – 8-bit unsigned integer specifying the value of the threshold when using RVS mode.

ThresholdSkip –16-bit unsigned integer specifying the value of the threshold when using Skip mode.

ThresholdLSBS –8-bit unsigned integer specifying the value of the threshold when using LSBS mode.

GreaterFlag – 1-bit binary value specifying whether a thresholding operation is to be applied as greater than or smaller than a threshold.

PreciseFlag – 1-bit binary value specifying the precision of the Scale and Threshold syntax elements.

Log2BlockSize – 3-bit unsigned integer specifying the logarithm of resampling block size.

4. Detailed solutions

The detailed solutions below should be considered as examples to explain general concepts. These solutions should not be interpreted in a narrow way. Furthermore, these solutions can be combined in any manner.

4.1 Core of the present disclosure

The target of the disclosure is to improve the reconstruction capability of the synthesis transform module with constraint on computational resources. The core of the present disclosure is to simplify the synthesis transform module while maintaining the reconstruction capability. The structure of attention module and the position of attention module may be modified.

4.2 Details of the present disclosure

1. Enable flags are included in picture header for RVS mode, skip mode and LSBS mode:
    - **rvs_enable_flag** – 1-bit binary value specifying the on/off status of RVS mode. 0 indicates disabling RVS mode for luma and chroma components. 1 indicates enabling RVS mode for luma and chroma components.

- **skip_enable_flag** – 1-bit binary value specifying the on/off of Skip mode. 0 indicates disabling skip mode for luma and chroma components. 1 indicates enabling skip mode for luma and chroma components.

- **lsbs_enable_falg** – 1-bit binary value specifying the on/off of LSBS mode. 0 indicates disabling LSBS mode for luma and chroma components. 1 indicates enabling LSBS mode for luma and chroma components.

2. Numbers of parameter sets are signaled for RVS mode and LSBS mode with the following syntax. The number of parameters sets of the skip mode is always inferred as 1.

- **num_rvs_params** – 3-bit unsigned integer, the number of parameters sets used in the adaptive quantization process, controlling the quantization of the residuals.

- **num_lsbs_params** – 3-bit unsigned integer specifying the number of parameters sets used in the latent domain masking and scaling, determine scaling at the decoder after $\hat{y}$ is reconstructed.

3. To denote whether the mode is applied to luma or chroma or both luma and chroma, application flags are designed for RVS, and LSBS mode. When **rvs_enable_flag** is zero, the decoder will not parse **application_flag_rvs**. When **lsbs_enable_falg** is zero, the decoder will not parse **application_flag_lsbs**.

- **application_flag_rvs** – 2-bit unsigned integer. 0 indicates RVS parameter set is applied to luma component, 1 indicates RVS parameter set is applied to chroma component, 2 indicates RVS parameter set applied to both components.

- **application_flag_lsbs** – 2-bit unsigned integer. 0 indicates LSBS parameter set is applied to luma component, 1 indicates LSBS parameter set is applied to chroma component, 2 indicates LSBS parameter set applied to both components.

4. To denote whether the skip mode is applied to luma or chroma or both luma and chroma, **the skip_mode_idx** syntax is designed. When **skip_enable_flag is zero.** the decoder will not parse **skip_mode_idx.**

- **skip_mode_idx** – 2-bit unsigned integer. 0 indicates Skip mode parameter set is applied to luma component, 1 indicates Skip mode parameter set is applied to chroma component. Value of 2 and 3 indicates Skip mode parameter set applied to both components.

5. Scale parameters of RVS mode, and LSBS mode are signaled with the following syntax elements. The precisions can be additionally signaled or inferred.

- **scale_rvs** – 16-bit or 8-bit unsigned integer specifying the value of the multiplier to be used in processing samples of RVS mode.

- **scale1_lsbs** – 14-bit unsigned integer specifying the value of the multiplier to be used in processing samples of LSBS mode.

- **scale2_lsbs** – 14-bit unsigned integer specifying the value of the multiplier to be used in processing samples of LSBS mode.

6. Threshold values of RVS mode, skip mode and LSBS mode are signaled with the following syntax elements. The precisions can be additionally signaled or inferred.

   - **thr_rvs** – 12-bit or 9-bit unsigned integer specifying the value of the threshold when using RVS mode.

   - **thr_skip** –16-bit or 8-bit unsigned integer specifying the value of the threshold when using Skip mode.

   - **thr_lsbs** –12-bit or 9-bit unsigned integer specifying the value of the threshold when using LSBS mode.

7. Greater flags of RVS mode and LSBS mode will be signaled with the following syntax elements. The Greater flag of skip mode is inferred to be true without signaling.

   - **greater_flag_rvs** – 1-bit binary value specifying whether a thresholding operation is to be applied as greater than or smaller than a threshold for RVS mode.

   - **greater_flag_lsbs** – 1-bit binary value specifying whether a thresholding operation is to be applied as greater than or smaller than a threshold for LSBS mode.

8. Resampling block size of RVS mode, skip mode and LSBS mode are signaled. The block size is denoted in log domain.

   - **log2_block_size_rvs** – 3-bit unsigned integer specifying the logarithm of resampling block size of RVS mode.

   - **log2_block_size_skip** – 3-bit unsigned integer specifying the logarithm of resampling block size of Skip mode.

   - **log2_block_size_lsbs** – 3-bit unsigned integer specifying the logarithm of resampling block size of LSBS mode.

9. The masking generation is a core function which is used by all three aforementioned coding tools. The input of the mask generation core function is the tensor with sigma samples σ (σ_Y and σ_{UV} in primary and secondary components coding pipelines) of size $[C, h_4, w_4]$. The output is a mask $mask[C, h_4, w_4]$ to be used by one or more of the aforementioned coding tools (those that are enabled). To generate the mask, three syntax elements are used, namely $Threshold$, $GreaterFlag$ and $Log2BlockSize$.
   a) In one example, the mask generation process is illustrated as follows. According to the $Log2BlockSize$, the $BlockSize$ is calculated as

– $BlockSize = 2^{Log2BlockSize}$.

First step is pooling. If the $BlockSize$ is greater than 1, a pooling operation is applied to the input sigma samples tensor first. The pooling operation is based on the specific mode. With RVS or LSBS mode, average pooling is used. With Skip mode, max pooling is used. A kernel size equal to $BlockSize$ in horizontal and vertical dimension is used when conducting pooling operation.

Pooling process generated pooled sigma tensor σ_p of size $[C, h_p, w_p]$, with $h_p = ceil(h_4/BlockSize), w_p = ceil(w_4/BlockSize)$. If the $BlockSize$ is equal to 1, the size of

the pooled sigma samples tensor size is equal to size of the variance values tensor.

Afterwards each one of the pooled sigma samples are compared with the $Threshold$, and the comparison is stored in a pooled mask tensor mask_p. The pooled mask samples are obtained according to the following:

- For $c = c = 0..C-1, i = 0..h_p - 1$ and $j = 0..w_p - 1$

$$mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False \ Otherwise \end{cases}.$$

After the pooled mask samples tensor mask_p is obtained, and if the *BlockSize* is greater than 1, an up-sampling operation is applied to mask_p to obtain the final mask samples tensor. The up-sampling operation is based on nearest neighbor. If the *BlockSize* is equal to 1, the up-sampling operation is skipped.

$$mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ], \quad c = 0..C-1, i = 0..h_4 - 1 \text{ and } j = 0..w_4 - 1.$$

10. RVS mode is designed as follows. RVS mode scales both the residual and the variance parameter used to create the entropy coding model. Residual and variance scaling work together and share the same scaling factors. The position of residual scaling is after Gain Unit on encoder side. The position of inverse residual scaling is right after inverse Gain Unit. Variance scaling is located after Hyper Scale Decoder. The process of RVS achieves adaptive quantization of residual samples based on their corresponding variance value.

Residual and Variance Scaling (RVS) uses a maximum 8 sets of control parameters, defined by num_rvs_params(signalled to the decoder in Picture Header).

At the decoder, the input of the RVS are the residual tensor $\hat{r}[C, h_4, w_4]$ after inverse gain unit function and variance tensor $\sigma[C, h_4, w_4]$.

The process of RVS at the decoder is as follows:

- First the σ_{temp} and $\hat{r}_{temp}$ tensors are initialized to be equal to variance tensor σ and quantized residual tensor $\hat{r}$ respectively.
  - if rvs_enable_flag is equal to 1;For $idx = 0..num\_rvs\_params - 1$ the following ordered steps are applied:
    - if $application\_flag\_rvs[idx]$ is not equal to 0 and the current component is secondary component, or if $application\_flag\_rvs[idx]$ is not equal to 1 and the current component is primary component;
      - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with thr_rvs$[idx]$, greater_flag_rvs$[idx]$, $log2\_block\_size\_rvs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
      - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$, the modified residual tensor and modified sigma tensor are obtained as follows:

- $\hat{r}_{temp}[c,i,j] =$
$\begin{cases} \hat{r}_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \hat{r}_{temp}[c,i,j] & overwise \end{cases}$

- $\sigma_{temp}[c,i,j] =$
$\begin{cases} \sigma_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \sigma_{temp}[c,i,j] & overwise \end{cases}$

The output of the RVS process is the modified variance tensor $\sigma[C, h_4, w_4]$ and modified residual tensor $\hat{r}[C, h_4, w_4]$ which are set to σ_{temp} and $\hat{r}_{temp}$ respectively.

11. The residual skip process uses maximum 2 sets of control parameters, defined by $skip\_mode\_idx$(signalled to the decoder in Picture Header). At the decoder, the inputs of skip mode process are the 1D $\{s'_m\}$ after the entropy decoding process of steam#2 C.7, $mask$ computed as described is section G.2 using the variance tensor σ after the hyper scale decoding process. The output of the lossless decoding process is a 1D array $\{s'_m\}$, whose size is equal to the total number of "1"s in the $mask$ tensor. In other words, the $mask$ tensor determines which samples of the residual tensor $\hat{r}$ are included in the bitstream. All of the other samples of the quantized residual tensor are inferred to be equal to zero. The process of residual skip mode at the decoder is as follows:
   - Tensors $\hat{r}$ is initialized to be equal to all zeros.
   - The counter k is set equal to 0.
   - Dimensions $[C, h_4, w_4]$ are set equal to number of channels, height and width of the sigma tensor σ ($C = C_p = 128$, $h_4 = h_{4,Y}, w_4 = w_{4,Y}$ for primary component, $C = C_s = 64$, $h_4 = h_{4,UV}, w_4 = w_{4,UV}$ for secondary component).
   - idx is set equal to 0 if current component is primary component or 0 otherwise.
   - if $skip\_mode\_idx$ is not equal to (1- idx);
      - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with thr_skip$[idx]$, $log2\_block\_size\_skip[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
      - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$;
         - $\hat{r}[c,i,j] =$
$\begin{cases} s_k, & mask[idx][c,i,j] == True\ \ OR\ \ skip\_enable\_flag == False \\ 0, & otherwise \end{cases}$
         - $k = k + 1$.

The output of this process is the residual tensor $\hat{r}$.

12. Latent Scale Before Synthesis (LSBS) uses a maximum 8 sets of control parameters, defined by num_lsbs_params (signalled to the decoder in Picture Header). At the decoder, the input of the LSBS process are the residual tensor $\hat{r}[C, h_4, w_4]$ after entropy decoding (and Skip Mode if applicable), the prediction tensor $\mu[C, h_4, w_4]$ after the prediction fusion process, latent tensor $\hat{y}[C, h_4, w_4]$, and binary mask generated using variance σ as described in section G.2. The process of LSBS at the decoder is as follows:
   - Tensors $\hat{y}_{temp}, \hat{r}_{temp}, \mu_{temp}$ are initialized to be equal to latent samples tensor $\hat{y}$, residual tensor $\hat{r}$ and prediction tensor μ respectively.

- For $idx = 0..num\_lsbs\_params - 1$ the $\hat{y}_{temp}$ is modified as follows:
  - if lsbs_enable_flag is equal to True;
    - if $application\_flag\_lsbs[idx]$ is not equal to 0 and current component is primary component or $application\_flag\_lsbs[idx]$ is not equal to 1 and current component is secondary component;
      - A tensor $mask[idx]$ is generated using the mask generation core function G.2 with $thr\_lsbs[idx]$, $greater\_flag\_lsbs[idx]$, $log2\_block\_size\_lsbs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output.
      - For $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$
        - $\hat{y}_{temp}[c,i,j] = \hat{y}_{temp}[c,i,j] + (mask[idx][c,i,j] == True\ ?\ \mu[c,i,j] \cdot scale1\_lsbs[idx] + \hat{r}[c,i,j] \cdot scale2\_lsbs[idx] : 0)$.

The output of this process is the modified latent tensor $\hat{y}$, which is set equal to $\hat{y}_{temp}$.

13. Syntax table of mask and scale tools are designed as follows.

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to adaptBin(A, B).

Inputs to this process are bits from the RBSP.

Outputs of this process nomDenomPair.

AdaptBin(nomRange1, nomRange2){	
if A != B	
precision_flag	uf(1)
else	
precision_flag = 0	
if precision flag	
nominator	uf(nomRange1)
denominator = nomRange1 - 3	
else	
nominator	uf(nomRange2)
denominator = nomRange2 - 3	
nomDenomPair[0] = nominator	
nomDenomPair[1] = denominator	

When the AdaptBin(A, B) is invoked,precision_flag and value are parsed. The nomDenomPair is obtained according to the value and precision_flag as described in table above.

Syntax Table

MaskScale(){	
rvs_enable_flag	uf(1)
skipmode_enable_flag	uf(1)
lsbs_enable_flag	uf(1)

if rvs_enable_flag{	
num_rvs_ params	uf(3)
for(Idx = 0; Idx < num_rvs_ params + 1; Idx ++) {	
log2_block_size_rvs [Idx]	uf(3)
greater_flag_rvs [Idx]	uf(1)
application_flag_rvs [Idx]	uf(2)
thr_rvs [Idx]	AdaptBin(12, 9)
scale_rvs [Idx]	AdaptBin(16, 8)
}	
if skipmode_enable_flag{	
skip_mode_idx	uf(2)
if skip_mode_idx == 0 or skip_mode_idx == 2 or or skip_mode_idx == 3	
log2_block_size_skip [0]	uf(3)
thr_skip [0]	AdaptBin(16, 8)
if skip_mode_idx ==2	
log2_block_size_skip [0] = log2_block_size_skip [1]	
thr_skip [0] = thr_skip [1]	
if skip_mode_idx == 1 or skip_mode_idx == 3	
log2_block_size_skip [1]	uf(3)
thr_skip [1]	AdaptBin(16, 8)
}	
if lsbs_enable_flag{	
num_lsbs_ params	uf(3)
for(Idx = 0; Idx < num_lsbs_ params + 1; Idx ++) {	
log2_block_size_lsbs [Idx]	uf(3)
greater_flag_lsbs [Idx]	uf(1)
application_flag_lsbs [Idx]	uf(2)
thr_lsbs [Idx]	AdaptBin(12, 9)
scale1_lsbs [Idx]	AdaptBin(14, 14)
scale2_lsbs [Idx]	AdaptBin(14, 14)
}	
}	

General aspects

1. Whether to and/or how to apply the disclosed methods above may be signalled at block level/ sequence level/group of pictures level/picture level/slice level/tile group level, such

as in coding structures of CTU/CU/TU/PU/CTB/CB/TB/PB, or sequence header/picture header/SPS/VPS/DPS/DCI/PPS/APS/slice header/tile group header.

2. Whether to and/or how to apply the disclosed methods above may be dependent on coded information, such as block size, colour format, single/dual tree partitioning, colour component, slice/picture type.
3. The proposed methods disclosed in this document may be used in other coding tools which require chroma fusion.
4. A syntax element disclosed above may be binarized as a flag, a fixed length code, an EG(x) code, a unary code, a truncated unary code, a truncated binary code, etc. It can be signed or unsigned.
5. A syntax element disclosed above may be coded with at least one context model. Or it may be bypass coded.
6. A syntax element disclosed above may be signaled in a conditional way.
    a. The SE is signaled only if the corresponding function is applicable.
    b. The SE is signaled only if the dimensions (width and/or height) of the block satisfy a condition.
7. A syntax element disclosed above may be signaled at block level/ sequence level/group of pictures level/picture level/slice level/tile group level, such as in coding structures of CTU/CU/TU/PU/CTB/CB/TB/PB, or sequence header/picture header/SPS/VPS/DPS/DCI/PPS/APS/slice header/tile group header.

5. Embodiments

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to adaptBin(A, B).

Inputs to this process are bits from the RBSP.

Outputs of this process nomDenomPair.

AdaptBin(nomRange1, nomRange2){	
if A != B	
precision_flag	uf(1)
else	
precision_flag = 0	
if precision flag	
nominator	uf(nomRange1)
denominator = nomRange1 - 3	
else	
nominator	uf(nomRange2)
denominator = nomRange2 - 3	
nomDenomPair[0] = nominator	
nomDenomPair[1] = denominator	

When the AdaptBin(A, B) is invoked,precision_flag and value are parsed. The nomDenomPair is obtained according to the value and precision_flag as described in table above.

Syntax Table

MaskScale(){	
rvs_enable_flag	uf(1)
skipmode_enable_flag	uf(1)
lsbs_enable_flag	uf(1)
if rvs_enable_flag{	
num_rvs_ params	uf(3)
for(Idx = 0; Idx < num_rvs_ params + 1; Idx ++) {	
log2_block_size_rvs [Idx]	uf(3)
greater_flag_rvs [Idx]	uf(1)
application_flag_rvs [Idx]	uf(2)
thr_rvs [Idx]	AdaptBin(12, 9)
scale_rvs [Idx]	AdaptBin(16, 8)
}	
if skipmode_enable_flag{	
skip_mode_idx	uf(2)
if skip_mode_idx == 0 or skip_mode_idx == 2 or or skip_mode_idx == 3	
log2_block_size_skip [0]	uf(3)
thr_skip [0]	AdaptBin(16, 8)
if skip_mode_idx ==2	
log2_block_size_skip [0] = log2_block_size_skip [1]	
thr_skip [0] = thr_skip [1]	
if skip_mode_idx == 1 or skip_mode_idx == 3	
log2_block_size_skip [1]	uf(3)
thr_skip [1]	AdaptBin(16, 8)
}	
if lsbs_enable_flag{	
num_lsbs_ params	uf(3)
for(Idx = 0; Idx < num_lsbs_ params + 1; Idx ++) {	
log2_block_size_lsbs [Idx]	uf(3)
greater_flag_lsbs [Idx]	uf(1)
application_flag_lsbs [Idx]	uf(2)
thr_lsbs [Idx]	AdaptBin(12, 9)

scale1_lsbs [Idx]	AdaptBin(14, 14)
scale2_lsbs [Idx]	AdaptBin(14, 14)
}	
}	

[0134] As used herein, the term "video unit" or "video block" may be a sequence, a picture, a slice, a tile, a brick, a subpicture, a coding tree unit (CTU)/coding tree block (CTB), a CTU/CTB row, one or multiple coding units (CUs)/coding blocks (CBs), one ore multiple CTUs/CTBs, one or multiple Virtual Pipeline Data Unit (VPDU), a sub-region within a picture/slice/tile/brick.

[0135] FIG. 13 illustrates a flowchart of a method 1300 for video processing in accordance with embodiments of the present disclosure. The method 1300 is implemented during a conversion between a video unit of a video and a bitstream of the video.

[0136] At block 1310, a conversion between a video unit of a video and a bitstream of the video is perfomred according to a rule. T he rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header. In this way, it can simplify the synthesis transform module while maintaining the reconstruction capability.

[0137] In some embodiments, the conversion includes encoding the video unit into the bitstream. In some other embodiments, the conversion includes decoding the video unit from the bitstream.

[0138] In some embodiments, the first syntax element which is represented as rvs_enable_flag equal to 0 indicates disabling the RVS mode for luma and chroma components. In addition, the first syntax element which is represented as rvs_enable_flag equal to 1 indicates enabling the RVS mode for luma and chroma components.

[0139] In some embodiments, the second syntax element which is represented as skip_enable_flag equal to 0 indicates disabling skip mode for luma and chroma components. In additoin, the second syntax element which is represented as skip_enable_flag equal to 1 indicates enabling skip mode for luma and chroma components.

[0140] In some embodiments, the third syntax element which is represented as lsbs_enable_flag equal to 0 indicates disabling the LSBS mode for luma and chroma components. In additoin, the third syntax element which is represented as lsbs_enable_flag equal to 1 indicates enabling the LSBS mode for luma and chroma components.

[0141] In some embodiments, the bitstream comprises a fourth syntax element indicating the number of parameters sets for the RVS mode. Alterntaively, or in addition, the bitstream comprises a fifth syntax element indicating the number of parameters sets for the LSBS mode.

[0142] In some embodiments, the fourth syntax which is represented as num_rvs_params and is 3-biy unsigned integer indicates the number of parameters sets used in an adaptive quantization process that control a quantization of residuals. Alternatively, or in addtion, the fifth syntax element which is represented as num_lsbs_params and is 3-bit usgined integer indicates the number of parameters sets used in a latent domain masking and scaling that determine scaling at a decoder after a modified latent tensor is reconstructed. In some embodiments, the number of parameters sets of the skip mode is inferred as 1.

[0143] In some embodiments, the bitstream comprises a sixth syntax element indicating whether the RVS mode is applied to luma component or chroma component or to both luma and chroma components. In some embodiments, the sixth syntax element is represented as application_flag_rvs and is 2-bit unsigned integer. For the sixth syntax element equal to 0 indicates RVS parameter set is applied to luma component. As another example, the sixth syntax element equal to 1 indicates RVS parameter set is applied to chroma component, and the sixth syntax element equal to 2 indicates RVS parameter set is applied to both luma and chroma components. In some embodiments, if the first syntax element is equal to 0, the sixth syntax element is not parsed by a decoder.

[0144] In some embodiments, the bitstream comprises a seventh syntax element indicating whether the LSBS mode is applied to luma component or chroma component or to both luma and chroma components. In some embodiments, the seventh syntax element is represented as application_flag_lsbs and is 2-bit unsigned integer. For example, the seventh syntax element equal to 0 indicates LSBS parameter set is applied to luma component. As another example, seventh syntax element equal to 1 indicates LSBS

parameter set is applied to chroma component. By way of example, the seventh syntax element equal to 2 indicates LSBS parameter set is applied to both luma and chroma components. In some embodiments, if the third syntax element is equal to 0, the seventh syntax element is not parsed by a decoder.

[0145] In some embodiments, the bitstream comprises an eighth syntax element indicating whether the skip mode is applied to luma component or chroma component or to both luma and chroma components.

[0146] In some embodiments, the eighth syntax element is represented as skip_mode_indx and is 2-bit unsiged integer. For example, the eighth syntax element equal to 0 indicates skip mode parameter set is applied to luma component. As another example, the eighth syntax element equal to 1 indicates skip mode parameter set is applied to chroma component. By way of example, the eighth syntax element equal to 2 indicates skip mode parameter set is applied to both luma and chroma components. In some embodiments, if the fourth syntax element is equal to 0, the eighth syntax element is not parsed by a decoder.

[0147] In some embodiments, the bitstream comprises a ninth syntax element indicating scale parameters of RVS mode. Alternatively, or in addtion, the bitstream comprises one or more tenth syntax elements indicating scale parameters of LSBS mode. In some embodiments, the ninth syntax element which is represented as scale_rvs and is 16-bit or 8-bit unsigned integer indicates a value of a multiplier to be used in processing samples of the RVS mode.

[0148] In some embodiments, the one or more tenth syntax elements which are 14-bit unsigned integer indicate a value of a multiplier to be used in processing samples of the LSBS mode. In some examples, one of the one or more tenth syntax elements is represented as scale1_lsbs and the other of the one or more tenth syntax elements is represented as scale2_lsbsl. For example, if a value of the LSBS mode is less than a threshold , the scale_lsbs may be used.

[0149] In some embodiments, a precision of the scale parameters of RVS mode is signaled based on a condition or inferred. Alternatively, or in addtion, a precision of the scale parameters of LSBS mode is signaled based on a condition or inferred.

[0150] In some embodiments, the bitstream comprises an eleventh syntax element

indicating a threshold value of RVS mode. Alternatively, or in addition, the bitstream comprises a twelfth syntax element indicating a threshold value of skip mode. Alternatively, or in addition, the bitstream comprises a thirteenth syntax element indicting a threshold value of LSBS mode.

[0151] In some embodiments, the eleventh syntax element which is represented as thr_rvs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the RVS mode is used. For example, the twelfth syntax element which is represented as thr_skip and is 16-bit or 8-bit unsigned integer indicates the threshold value if the skip mode is used. As other example, the thirteenth syntax element which is represented as thr_lsbs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the LSBS mode is used.

[0152] In some embodiments, a precision of the threshold value of the RVS mode is signaled based on a condition or inferred. Alternatively, or in addition, a precision of the threshold value of the skip mode is signaled based on a condition or inferred. In some other embodiments, a precision of the threshold value of LSBL mode is signaled based on a condition or inferred.

[0153] In some embodiments, the bitstream comprises a fourteenth syntax element indicating a greater flag of the RVS mode. Alternatively or in addition, the bitstream comprises a fifteenth syntax element indicating a greater flag of the LSBS mode.

[0154] In some embodiments, the fourteenth syntax element which is represented as greater_flag_rvs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for the RVS mode. Alternatively, or in addition, the fifteenth syntax element which is represented as greater_flag_lsbs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for LSBS mode. In some embodiments, a greater flag of the skip mode is inferred to be true.

[0155] In some embodiments, the bitstream comprises a sixteenth syntax element indicating a resampling block size of the RVS mode. Alternatively, or in addition, the bitstream comprises a seventeenth syntax element indicating a resampling block size of the skip mode. Alternatiely, or in addition, the bitstream comprises an eighteenth syntax element indicating a resampling blocks size of the LSBS mode.

[0156] In some embodiments, the sixteenth syntax element which is represented as

log2_block_size_rvs and is 3-bit unsigned integer indicates a logarithm of resampling blocks size of the RVS mode. For example, the seventeen the syntax element which is represented as log2_block_size_skip is 3-bit unsigned integer indicates a logarithm of resampling block size of the skip mode. As another example, the eighteenth syntax element which is represented as log2_block_size_lsbs and is 3-bit unsigned integer indicates a logarithm of resampling block size of the LSBS mode.

[0157] In some embodiments, a mask generation is applied to at least one of: the RVS mode, the skip mode, or the LSBS mode, an input of the mask generation is a tensor with sigma samples of size $[C, h_4, w_4]$, and an output of the mask generation is a mask which is represented as $mask[C, h_4, w_4]$ and used by at least one of: the RVS mode, the skip mode, or the LSBS mode. In some embodiments, a kernel size equal to the block size in horizontal and vertical dimension is used during conducting the pooling operation the block size is equal to $2^{Log2BlockSize}$, $Log2BlockSize$ represents a logarithm of resampling block size.

[0158] In some embodiments, the method 1300 further includes: in response to the block size being greater than 1, applying a pooling operation to the tensor with sigma tensor based on a coding mode, where a kernel size equal to the block size in horizontal and vertical dimension is used during conducting the pooling operation, where the pooled sigma samples tensor is of size $[C, h_p, w_p]$, with $h_p = ceil(h_4/BlockSize), w_p = ceil(w_4/BlockSize)$; comparing each one of the pooled sigma samples with the threshold value; storing the comparison in a pooled mask tensor; obtaining pooled mask samples according to: for c = $c = 0..C-1$, $i = 0..h_p - 1$ and $j = 0..w_p - 1$ $mask_p[c,i,j] =$

$$\begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False \ Otherwise \end{cases}$$

; and obtaining a final mask samples tensor by applying an up-sampling operation to the pooled mask samples based on nearest neighbor, where the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

[0159] In some embodiments, the method 1300 further includes: in response to the block size being equal to 1, applying a pooling operation to the tensor with sigma tensor, where a size of the pooled sigma samples tensor size is equal to size of variance values tensor; comparing each one of the pooled sigma samples with the threshold value; storing the comparison in a pooled mask tensor; obtaining pooled mask samples according to: for c

$= \quad c = 0..C-1 \quad , \quad i = 0..h_p - 1 \quad \text{and} \quad j = 0..w_p - 1 \quad mask_p[c,i,j] =$

$\begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False \ Otherwise \end{cases}$; and obtaining a final mask samples tensor, where the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

[0160] In some embodiments, if the RVS mode or LSBS mode is used, an averaging pooling is used in the pooling operationif the. In some other embodiments, if the skip mode is used, a max pooling is used in the pooling operation.

[0161] In some embodiments, the RVS mode scales both residual and variance parameter used to create an entropy coding model, residual and variance scaling work together and share same scaling factors, a position of residual scaling is after Gain Unit on encoder side, a position of inverse residual scaling is right after inverse Gain Unit, variance scaling is located after Hyper Scale Decoder, and an adaptive quantization of residual samples is obtained based on their corresponding variance value, and the RVS mode uses a maximum 8 sets of control parameters. In some embodiments, an input of RVS mode is residual tensor $\hat{r}[C, h_4, w_4]$ after inverse gain unit function and variance tensor $\sigma[C, h_4, w_4]$.

[0162] In some embodiments, a process of RVS at a decoder includes the following: initializing σ_{temp} and $\hat{r}_{temp}$ tensors to be equal to variance tensor σ and quantized residual tensor $\hat{r}$, respectively; if the first syntax element is equal to 1, for $idx = 0..num\_rvs\_params - 1$ the following ordered steps are applied: if $application\_flag\_rvs[idx]$ is not equal to 0 and a current component is secondary component, or if the sixth syntax element with the $application_flag_rvs[idx]$is not equal to 1 and the current component is primary component; generating a tensor $mask[idx]$ using the mask generation with thr_rvs$[idx]$, greater_flag_rvs$[idx]$, $log2\_block\_size\_rvs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$, obtaining modified residual tensor and modified sigma tensor as follows: $\hat{r}_{temp}[c,i,j] = \begin{cases} \hat{r}_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \hat{r}_{temp}[c,i,j] & overwise \end{cases}$, $\sigma_{temp}[c,i,j] = \begin{cases} \sigma_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \sigma_{temp}[c,i,j] & overwise \end{cases}$; and determining an output

of the RVS process as the modified variance tensor $\sigma[C, h_4, w_4]$ and modified residual tensor $\hat{r}[C, h_4, w_4]$ which are set to σ_{temp} and $\hat{r}_{temp}$ respectively.

[0163] In some embodiments, a residual skip process uses maximum 2 sets of control parameters, defined by which is signalled to a decoder in Picture Header. In some embidments, at the decoder, inputs of skip mode process are 1D after an entropy decoding process of steam#2, computed using a variance tensor after a hyper scale decoding process, an output of lossless decoding process is a 1D array of which size is equal to a total number of "1"s in the tensor. In some embodiments, a *mask* tensor determines which samples of the residual tensor $\hat{r}$ are included in the bitstream and all of the other samples of quantized residual tensor are inferred to be equal to zero.

[0164] In some embodiments, the residual skip process includes the following: initializing tensors $\hat{r}$ to be equal to all zeros; setting a counter k equal to 0; setting dimensions $[C, h_4, w_4]$ equal to number of channels, height and width of the sigma tensor σ, where $C = C_p = 128$, $h_4 = h_{4,Y}, w_4 = w_{4,Y}$ for primary component, and $C = C_s = 64$, $h_4 = h_{4,UV}, w_4 = w_{4,UV}$ for secondary component); setting idx equal to 0 if current component is primary component or 0 otherwiss; if $skip\_mode\_idx$ is not equal to (1-idx); generating a tensor $mask[idx]$ using the mask generation with thr_skip[idx], $log2\_block\_size\_skip[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$, $\hat{r}[c,i,j] = \begin{cases} s_k, & mask[idx][c,i,j] == True \ OR \ skip\_enable\_flag == False \\ 0, & otherwise \end{cases}$, and $k = k + 1$; anddetermining an output of the residual skip process as residual tensor $\hat{r}$.

[0165] In some embodiments, a LSBS process uses a maximum 8 sets of control parameters, defined by which is signalled to a decoder in Picture Header. In some embodiments, at the decoder, an input of the LSBS process is a residual tensor $\hat{r}[C, h_4, w_4]$ after and entropy decoding, a prediction tensor $\mu[C, h_4, w_4]$ after a prediction fusion process, latent tensor $\hat{y}[C, h_4, w_4]$, and binary mask generated using variance σ.

[0166] In some embodiments, the LSBS process comprises: initializing tensors $\hat{y}_{temp}, \hat{r}_{temp}, \mu_{temp}$ to be equal to latent samples tensor $\hat{y}$, residual tensor $\hat{r}$ and prediction tensor μ respectively; for $idx = 0..num\_lsbs\_params - 1$, modifying $\hat{y}_{temp}$ as follows: if lsbs_enable_flag is equal to True, if $application\_flag\_lsbs[idx]$ is not equal to 0 and a current component is primary component or $application\_flag\_lsbs[idx]$ is not

equal to 1 and the current component is secondary component: generating a tensor $mask[idx]$ using the mask generation with $thr\_lsbs[idx]$, $greater\_flag\_lsbs[idx]$, $log2\_block\_size\_lsbs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$: $\hat{y}_{temp}[c,i,j] = \hat{y}_{temp}[c,i,j] +$ $(mask[idx][c,i,j] == True\ ?\ \mu[c,i,j] \cdot scale1\_lsbs[idx] + \hat{r}[c,i,j] \cdot scale2\_lsbs[idx] : 0)$; and determining an output of the LSBS process as the modified latent tensor $\hat{y}$, which is set equal to $\hat{y}_{temp}$.

[0167] In some embodiments, an indication of whether to and/or how to perform the conversion is indicated at one of the followings: sequence level, group of pictures level, picture level, slice level, or tile group level. In some other embodiments, an indication of whether to and/or how to perform the conversion is indicated in one of the following: a sequence header, a picture header, a sequence parameter set (SPS), a video parameter set (VPS), a dependency parameter set (DPS), a decoding capability information (DCI), a picture parameter set (PPS), an adaptation parameter sets (APS), a slice header, or a tile group header.

[0168] In some embodiments, the method 1300 further comprises: determining, based on coded information of the video unit, whether and/or how to perform the conversion, the coded information including at least one of: a block size, a colour format, a single and/or dual tree partitioning, a colour component, a slice type, or a picture type.

[0169] In some embodiments, the video unit is applied with a coding tool that requires chroma fusion. In some embodiments, the SE is binarized as one of a flag, a fixed length code, an EG(x) code, a unary code, a truncated unary code, or a truncated binary code.

[0170] In some embodiments, the SE is signed or unsigned. In some embodiments, the SE is coded with at least one context model. Alternatively, the SE is bypass coded.

[0171] In some embodiments, the SE is signaled in a conditional way. In some embodiments, the SE is signaled only if a corresponding function is applicable. Alternatively, the SE is signaled only if dimensions of the video unit satisfy a condition. In some embodiments, the SE is indicated at one of the followings: sequence level, group of pictures level, picture level, slice level, or tile group level.

[0172] In some embodiments, the SE is indicated at one of the followings: a prediction block (PB), a transform block (TB), a coding block (CB), a prediction unit (PU), a

transform unit (TU), a coding unit (CU), a coding tree block (CTB), or a coding tree unit (CTU).

[0173] 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. The method comprises: generating the bitstream of the video according to a rule, where the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

[0174] According to still further embodiments of the present disclosure, a method for storing bitstream of a video is provided. The method comprises: generating the bitstream of the video according to a rule, where the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header; and storing the bitstream in a non-transitory computer-readable medium.

[0175] 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.

[0176] Clause 1. A method for video processing, comprising: performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

[0177] Clause 2. The method of clause 1, wherein the first syntax element which is represented as rvs_enable_flag equal to 0 indicates disabling the RVS mode for luma and chroma components, and the first syntax element which is represented as rvs_enable_flag equal to 1 indicates enabling the RVS mode for luma and chroma components.

[0178] Clause 3. The method of clause 1 or 2, wherein the second syntax element which is represented as skip_enable_flag equal to 0 indicates disabling skip mode for luma and chroma components, and the second syntax element which is represented as skip_enable_flag equal to 1 indicates enabling skip mode for luma and chroma components.

[0179] Clause 4. The method of any of clauses 1-3, wherein the third syntax element which is represented as lsbs_enable_flag equal to 0 indicates disabling the LSBS mode for luma and chroma components, and the third syntax element which is represented as lsbs_enable_flag equal to 1 indicates enabling the LSBS mode for luma and chroma components.

[0180] Clause 5. The method of any of clauses 1-4, wherein the bitstream comprises a fourth syntax element indicating the number of parameters sets for the RVS mode; and/or wherein the bitstream comprises a fifth syntax element indicating the number of parameters sets for the LSBS mode.

[0181] Clause 6. The method of clause 5, wherein the fourth syntax which is represented as num_rvs_params and is 3-biy unsigned integer indicates the number of parameters sets used in an adaptive quantization process that control a quantization of residuals; and/or wherein the fifth syntax element which is represented as num_lsbs_params and is 3-bit usgined integer indicates the number of parameters sets used in a latent domain masking and scaling that determine scaling at a decoder after a modified latent tensor is reconstructed.

[0182] Clause 7. The method of any of clauses 1-6, wherein the number of parameters sets of the skip mode is inferred as 1.

[0183] Clause 8. The method of any of clauses 1-7, wherein the bitstream comprises a sixth syntax element indicating whether the RVS mode is applied to luma component or chroma component or to both luma and chroma components.

[0184] Clause 9. The method of clause 8, wherein the sixth syntax element is represented as application_flag_rvs and is 2-bit unsigned integer, and/or the sixth syntax element equal to 0 indicates RVS parameter set is applied to luma component, and the sixth syntax element equal to 1 indicates RVS parameter set is applied to chroma component, and the sixth syntax element equal to 2 indicates RVS parameter set is applied

to both luma and chroma components.

[0185] Clause 10. The method of clause 9, wherein if the first syntax element is equal to 0, the sixth syntax element is not parsed by a decoder.

[0186] Clause 11. The method of any of clauses 1-10, wherein the bitstream comprises a seventh syntax element indicating whether the LSBS mode is applied to luma component or chroma component or to both luma and chroma components.

[0187] Clause 12. The method of clause 11, wherein the seventh syntax element is represented as application_flag_lsbs and is 2-bit unsigned integer, and/or the seventh syntax element equal to 0 indicates LSBS parameter set is applied to luma component, and the seventh syntax element equal to 1 indicates LSBS parameter set is applied to chroma component, and the seventh syntax element equal to 2 indicates LSBS parameter set is applied to both luma and chroma components.

[0188] Clause 13. The method of clause 12, wherein if the third syntax element is equal to 0, the seventh syntax element is not parsed by a decoder.

[0189] Clause 14. The method of any of clauses 1-13, wherein the bitstream comprises an eighth syntax element indicating whether the skip mode is applied to luma component or chroma component or to both luma and chroma components.

[0190] Clause 15. The method of clause 14, wherein the eighth syntax element is represented as skip_mode_indx and is 2-bit unsiged integer, the eighth syntax element equal to 0 indicates skip mode parameter set is applied to luma component, and the eighth syntax element equal to 1 indicates skip mode parameter set is applied to chroma component, and the eighth syntax element equal to 2 indicates skip mode parameter set is applied to both luma and chroma components.

[0191] Clause 16. The method of clause 14, wherein if the fourth syntax element is equal to 0, the eighth syntax element is not parsed by a decoder.

[0192] Clause 17. The method of any of clauses 1-16, wherein the bitstream comprises a ninth syntax element indicating scale parameters of RVS mode, and/or wherein the bitstream comprises one or more tenth syntax elements indicating scale parameters of LSBS mode.

[0193] Clause 18. The method of clause 17, wherein the ninth syntax element which is

represented as scale_rvs and is 16-bit or 8-bit unsigned integer indicates a value of a multiplier to be used in processing samples of the RVS mode.

[0194] Clause 19. The method of clause 17, wherein the one or more tenth syntax elements which are 14-bit unsigned integer indicate a value of a multiplier to be used in processing samples of the LSBS mode, and/or wherein one of the one or more tenth syntax elements is represented as scale1_lsbs and the other of the one or more tenth syntax elements is represented as scale2_lsbsl.

[0195] Clause 20. The method of clause 17, wherein a precision of the scale parameters of RVS mode is signaled based on a condition or inferred, and/or wherein a precision of the scale parameters of LSBS mode is signaled based on a condition or inferred.

[0196] Clause 21. The method of any of clauses 1-20, wherein the bitstream comprises an eleventh syntax element indicating a threshold value of RVS mode, and/or wherein the bitstream comprises a twelfth syntax element indicating a threshold value of skip mode, and/or wherein the bitstream comprises a thirteenth syntax element indicting a threshold value of LSBS mode.

[0197] Clause 22. The method of clause 21, wherein the eleventh syntax element which is represented as thr_rvs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the RVS mode is used, and/or wherein the twelfth syntax element which is represented as thr_skip and is 16-bit or 8-bit unsigned integer indicates the threshold value if the skip mode is used, and/or, wherein the thirteenth syntax element which is represented as thr_lsbs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the LSBS mode is used.

[0198] Clause 23. The method of clause 21, wherein a precision of the threshold value of the RVS mode is signaled based on a condition or inferred, and/or wherein a precision of the threshold value of the skip mode is signaled based on a condition or inferred, and/or wherein a precision of the threshold value of LSBL mode is signaled based on a condition or inferred.

[0199] Clause 24. The method of any of clauses 1-23, wherein the bitstream comprises a fourteenth syntax element indicating a greater flag of the RVS mode, and/or wherein the bitstream comprises a fifteenth syntax element indicating a greater flag of the LSBS mode.

[0200] Clause 25. The method of clause 24, wherein the fourteenth syntax element

which is represented as greater_flag_rvs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for the RVS mode, and/or wherein the fifteenth syntax element which is represented as greater_flag_lsbs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for LSBS mode.

[0201] Clause 26. The method of clause 24, wherein a greater flag of the skip mode is inferred to be true.

[0202] Clause 27. The method of any of clauses 1-26, wherein the bitstream comprises a sixteenth syntax element indicating a resampling block size of the RVS mode, and/or wherein the bitstream comprises a seventeenth syntax element indicating a resampling block size of the skip mode, and/or wherein the bitstream comprises an eighteenth syntax element indicating a resampling blocks size of the LSBS mode.

[0203] Clause 28. The method of clause 27, wherein the sixteenth syntax element which is represented as log2_block_size_rvs and is 3-bit unsigned integer indicates a logarithm of resampling blocks size of the RVS mode, and/or wherein the seventeen the syntax element which is represented as log2_block_size_skip is 3-bit unsigned integer indicates a logarithm of resampling block size of the skip mode, and/or wherein the eighteenth syntax element which is represented as log2_block_size_lsbs and is 3-bit unsigned integer indicates a logarithm of resampling block size of the LSBS mode.

[0204] Clause 29. The method of any of clauses 1-28, wherein a mask generation is applied to at least one of: the RVS mode, the skip mode, or the LSBS mode, an input of the mask generation is a tensor with sigma samples of size $[C, h_4, w_4]$, and an output of the mask generation is a mask which is represented as $mask[C, h_4, w_4]$ and used by at least one of: the RVS mode, the skip mode, or the LSBS mode.

[0205] Clause 30. The method of clause 29, wherein the mask generation is based on a threshold value, a greater flag, and a block size, and wherein the block size is equal to $2^{Log2BlockSize}$, $Log2BlockSize$ represents a logarithm of resampling block size.

[0206] Clause 31. The method of clause 30, further comprising: in response to the block size being greater than 1, applying a pooling operation to the tensor with sigma tensor based on a coding mode, wherein a kernel size equal to the block size in horizontal and vertical dimension is used during conducting the pooling operation, wherein the pooled

sigma samples tensor is of size $[C, h_p, w_p]$, with $h_p = ceil(h_4/BlockSize), w_p = ceil(w_4/BlockSize)$; comparing each one of the pooled sigma samples with the threshold value; storing the comparison in a pooled mask tensor; obtaining pooled mask samples according to: for c = $c = 0..C-1$, $i = 0..h_p - 1$ and $j = 0..w_p - 1$ $mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False\ Otherwise \end{cases}$; and obtaining a final mask samples tensor by applying an up-sampling operation to the pooled mask samples based on nearest neighbor, wherein the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

[0207] Clause 32. The method of clause 30, further comprising: in response to the block size being equal to 1, applying a pooling operation to the tensor with sigma tensor, wherein a size of the pooled sigma samples tensor size is equal to size of variance values tensor; comparing each one of the pooled sigma samples with the threshold value; storing the comparison in a pooled mask tensor; obtaining pooled mask samples according to: for c = $c = 0..C-1$, $i = 0..h_p - 1$ and $j = 0..w_p - 1$ $mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False\ Otherwise \end{cases}$; and obtaining a final mask samples tensor, wherein the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

[0208] Clause 33. The method of clause 31 or 32, wherein if the RVS mode or LSBS mode is used, an averaging pooling is used in the pooling operation, and if the skip mode is used, a max pooling is used in the pooling operation.

[0209] Clause 34. The method of any of clauses 1-33, wherein the RVS mode scales both residual and variance parameter used to create an entropy coding model, residual and variance scaling work together and share same scaling factors, a position of residual scaling is after Gain Unit on encoder side, a position of inverse residual scaling is right after inverse Gain Unit, variance scaling is located after Hyper Scale Decoder, and an adaptive quantization of residual samples is obtained based on their corresponding variance value, and the RVS mode uses a maximum 8 sets of control parameters.

[0210] Clause 35. The method of clause 34, wherein an input of RVS mode is residual tensor $\hat{r}[C, h_4, w_4]$ after inverse gain unit function and variance tensor $\sigma[C, h_4, w_4]$.

[0211] Clause 36. The method of clause 34, wherein a process of RVS at a decoder comprises the following: initializing σ_{temp} and $\hat{r}_{temp}$ tensors to be equal to variance tensor σ and quantized residual tensor $\hat{r}$, respectively; if the first syntax element is equal to 1, for $idx = 0..num\_rvs\_params - 1$ the following ordered steps are applied: if $application\_flag\_rvs[idx]$ is not equal to 0 and a current component is secondary component, or if the sixth syntax element with the $application_flag_rvs[idx]$is not equal to 1 and the current component is primary component; generating a tensor $mask[idx]$ using the mask generation with thr_rvs$[idx]$, greater_flag_rvs$[idx]$, $log2\_block\_size\_rvs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C - 1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$, obtaining modified residual tensor and modified sigma tensor as follows: $\hat{r}_{temp}[c, i, j] = \begin{cases} \hat{r}_{temp}[c, i, j] \cdot scale\_rvs[idx], & if\ mask[idx][c, i, j]\ is\ True, \\ \hat{r}_{temp}[c, i, j] & overwise \end{cases}$ $\sigma_{temp}[c, i, j] = \begin{cases} \sigma_{temp}[c, i, j] \cdot scale\_rvs[idx], & if\ mask[idx][c, i, j]\ is\ True, \\ \sigma_{temp}[c, i, j] & overwise \end{cases}$; and determining an output of the RVS process as the modified variance tensor $\sigma[C, h_4, w_4]$ and modified residual tensor $\hat{r}[C, h_4, w_4]$ which are set to σ_{temp} and $\hat{r}_{temp}$ respectively.

[0212] Clause 37. The method of any of clauses 1-36, wherein a residual skip process uses maximum 2 sets of control parameters, defined by $skip\_mode\_idx$ which is signalled to a decoder in Picture Header, at the decoder, inputs of skip mode process are 1D $\{s'_m\}$ after an entropy decoding process of steam#2, $mask$ computed using a variance tensor σ after a hyper scale decoding process, an output of lossless decoding process is a 1D array $\{s'_m\}$ of which size is equal to a total number of "1"s in the $mask$ tensor.

[0213] Clause 38. The method of clause 37, wherein a $mask$ tensor determines which samples of the residual tensor $\hat{r}$ are included in the bitstream and all of the other samples of quantized residual tensor are inferred to be equal to zero.

[0214] Clause 39. The method of clause 37, wherein the residual skip process comprises the following: initializing tensors $\hat{r}$ to be equal to all zeros; setting a counter k equal to 0; setting dimensions $[C, h_4, w_4]$ equal to number of channels, height and width of the sigma tensor σ, wherein $C = C_p = 128$, $h_4 = h_{4,Y}, w_4 = w_{4,Y}$ for primary component, and

$C = C_s = 64$, $h_4 = h_{4,UV}, w_4 = w_{4,UV}$ for secondary component); setting idx equal to 0 if current component is primary component or 0 otherwiss; if $skip\_mode\_idx$ is not equal to (1- idx); generating a tensor $mask[idx]$ using the mask generation with thr_skip$[idx]$, $log2\_block\_size\_skip[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$, $\hat{r}[c,i,j] = \begin{cases} s_k, & mask[idx][c,i,j] == True \ OR \ skip\_enable\_flag == False \\ 0, & otherwise \end{cases}$, and $k = k + 1$; and determining an output of the residual skip process as residual tensor $\hat{r}$.

[0215] Clause 40. The method of any of clauses 1-39, wherein a LSBS process uses a maximum 8 sets of control parameters, defined by num_lsbs_params which is signalled to a decoder in Picture Header, and at the decoder, an input of the LSBS process is a residual tensor $\hat{r}[C, h_4, w_4]$ after and entropy decoding, a prediction tensor $\mu[C, h_4, w_4]$ after a prediction fusion process, latent tensor $\hat{y}[C, h_4, w_4]$, and binary mask generated using variance σ.

[0216] Clause 41. The method of clause 40, wherein the LSBS process comprises: initializing tensors $\hat{y}_{temp}, \hat{r}_{temp}, \mu_{temp}$ to be equal to latent samples tensor $\hat{y}$, residual tensor $\hat{r}$ and prediction tensor μ respectively; for $idx = 0..num\_lsbs\_params - 1$, modifying $\hat{y}_{temp}$ as follows: if lsbs_enable_flag is equal to True, if $application\_flag\_lsbs[idx]$ is not equal to 0 and a current component is primary component or $application\_flag\_lsbs[idx]$ is not equal to 1 and the current component is secondary component: generating a tensor $mask[idx]$ using the mask generation with $thr\_lsbs[idx]$, $greater\_flag\_lsbs[idx]$, $log2\_block\_size\_lsbs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output; for $c = 0..C-1$, $i = 0..h_4 - 1$, $j = 0..w_4 - 1$: $\hat{y}_{temp}[c,i,j] = \hat{y}_{temp}[c,i,j] + (mask[idx][c,i,j] == True \ ? \ \mu[c,i,j] \cdot scale1\_lsbs[idx] + \hat{r}[c,i,j] \cdot scale2\_lsbs[idx] : 0)$; and determining an output of the LSBS process as the modified latent tensor $\hat{y}$, which is set equal to $\hat{y}_{temp}$.

[0217] Clause 42. The method of any of clauses 1-41, wherein an indication of whether to and/or how to perform the conversion is indicated at one of the followings: sequence level, group of pictures level, picture level, slice level, or tile group level.

[0218] Clause 43. The method of any of clauses 1-41, wherein an indication of whether to and/or how to perform the conversion is indicated in one of the following: a sequence header, a picture header, a sequence parameter set (SPS), a video parameter set (VPS), a

dependency parameter set (DPS), a decoding capability information (DCI), a picture parameter set (PPS), an adaptation parameter sets (APS), a slice header, or a tile group header.

[0219] Clause 44. The method of any of clauses 1-41, further comprising: determining, based on coded information of the video unit, whether and/or how to perform the conversion, the coded information including at least one of: a block size, a colour format, a single and/or dual tree partitioning, a colour component, a slice type, or a picture type.

[0220] Clause 45. The method of any of clauses 1-44, wherein the video unit is applied with a coding tool that requires chroma fusion.

[0221] Clause 46. The method of any of clauses 1-45, wherein the SE is binarized as one of a flag, a fixed length code, an EG(x) code, a unary code, a truncated unary code, or a truncated binary code.

[0222] Clause 47. The method of clause 46, wherein the SE is signed or unsigned.

[0223] Clause 48. The method of any of clauses 1-47, wherein the SE is coded with at least one context model, or wherein the SE is bypass coded.

[0224] Clause 49. The method of any of clauses 1-48, wherein the SE is signaled in a conditional way.

[0225] Clause 50. The method of clause 49, wherein the SE is signaled only if a corresponding function is applicable, or wherein the SE is signaled only if dimensions of the video unit satisfy a condition.

[0226] Clause 51. The method of any of clauses 1-50, wherein the SE is indicated at one of the followings: sequence level, group of pictures level, picture level, slice level, or tile group level.

[0227] Clause 52. The method of any of clauses 1-50, wherein the SE is indicated at one of the followings: a prediction block (PB), a transform block (TB), a coding block (CB), a prediction unit (PU), a transform unit (TU), a coding unit (CU), a coding tree block (CTB), or a coding tree unit (CTU).

[0228] Clause 53. The method of any of clauses 1-52, wherein the conversion includes encoding the video unit into the bitstream.

[0229] Clause 54. The method of any of clauses 1-52, wherein the conversion includes

decoding the video unit from the bitstream.

[0230] Clause 55. 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 1-54.

[0231] Clause 56. A non-transitory computer-readable storage medium storing instructions that cause a processor to perform a method in accordance with any of clauses 1-54.

[0232] Clause 57. 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 of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

[0233] Clause 58. A method for storing a bitstream of a video, comprising: generating the bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header; and storing the bitstream in a non-transitory computer-readable medium.

Example Device

[0234] FIG. 14 illustrates a block diagram of a computing device 1400 in which various embodiments of the present disclosure can be implemented. The computing device 1400 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).

[0235] It would be appreciated that the computing device 1400 shown in FIG. 14 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.

[0236] As shown in FIG. 14, the computing device 1400 includes a general-purpose computing device 1400. The computing device 1400 may at least comprise one or more processors or processing units 1410, a memory 1420, a storage unit 1430, one or more communication units 1440, one or more input devices 1450, and one or more output devices 1460.

[0237] In some embodiments, the computing device 1400 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 1400 can support any type of interface to a user (such as "wearable" circuitry and the like).

[0238] The processing unit 1410 may be a physical or virtual processor and can implement various processes based on programs stored in the memory 1420. 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 1400. The processing unit 1410 may also be referred to as a central processing unit (CPU), a microprocessor, a controller or a microcontroller.

[0239] The computing device 1400 typically includes various computer storage medium. Such medium can be any medium accessible by the computing device 1400, including, but not limited to, volatile and non-volatile medium, or detachable and non-detachable medium. The memory 1420 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 1430 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 1400.

[0240] The computing device 1400 may further include additional detachable/non-detachable, volatile/non-volatile memory medium. Although not shown in FIG. 14, 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.

[0241] The communication unit 1440 communicates with a further computing device via the communication medium. In addition, the functions of the components in the computing device 1400 can be implemented by a single computing cluster or multiple computing machines that can communicate via communication connections. Therefore, the computing device 1400 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.

[0242] The input device 1450 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 1460 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 1440, the computing device 1400 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 1400, or any devices (such as a network card, a modem and the like) enabling the computing device 1400 to communicate with one or more other computing devices, if required. Such communication can be performed via input/output (I/O) interfaces (not shown).

[0243] In some embodiments, instead of being integrated in a single device, some or all components of the computing device 1400 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.

[0244] The computing device 1400 may be used to implement video encoding/decoding in embodiments of the present disclosure. The memory 1420 may include one or more video coding modules 1425 having one or more program instructions. These modules are accessible and executable by the processing unit 1410 to perform the functionalities of the various embodiments described herein.

[0245] In the example embodiments of performing video encoding, the input device 1450 may receive video data as an input 1470 to be encoded. The video data may be processed, for example, by the video coding module 1425, to generate an encoded bitstream. The encoded bitstream may be provided via the output device 1460 as an output 1480.

[0246] In the example embodiments of performing video decoding, the input device 1450 may receive an encoded bitstream as the input 1470. The encoded bitstream may be processed, for example, by the video coding module 1425, to generate decoded video data. The decoded video data may be provided via the output device 1460 as the output 1480.

[0247] 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 video unit of a video and a bitstream of the video according to a rule,

wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

2. The method of claim 1, wherein the first syntax element which is represented as rvs_enable_flag equal to 0 indicates disabling the RVS mode for luma and chroma components, and the first syntax element which is represented as rvs_enable_flag equal to 1 indicates enabling the RVS mode for luma and chroma components.

3. The method of claim 1 or 2, wherein the second syntax element which is represented as skip_enable_flag equal to 0 indicates disabling skip mode for luma and chroma components, and the second syntax element which is represented as skip_enable_flag equal to 1 indicates enabling skip mode for luma and chroma components.

4. The method of any of claims 1-3, wherein the third syntax element which is represented as lsbs_enable_flag equal to 0 indicates disabling the LSBS mode for luma and chroma components, and the third syntax element which is represented as lsbs_enable_flag equal to 1 indicates enabling the LSBS mode for luma and chroma components.

5. The method of any of claims 1-4, wherein the bitstream comprises a fourth syntax element indicating the number of parameters sets for the RVS mode; and/or

wherein the bitstream comprises a fifth syntax element indicating the number of parameters sets for the LSBS mode.

6. The method of claim 5, wherein the fourth syntax which is represented as num_rvs_params and is 3-biy unsigned integer indicates the number of parameters sets used in an adaptive quantization process that control a quantization of residuals; and/or

wherein the fifth syntax element which is represented as num_lsbs_params and is 3-bit usgined integer indicates the number of parameters sets used in a latent domain masking and scaling that determine scaling at a decoder after a modified latent tensor is reconstructed.

7. The method of any of claims 1-6, wherein the number of parameters sets of the skip mode is inferred as 1.

8. The method of any of claims 1-7, wherein the bitstream comprises a sixth syntax element indicating whether the RVS mode is applied to luma component or chroma component or to both luma and chroma components.

9. The method of claim 8, wherein the sixth syntax element is represented as application_flag_rvs and is 2-bit unsigned integer, and/or

the sixth syntax element equal to 0 indicates RVS parameter set is applied to luma component, and

the sixth syntax element equal to 1 indicates RVS parameter set is applied to chroma component, and

the sixth syntax element equal to 2 indicates RVS parameter set is applied to both luma and chroma components.

10. The method of claim 9, wherein if the first syntax element is equal to 0, the sixth syntax element is not parsed by a decoder.

11. The method of any of claims 1-10, wherein the bitstream comprises a seventh syntax element indicating whether the LSBS mode is applied to luma component or chroma component or to both luma and chroma components.

12. The method of claim 11, wherein the seventh syntax element is represented as application_flag_lsbs and is 2-bit unsigned integer, and/or

the seventh syntax element equal to 0 indicates LSBS parameter set is applied to luma component, and

the seventh syntax element equal to 1 indicates LSBS parameter set is applied to chroma component, and

the seventh syntax element equal to 2 indicates LSBS parameter set is applied to both luma and chroma components.

13. The method of claim 12, wherein if the third syntax element is equal to 0, the seventh syntax element is not parsed by a decoder.

14. The method of any of claims 1-13, wherein the bitstream comprises an eighth syntax element indicating whether the skip mode is applied to luma component or chroma component or to both luma and chroma components.

15. The method of claim 14, wherein the eighth syntax element is represented as skip_mode_indx and is 2-bit unsiged integer, and/or

the eighth syntax element equal to 0 indicates skip mode parameter set is applied to luma component, and

the eighth syntax element equal to 1 indicates skip mode parameter set is applied to chroma component, and

the eighth syntax element equal to 2 indicates skip mode parameter set is applied to both luma and chroma components.

16. The method of claim 14, wherein if the fourth syntax element is equal to 0, the eighth syntax element is not parsed by a decoder.

17. The method of any of claims 1-16, wherein the bitstream comprises a ninth syntax element indicating scale parameters of RVS mode, and/or

wherein the bitstream comprises one or more tenth syntax elements indicating scale parameters of LSBS mode.

18. The method of claim 17, wherein the ninth syntax element which is represented as scale_rvs and is 16-bit or 8-bit unsigned integer indicates a value of a multiplier to be used in processing samples of the RVS mode.

19. The method of claim 17, wherein the one or more tenth syntax elements which are 14-bit unsigned integer indicate a value of a multiplier to be used in processing samples of the LSBS mode, and/or

wherein one of the one or more tenth syntax elements is represented as scale1_lsbs and the other of the one or more tenth syntax elements is represented as scale2_lsbsl.

20. The method of claim 17, wherein a precision of the scale parameters of RVS mode is signaled based on a condition or inferred, and/or

wherein a precision of the scale parameters of LSBS mode is signaled based on a condition or inferred.

21. The method of any of claims 1-20, wherein the bitstream comprises an eleventh syntax element indicating a threshold value of RVS mode, and/or

wherein the bitstream comprises a twelfth syntax element indicating a threshold value of skip mode, and/or

wherein the bitstream comprises a thirteenth syntax element indicting a threshold value of LSBS mode.

22. The method of claim 21, wherein the eleventh syntax element which is represented as thr_rvs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the RVS mode is used, and/or

wherein the twelfth syntax element which is represented as thr_skip and is 16-bit or 8-bit unsigned integer indicates the threshold value if the skip mode is used, and/or,

wherein the thirteenth syntax element which is represented as thr_lsbs and is 12-bit or 9-bit unsigned integer indicates the threshold value if the LSBS mode is used.

23. The method of claim 21, wherein a precision of the threshold value of the RVS mode is signaled based on a condition or inferred, and/or

wherein a precision of the threshold value of the skip mode is signaled based on a condition or inferred, and/or

wherein a precision of the threshold value of LSBL mode is signaled based on a condition or inferred.

24. The method of any of claims 1-23, wherein the bitstream comprises a fourteenth syntax element indicating a greater flag of the RVS mode, and/or

wherein the bitstream comprises a fifteenth syntax element indicating a greater flag of the LSBS mode.

25. The method of claim 24, wherein the fourteenth syntax element which is represented as greater_flag_rvs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for the RVS mode, and/or

wherein the fifteenth syntax element which is represented as greater_flag_lsbs and is 1-bit binary value indicates whether a thresholding operation is to be applied as greater than or smaller than a threshold for LSBS mode.

26. The method of claim 24, wherein a greater flag of the skip mode is inferred to be true.

27. The method of any of claims 1-26, wherein the bitstream comprises a sixteenth syntax element indicating a resampling block size of the RVS mode, and/or

wherein the bitstream comprises a seventeenth syntax element indicating a resampling block size of the skip mode, and/or

wherein the bitstream comprises an eighteenth syntax element indicating a resampling blocks size of the LSBS mode.

28. The method of claim 27, wherein the sixteenth syntax element which is represented as log2_block_size_rvs and is 3-bit unsigned integer indicates a logarithm of resampling blocks size of the RVS mode, and/or

wherein the seventeen the syntax element which is represented as log2_block_size_skip is 3-bit unsigned integer indicates a logarithm of resampling block size of the skip mode, and/or

wherein the eighteenth syntax element which is represented as log2_block_size_lsbs and is 3-bit unsigned integer indicates a logarithm of resampling block size of the LSBS mode.

29. The method of any of claims 1-28, wherein a mask generation is applied to at least one of: the RVS mode, the skip mode, or the LSBS mode, an input of the mask generation is a tensor with sigma samples of size $[C, h_4, w_4]$, and an output of the mask generation is a mask which is represented as $mask[C, h_4, w_4]$ and used by at least one of: the RVS mode, the skip mode, or the LSBS mode.

30. The method of claim 29, wherein the mask generation is based on a threshold value, a greater flag, and a block size, and

wherein the block size is equal to $2^{Log2BlockSize}$, $Log2BlockSize$ represents a logarithm of resampling block size.

31. The method of claim 30, further comprising:

in response to the block size being greater than 1, applying a pooling operation to the tensor with sigma tensor based on a coding mode, wherein a kernel size equal to the block size in horizontal and vertical dimension is used during conducting the pooling operation, wherein the pooled sigma samples tensor is of size $[C, h_p, w_p]$, with $h_p = ceil(h_4/BlockSize), w_p = ceil(w_4/BlockSize)$;

comparing each one of the pooled sigma samples with the threshold value;

storing the comparison in a pooled mask tensor;

obtaining pooled mask samples according to: for c = $c = 0..C-1$, $i = 0..h_p - 1$ and $j = 0..w_p - 1$

$$mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False \ Otherwise \end{cases}; \text{ and}$$

obtaining a final mask samples tensor by applying an up-sampling operation to the pooled mask samples based on nearest neighbor, wherein the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

32. The method of claim 30, further comprising:

in response to the block size being equal to 1, applying a pooling operation to the tensor with sigma tensor, wherein a size of the pooled sigma samples tensor size is equal to size of variance values tensor;

comparing each one of the pooled sigma samples with the threshold value;

storing the comparison in a pooled mask tensor;

obtaining pooled mask samples according to: for c = $c = 0..C-1$, $i = 0..h_p - 1$ and $j = 0..w_p - 1$

$$mask_p[c,i,j] = \begin{cases} True \text{ if } \sigma_p[c,i,j] > Threshold \cap GreaterFlag == True \\ True \text{ if } \sigma_p[c,i,j] < Threshold \cap GreaterFlag == False \\ False \ Otherwise \end{cases}; \text{ and}$$

obtaining a final mask samples tensor, wherein the final mask samples tensor is represented as $mask[c,i,j] = mask_p[c, i/BlockSize\ , j/BlockSize\ ]$, $c = 0..C-1$, $i = 0..h_4 - 1$ and $j = 0..w_4 - 1$.

33. The method of claim 31 or 32, wherein if the RVS mode or LSBS mode is used, an averaging pooling is used in the pooling operation, and

if the skip mode is used, a max pooling is used in the pooling operation.

34. The method of any of claims 1-33, wherein the RVS mode scales both residual and variance parameter used to create an entropy coding model, residual and variance scaling work together and share same scaling factors, a position of residual scaling is after Gain Unit on encoder side, a position of inverse residual scaling is right after inverse Gain Unit, variance scaling is located after Hyper Scale Decoder, and an adaptive quantization of residual samples is obtained based on their corresponding variance value, and the RVS mode uses a maximum 8 sets of control parameters.

35. The method of claim 34, wherein an input of RVS mode is residual tensor $\hat{r}[C, h_4, w_4]$ after inverse gain unit function and variance tensor $\sigma[C, h_4, w_4]$.

36. The method of claim 34, wherein a process of RVS at a decoder comprises the following:

initializing σ_{temp} and $\hat{r}_{temp}$ tensors to be equal to variance tensor σ and quantized residual tensor $\hat{r}$, respectively;

if the first syntax element is equal to 1, for $idx = 0..num\_rvs\_params - 1$ the following ordered steps are applied:

if $application\_flag\_rvs[idx]$ is not equal to 0 and a current component is secondary component, or if the sixth syntax element with the $application_flag_rvs[idx]$is not equal to 1 and the current component is primary component;

generating a tensor $mask[idx]$ using the mask generation with thr_rvs[idx], greater_flag_rvs[idx], $log2\_block\_size\_rvs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output;

for $c = 0..C-1$, $i = 0..h_4-1$, $j = 0..w_4-1$, obtaining modified residual tensor and modified sigma tensor as follows:

$$\hat{r}_{temp}[c,i,j] = \begin{cases} \hat{r}_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \hat{r}_{temp}[c,i,j] & overwise \end{cases}$$

$$\sigma_{temp}[c,i,j] = \begin{cases} \sigma_{temp}[c,i,j] \cdot scale\_rvs[idx], & if\ mask[idx][c,i,j]\ is\ True, \\ \sigma_{temp}[c,i,j] & overwise \end{cases}$$

; and

determining an output of the RVS process as the modified variance tensor $\sigma[C, h_4, w_4]$ and modified residual tensor $\hat{r}[C, h_4, w_4]$ which are set to σ_{temp} and $\hat{r}_{temp}$ respectively.

37. The method of any of claims 1-36, wherein a residual skip process uses maximum 2 sets of control parameters, defined by $skip\_mode\_idx$ which is signalled to a decoder in Picture Header,

wherein at the decoder, inputs of skip mode process are 1D $\{s'_m\}$ after an entropy decoding process of steam#2, $mask$ computed using a variance tensor σ after a hyper scale decoding process, an output of lossless decoding process is a 1D array $\{s'_m\}$ of which size is equal to a total number of "1"s in the $mask$ tensor.

38. The method of claim 37, wherein a $mask$ tensor determines which samples of the residual tensor $\hat{r}$ are included in the bitstream and all of the other samples of quantized residual tensor are inferred to be equal to zero.

39. The method of claim 37, wherein the residual skip process comprises the following:

initializing tensors $\hat{r}$ to be equal to all zeros;

setting a counter k equal to 0;

setting dimensions $[C, h_4, w_4]$ equal to number of channels, height and width of the sigma tensor σ, wherein $C = C_p = 128$, $h_4 = h_{4,Y}, w_4 = w_{4,Y}$ for primary component, and $C = C_s = 64$, $h_4 = h_{4,UV}, w_4 = w_{4,UV}$ for secondary component);

setting idx equal to 0 if current component is primary component or 0 otherwiss;

if $skip\_mode\_idx$ is not equal to (1- idx);

generating a tensor $mask[idx]$ using the mask generation with thr_skip$[idx]$, $log2\_block\_size\_skip[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output;

for $c = 0..C-1$, $i = 0..h_4-1$, $j = 0..w_4-1$,
$\hat{r}[c,i,j] = \begin{cases} s_k, & mask[idx][c,i,j] == True \ OR \ skip\_enable\_flag == False \\ 0, & otherwise \end{cases}$,
and $k = k+1$; and

determining an output of the residual skip process as residual tensor $\hat{r}$.

40. The method of any of claims 1-39, wherein a LSBS process uses a maximum 8 sets of control parameters, defined by num_lsbs_params which is signalled to a decoder in Picture Header, and

wherein at the decoder, an input of the LSBS process is a residual tensor $\hat{r}[C,h_4,w_4]$ after and entropy decoding, a prediction tensor $\mu[C,h_4,w_4]$ after a prediction fusion process, latent tensor $\hat{y}[C,h_4,w_4]$, and binary mask generated using variance σ.

41. The method of claim 40, wherein the LSBS process comprises:

initializing tensors $\hat{y}_{temp}, \hat{r}_{temp}, \mu_{temp}$ to be equal to latent samples tensor $\hat{y}$, residual tensor $\hat{r}$ and prediction tensor μ respectively;

for $idx = 0..num\_lsbs\_params - 1$, modifying $\hat{y}_{temp}$ as follows:

if lsbs_enable_flag is equal to True,

if $application\_flag\_lsbs[idx]$ is not equal to 0 and a current component is primary component or $application\_flag\_lsbs[idx]$ is not equal to 1 and the current component is secondary component:

generating a tensor $mask[idx]$ using the mask generation with $thr\_lsbs[idx]$, $greater\_flag\_lsbs[idx]$, $log2\_block\_size\_lsbs[idx]$ and sigma samples tensor as inputs and $mask[idx]$ as output;

for $c = 0..C-1$, $i = 0..h_4-1$, $j = 0..w_4-1$:

$\hat{y}_{temp}[c,i,j] = \hat{y}_{temp}[c,i,j] + (mask[idx][c,i,j] == True \ ? \ \mu[c,i,j] \cdot scale1\_lsbs[idx] + \hat{r}[c,i,j] \cdot scale2\_lsbs[idx] : 0)$; and

determining an output of the LSBS process as the modified latent tensor $\hat{y}$, which is set equal to $\hat{y}_{temp}$.

42. The method of any of claims 1-41, wherein an indication of whether to and/or how to perform the conversion is indicated at one of the followings:

sequence level,

group of pictures level,

picture level,

slice level, or

tile group level.

43. The method of any of claims 1-41, wherein an indication of whether to and/or how to perform the conversion is indicated in one of the following:

a sequence header,

a picture header,

a sequence parameter set (SPS),

a video parameter set (VPS),

a dependency parameter set (DPS),

a decoding capability information (DCI),

a picture parameter set (PPS),

an adaptation parameter sets (APS),

a slice header, or

a tile group header.

44. The method of any of claims 1-41, further comprising:

determining, based on coded information of the video unit, whether and/or how to perform the conversion, the coded information including at least one of:

a block size,

a colour format,

a single and/or dual tree partitioning,

a colour component,

a slice type, or

a picture type.

45. The method of any of claims 1-44, wherein the video unit is applied with a coding tool that requires chroma fusion.

46. The method of any of claims 1-45, wherein the SE is binarized as one of a flag, a fixed length code, an EG(x) code, a unary code, a truncated unary code, or a truncated binary code.

47. The method of claim 46, wherein the SE is signed or unsigned.

48. The method of any of claims 1-47, wherein the SE is coded with at least one context model, or

wherein the SE is bypass coded.

49. The method of any of claims 1-48, wherein the SE is signaled in a conditional way.

50. The method of claim 49, wherein the SE is signaled only if a corresponding function is applicable, or

wherein the SE is signaled only if dimensions of the video unit satisfy a condition.

51. The method of any of claims 1-50, wherein the SE is indicated at one of the followings:

sequence level,

group of pictures level,

picture level,

slice level, or

tile group level.

52. The method of any of claims 1-50, wherein the SE is indicated at one of the followings:

a prediction block (PB),

a transform block (TB),

a coding block (CB),

a prediction unit (PU),

a transform unit (TU),

a coding unit (CU),

a coding tree block (CTB), or

a coding tree unit (CTU).

53. The method of any of claims 1-52, wherein the conversion includes encoding the video unit into the bitstream.

54. The method of any of claims 1-52, wherein the conversion includes decoding the video unit from the bitstream.

55. 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 of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header.

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

generating the bitstream of the video according to a rule, wherein the rule indicates that a first syntax element indicating whether a residual and variance scale (RVS) mode being enabled or not, a second syntax element indicating whether a skip mode being enabled or not, and a third syntax element indicating whether a latent scale before synthesis (LSBS) mode being enabled or not are included in a picture header; and

storing the bitstream in a non-transitory computer-readable medium.

DRAWINGS

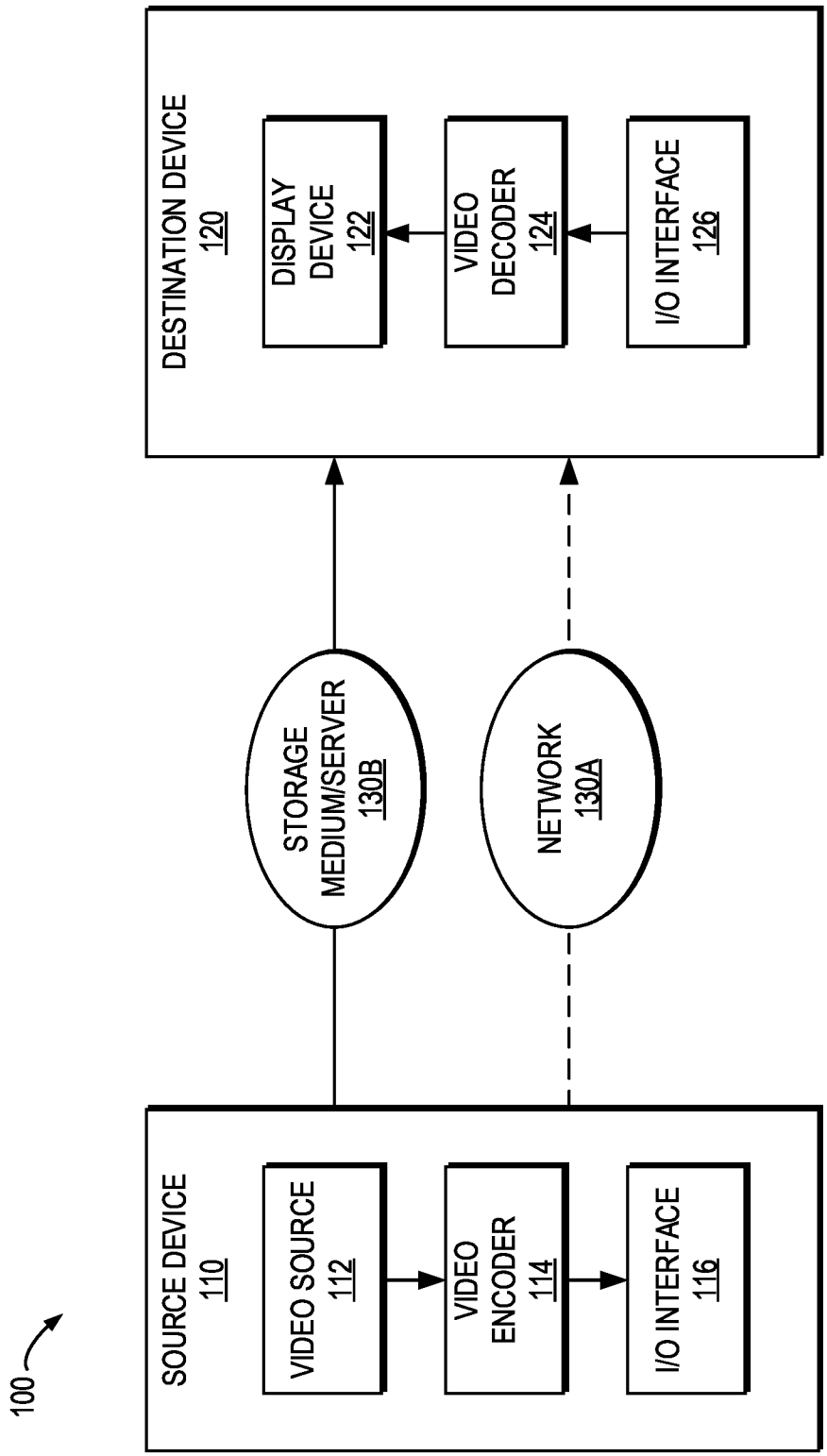


FIG. 1

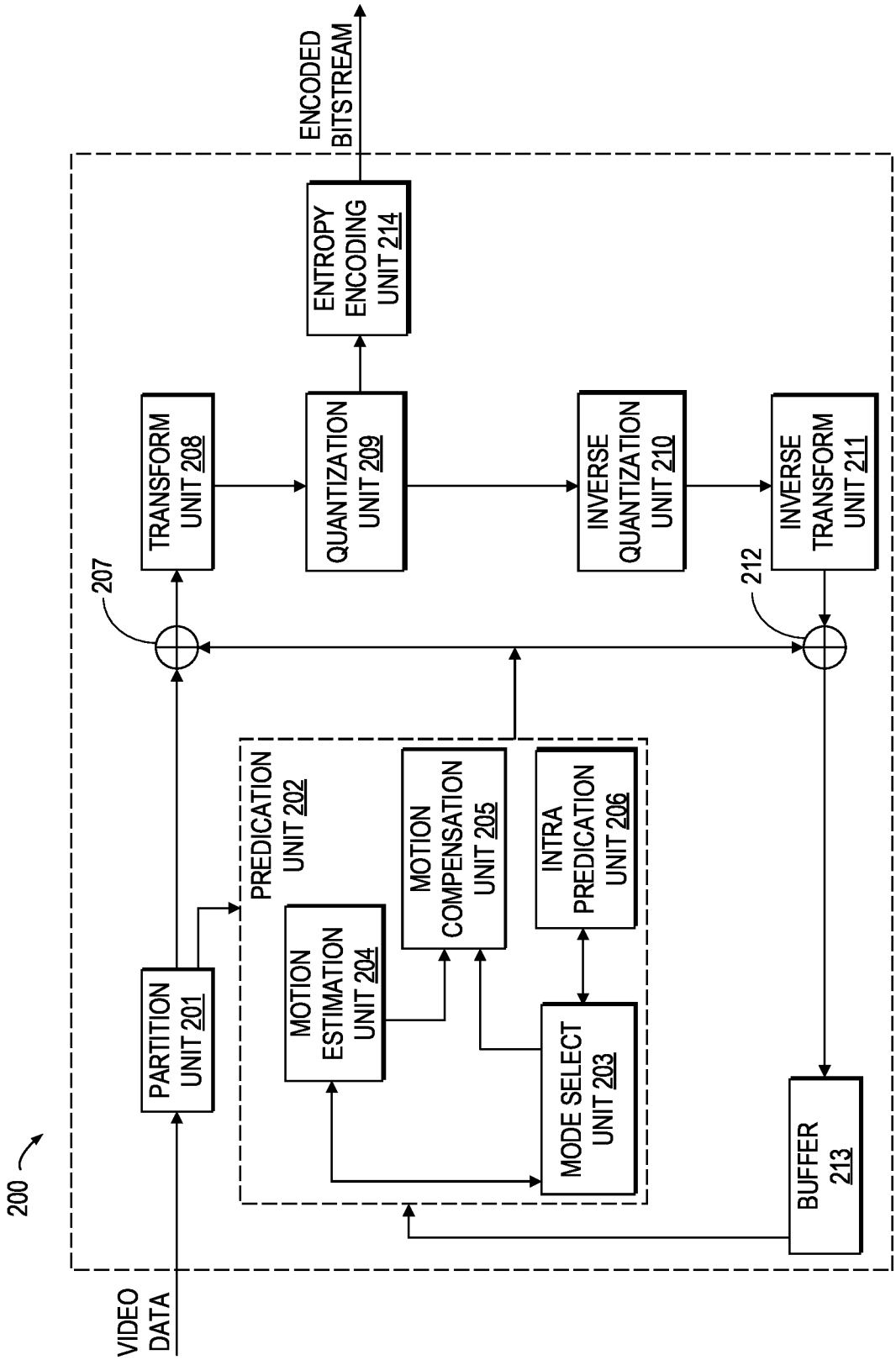

200
VIDEO DATA
PARTITION UNIT 201
PREDICATION UNIT 202
MOTION ESTIMATION UNIT 204
MOTION COMPENSATION UNIT 205
INTRA PREDICATION UNIT 206
MODE SELECT UNIT 203
207
TRANSFORM UNIT 208
QUANTIZATION UNIT 209
ENTROPY ENCODING UNIT 214
ENCODED BITSTREAM
INVERSE QUANTIZATION UNIT 210
INVERSE TRANSFORM UNIT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

FIG. 2

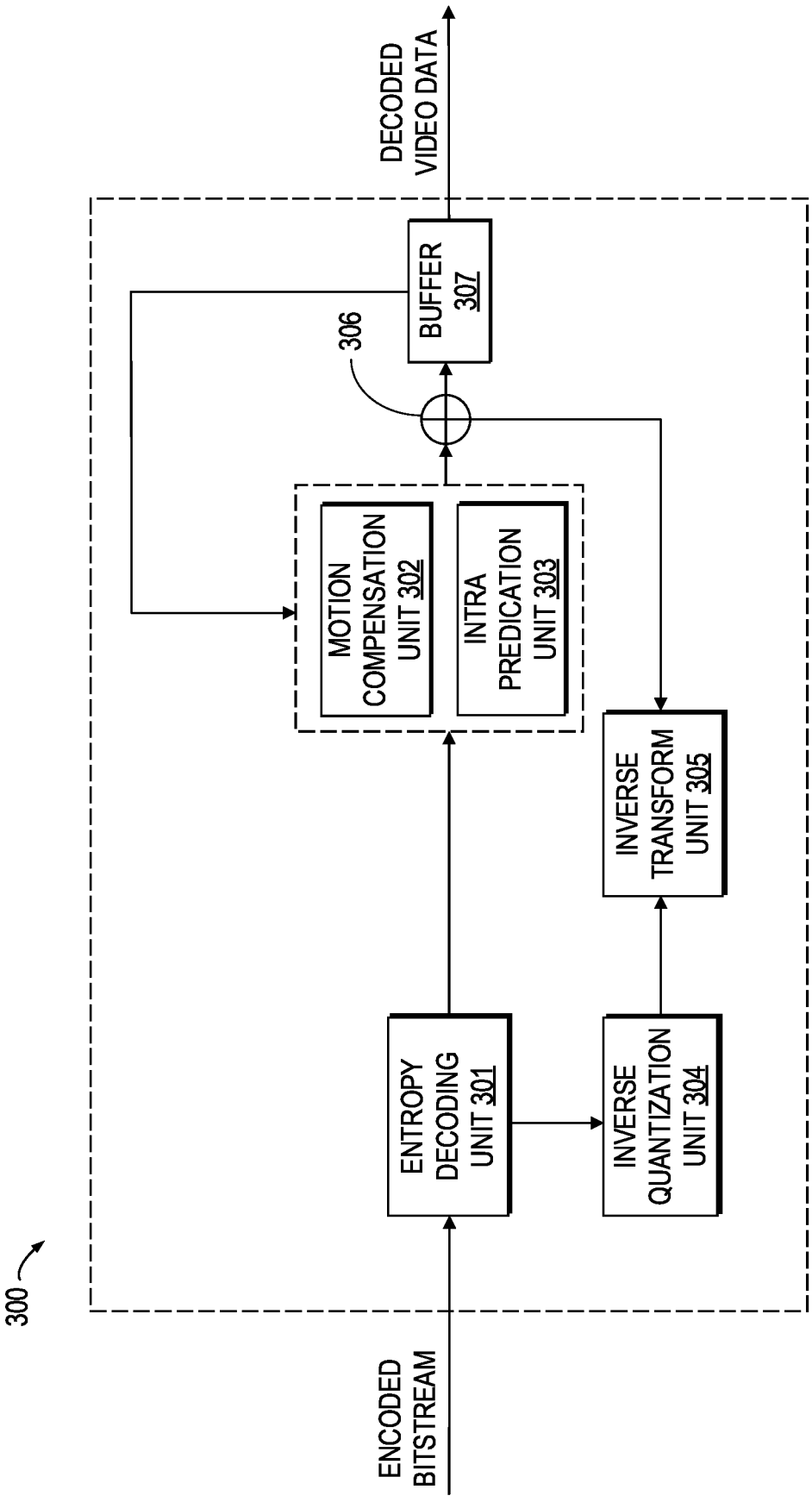
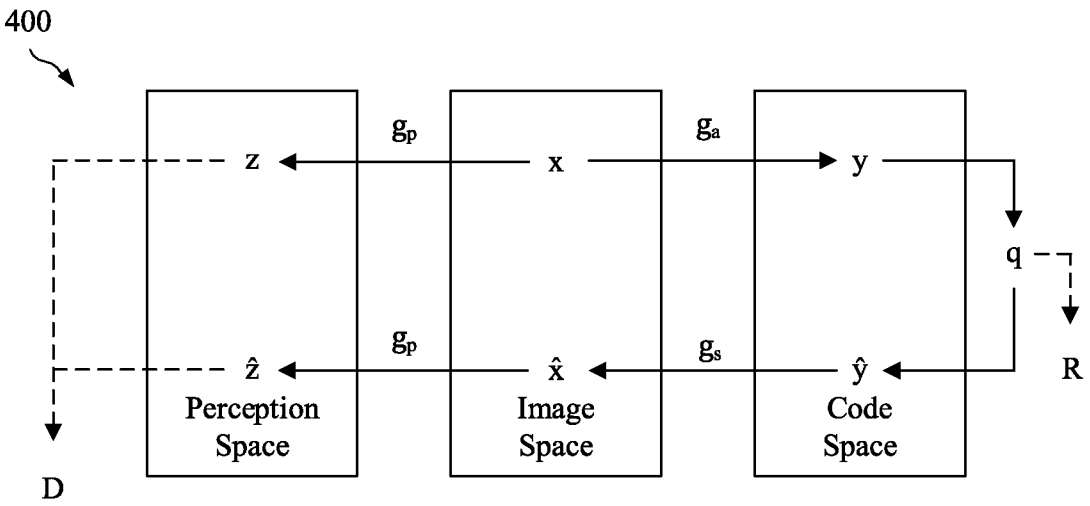

DECODED VIDEO DATA
BUFFER 307
306
MOTION COMPENSATION UNIT 302
INTRA PREDICATION UNIT 303
INVERSE TRANSFORM UNIT 305
ENTROPY DECODING UNIT 301
INVERSE QUANTIZATION UNIT 304
300
ENCODED BITSTREAM

FIG. 3



$$\min_{g_a, g_s, q} D(z, \hat{z}) + \lambda R(q(y))$$

FIG. 4

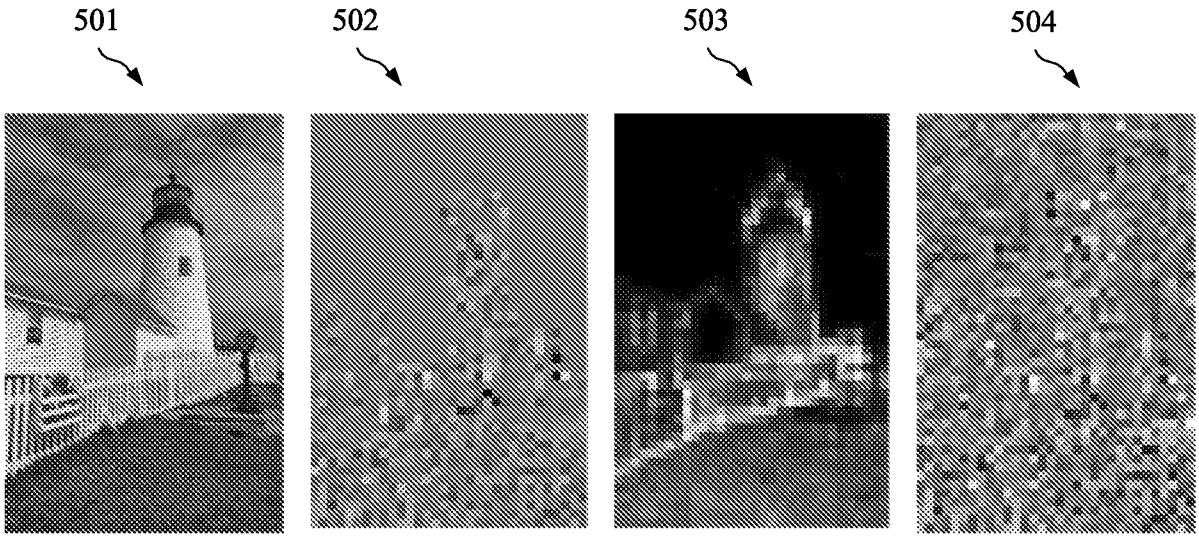


FIG. 5

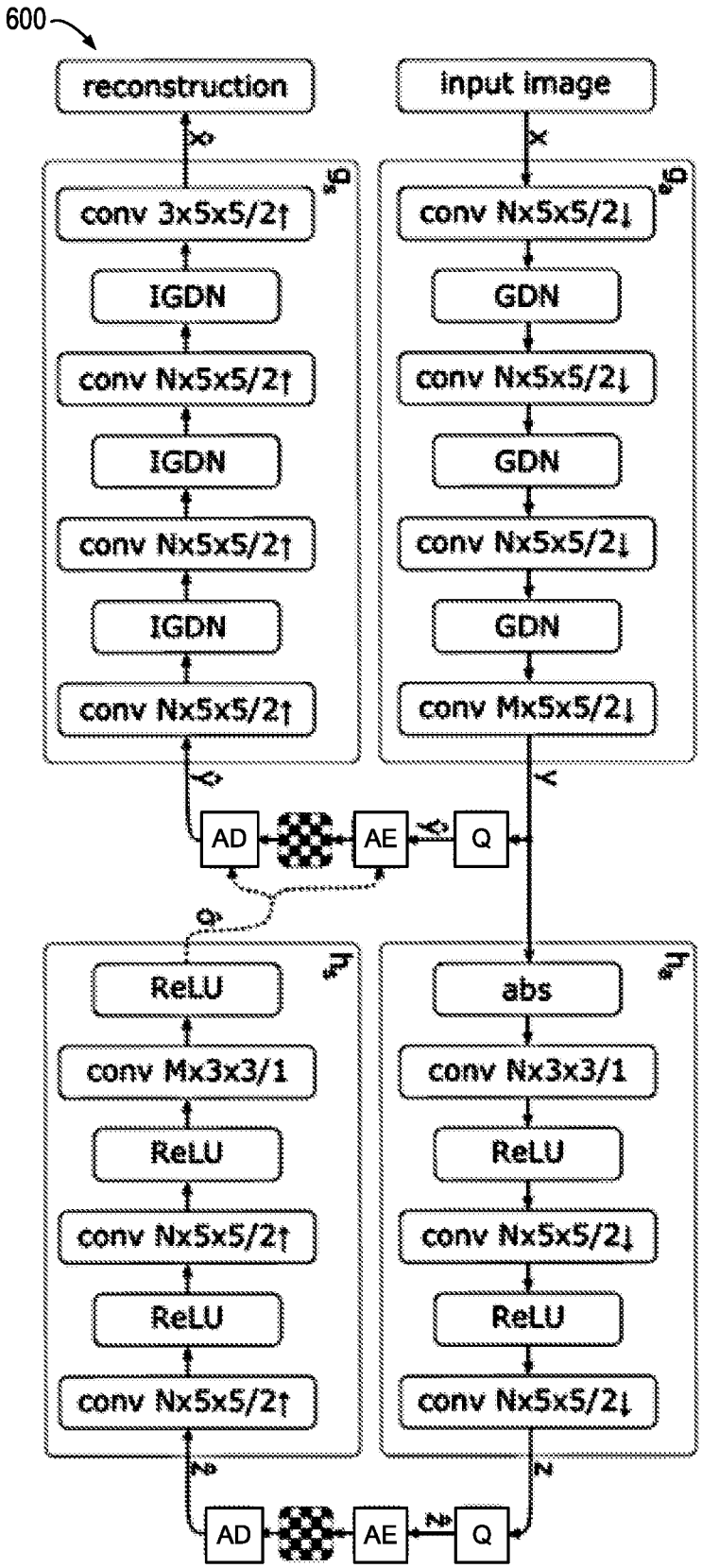

600
reconstruction
input image
conv 3x5x5/2↑
IGDN
conv Nx5x5/2↑
IGDN
conv Nx5x5/2↑
IGDN
conv Nx5x5/2↑
conv Nx5x5/2↓
GDN
conv Nx5x5/2↓
GDN
conv Nx5x5/2↓
GDN
conv Mx5x5/2↓
AD
AE
Q
ReLU
conv Mx3x3/1
ReLU
conv Nx5x5/2↑
ReLU
conv Nx5x5/2↑
abs
conv Nx3x3/1
ReLU
conv Nx5x5/2↓
ReLU
conv Nx5x5/2↓
AD
AE
Q

FIG. 6

Component	Symbol
Input Image	$\boldsymbol{x}$
Encoder	$f(\boldsymbol{x};\boldsymbol{\theta}_e)$
Latents	$\boldsymbol{y}$
Latents (quantized)	$\hat{\boldsymbol{y}}$
Decoder	$g(\hat{\boldsymbol{y}};\boldsymbol{\theta}_d)$
Hyper Encoder	$f_h(\boldsymbol{y};\boldsymbol{\theta}_{he})$
Hyper-latents	$\boldsymbol{z}$
Hyper-latents (quant.)	$\hat{\boldsymbol{z}}$
Hyper Decoder	$g_h(\hat{\boldsymbol{z}};\boldsymbol{\theta}_{hd})$
Context Model	$g_{cm}(\boldsymbol{y}_{<i};\boldsymbol{\theta}_{cm})$
Entropy Parameters	$g_{ep}(\cdot;\boldsymbol{\theta}_{ep})$
Reconstruction	$\tilde{\boldsymbol{x}}$

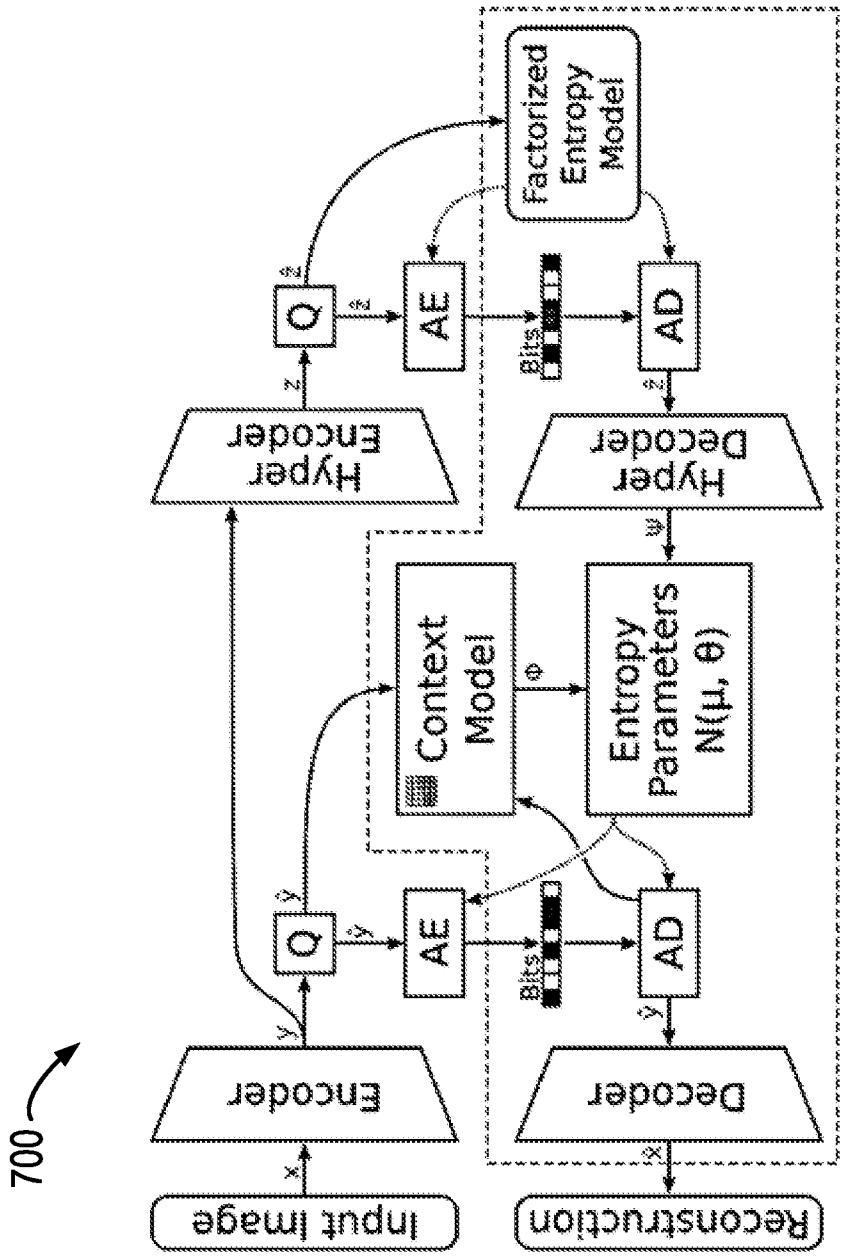


FIG. 7

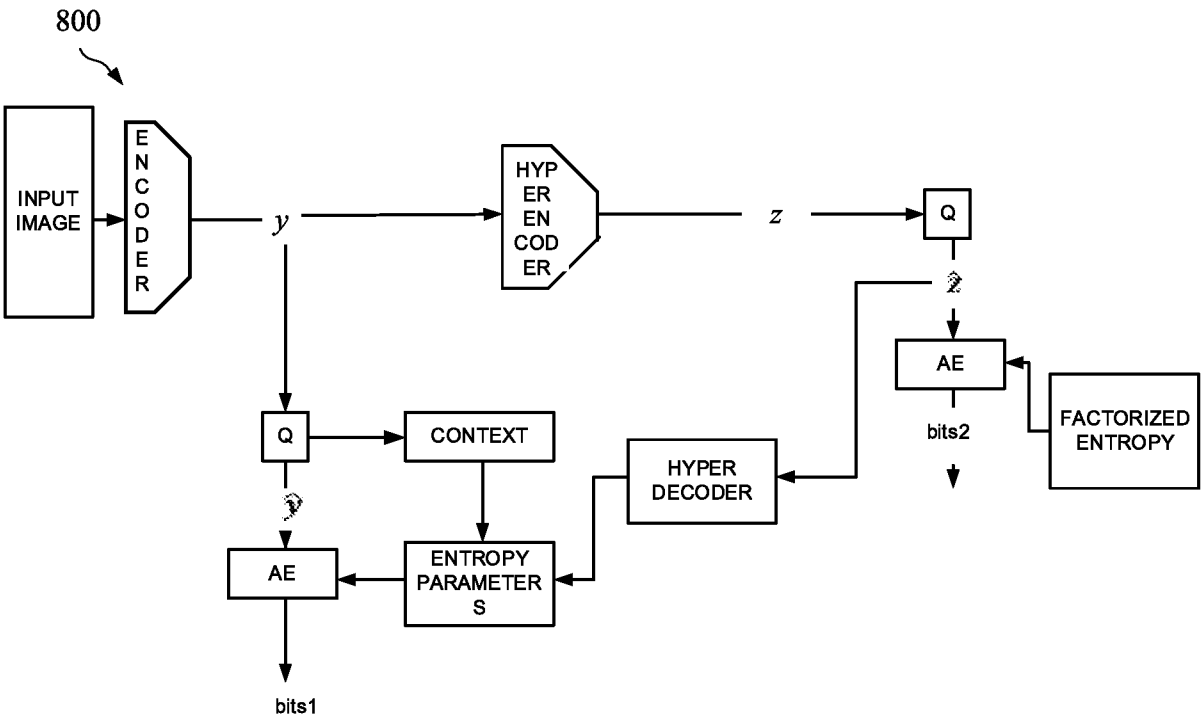

800
INPUT IMAGE
ENCODER
y
HYPER ENCODER
z
Q
AE
bits2
FACTORIZED ENTROPY
HYPER DECODER
Q
CONTEXT
AE
ENTROPY PARAMETERS
bits1

FIG. 8

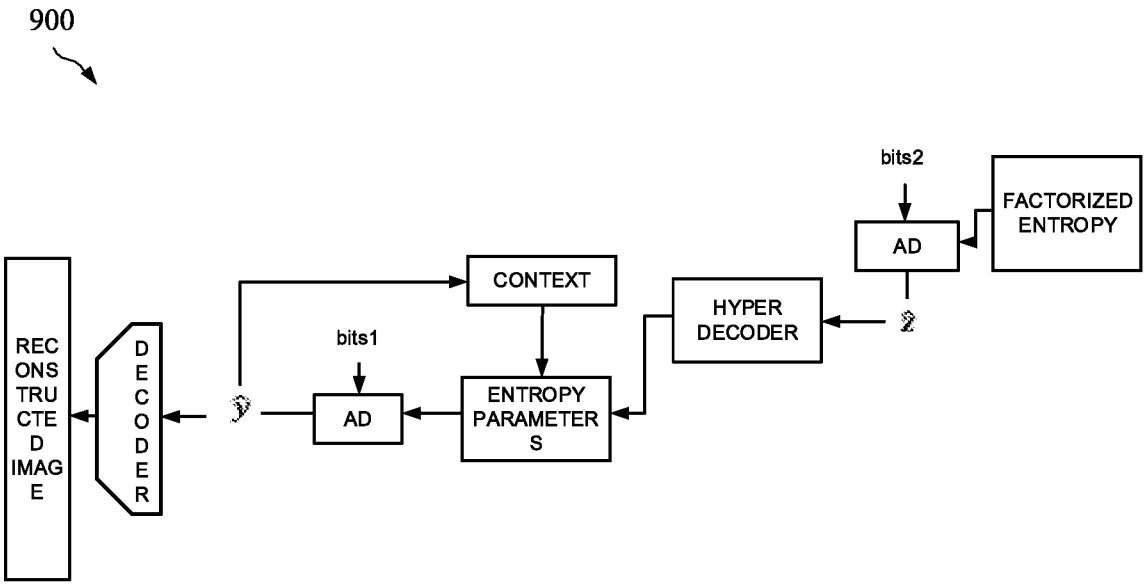

900
bits2
FACTORIZED ENTROPY
AD
CONTEXT
HYPER DECODER
bits1
AD
ENTROPY PARAMETERS
RECONSTRUCTED IMAGE
DECODER

FIG. 9

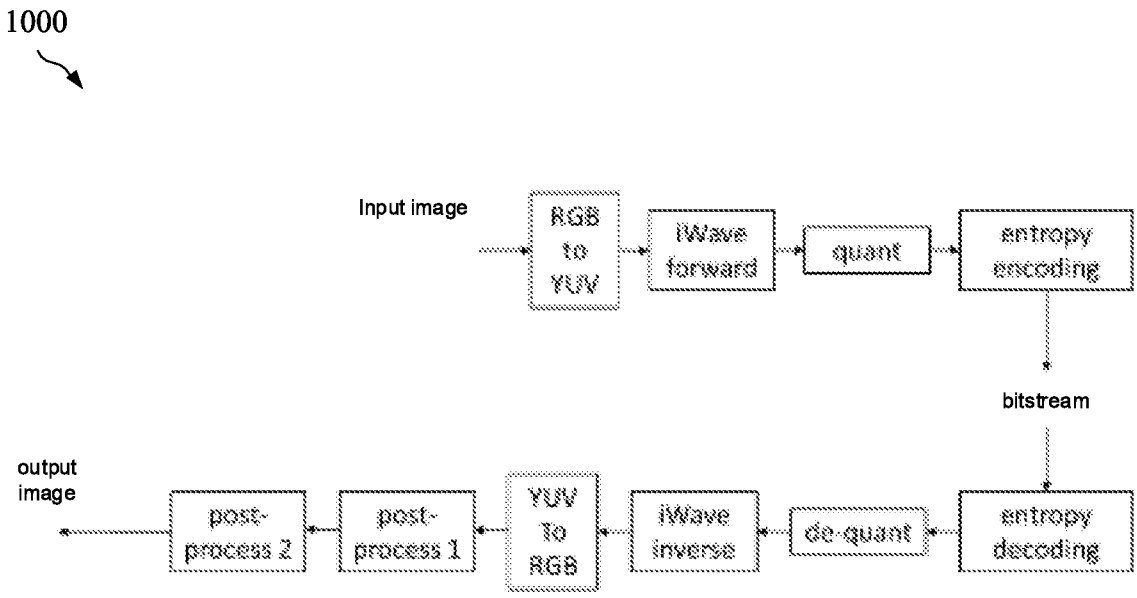

1000
Input image
RGB to YUV
iWave forward
quant
entropy encoding
bitstream
output image
post-process 2
post-process 1
YUV To RGB
iWave inverse
de-quant
entropy decoding

FIG. 10

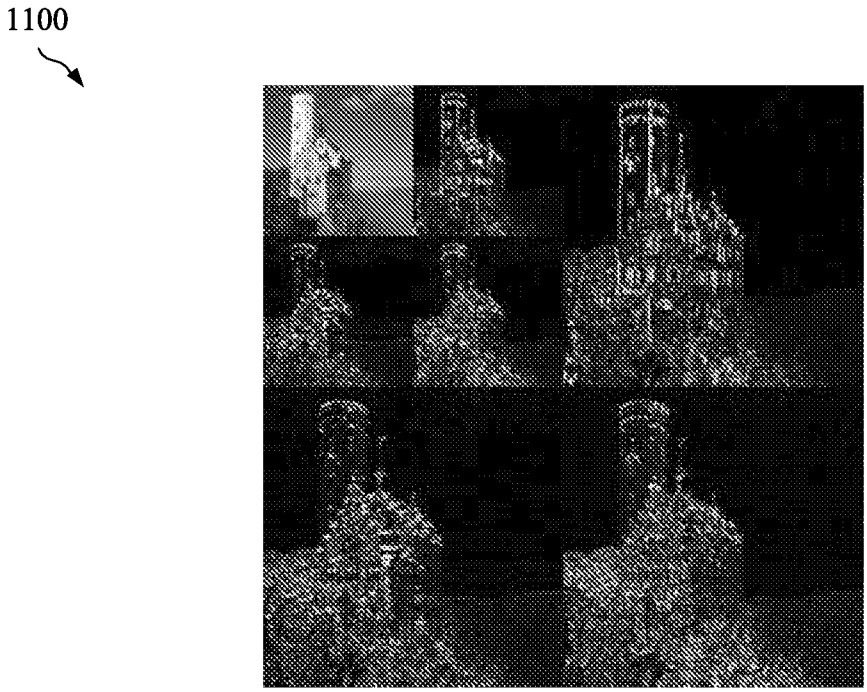

1100

FIG. 11

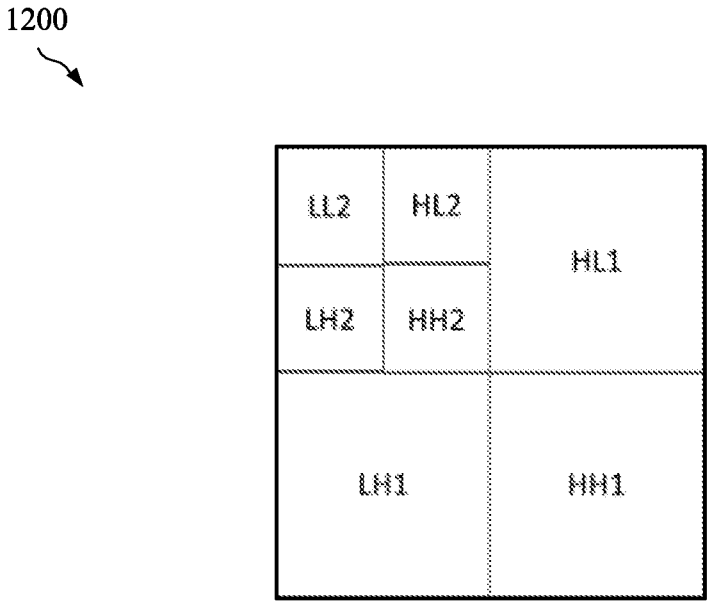


FIG. 12

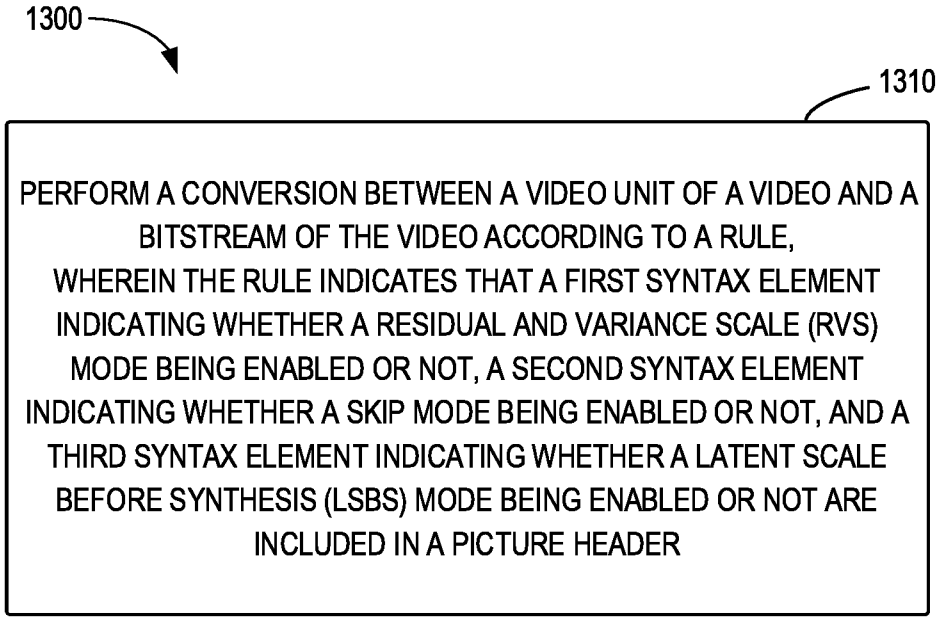

1300
1310
PERFORM A CONVERSION BETWEEN A VIDEO UNIT OF A VIDEO AND A BITSTREAM OF THE VIDEO ACCORDING TO A RULE,
WHEREIN THE RULE INDICATES THAT A FIRST SYNTAX ELEMENT INDICATING WHETHER A RESIDUAL AND VARIANCE SCALE (RVS) MODE BEING ENABLED OR NOT, A SECOND SYNTAX ELEMENT INDICATING WHETHER A SKIP MODE BEING ENABLED OR NOT, AND A THIRD SYNTAX ELEMENT INDICATING WHETHER A LATENT SCALE BEFORE SYNTHESIS (LSBS) MODE BEING ENABLED OR NOT ARE INCLUDED IN A PICTURE HEADER

FIG. 13

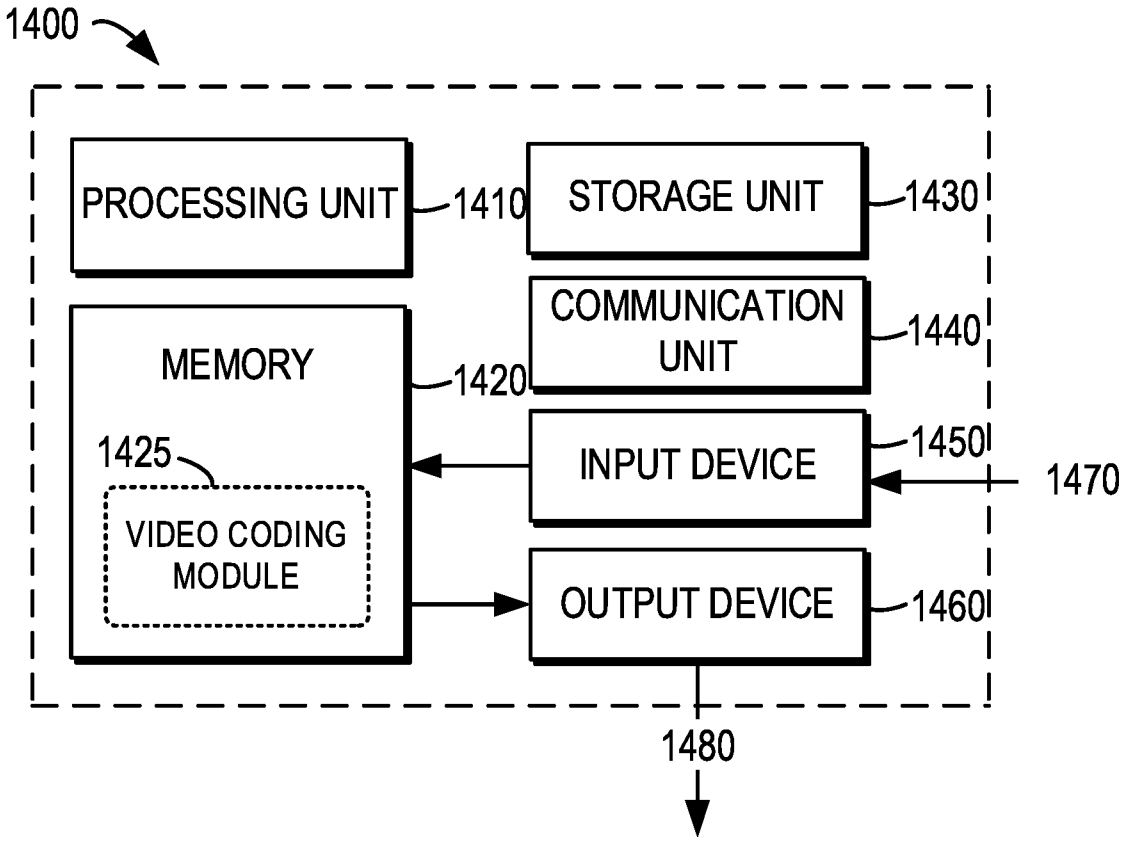

1400
PROCESSING UNIT
1410
STORAGE UNIT
1430
COMMUNICATION UNIT
1440
MEMORY
1420
1425
VIDEO CODING MODULE
INPUT DEVICE
1450
1470
OUTPUT DEVICE
1460
1480

FIG. 14

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2024/088134

A. CLASSIFICATION OF SUBJECT MATTER

H04N19/70(2014.01)i; H04N19/587(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,ENTXTC,VEN,DWPI,JVET:auto,encoder,hyper,hyperdecoder,hyperencoder,LSBS,prior,RVS,syntax element,rvs_enable_flag,residual and variance scale,lsbs_enable_flag,skip_enable_flag,latent scal+ before synthesis,skip mode, deep learning

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2022239944 A1 (LEMON INC.) 28 July 2022 (2022-07-28) description, paragraphs 0027-0239	1-56
A	US 2022394240 A1 (LEMON INC.) 08 December 2022 (2022-12-08) the whole document	1-56
A	US 2022272345 A1 (DEEP RENDER LTD) 25 August 2022 (2022-08-25) the whole document	1-56
A	WO 2021170058 A1 (BEIJING BYTEDANCE NETWORK TECHNOLOGY CO., LTD. et al.) 02 September 2021 (2021-09-02) the whole document	1-56
A	WO 2021202556 A1 (BEIJING DAJIA INTERNET INFORMATION TECHNOLOGY CO., LTD. et al.) 07 October 2021 (2021-10-07) the whole document	1-56

☐ 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
30 May 2024	**27 June 2024**

Name and mailing address of the ISA/CN	Authorized officer
CHINA NATIONAL INTELLECTUAL PROPERTY ADMINISTRATION **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China**	**YANG,Zhe** Telephone No. **(+86) 010-53961708**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2024/088134

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2022239944	A1	28 July 2022	CN	114793282	A	01 July 2022
US	2022394240	A1	08 December 2022	CN	115442618	A	06 December 2022
US	2022272345	A1	25 August 2022	KR	20230107563	A	17 July 2023
				JP	2023547874	A	14 November 2023
				US	2022286682	A1	08 September 2022
				US	2023179768	A1	08 June 2023
				EP	4233314	A1	30 August 2023
				US	2024107022	A1	28 March 2024
				WO	2022084702	A1	28 April 2022
				GB	202016824	D0	09 December 2020
WO	2021170058	A1	02 September 2021	US	2023111133	A1	13 April 2023
				US	2024073456	A1	29 February 2024
				CN	115299063	A	04 November 2022
WO	2021202556	A1	07 October 2021	CN	115362685	A	18 November 2022