



(51) International Patent Classification:

H04N 19/102 (2014.01) H04N 19/85 (2014.01)
H04N 19/55 (2014.01) G06N 3/02 (2006.01)
H04N 19/59 (2014.01) H04N 19/42 (2014.01)
H04N 19/80 (2014.01) G06N 3/08 (2023.01)
H04N 19/117 (2014.01)

(21) International Application Number:

PCT/US2024/060832

(22) International Filing Date:

18 December 2024 (18.12.2024)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

63/612,288 19 December 2023 (19.12.2023) US

(71) Applicant: **BYTEDANCE INC.** [US/US]; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US).

(72) Inventors: **ESENLIK, Semih**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **WANG, Ye-Kui**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Zhaobin**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Kai**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Li**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **XU, Jizheng**; c/o Bytedance Inc., 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US).

(74) Agent: **BINDSEIL, James J.** et al.; ASTUTE IP LAW GROUP, P.O. Box 66358, Washington, District of Columbia 20035 (US).

(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).

Published:

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

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

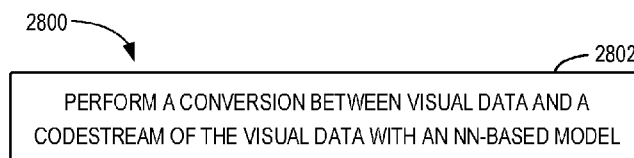


Fig. 28

(57) Abstract: Embodiments of the present disclosure provide a solution for visual data processing. A method for visual data processing is proposed. The method comprises: performing a conversion between visual data and a codestream of the visual data with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.



METHOD, APPARATUS, AND MEDIUM FOR VISUAL DATA PROCESSING**FIELDS**

[0001] Embodiments of the present disclosure relates generally to visual data processing techniques, and more particularly, to neural network-based visual data coding.

BACKGROUND

[0002] The past decade has witnessed the rapid development of deep learning in a variety of areas, especially in computer vision and image processing. Neural network was invented originally with the interdisciplinary research of neuroscience and mathematics. It has shown strong capabilities in the context of non-linear transform and classification. Neural network-based image/video compression technology has gained significant progress during the past half decade. It is reported that the latest neural network-based image compression algorithm achieves comparable rate-distortion (R-D) performance with Versatile Video Coding (VVC). With the performance of neural image compression continually being improved, neural network-based video compression has become an actively developing research area. However, coding efficiency of neural network-based image/video coding is generally expected to be further improved.

SUMMARY

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

[0004] In a first aspect, a method for visual data processing is proposed. The method comprises: performing a conversion between visual data and a codestream of the visual data with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

[0005] Based on the method in accordance with the first aspect of the present disclosure, the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

Compared with the conventional solution lacking such an indication, the proposed method can better support independent coding of different regions, and thus the coding efficiency can be improved.

[0006] In a second aspect, an apparatus for visual data 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.

[0007] 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.

[0008] In a fourth aspect, another non-transitory computer-readable recording medium is proposed. The non-transitory computer-readable recording medium stores a codestream of visual data which is generated by a method performed by an apparatus for visual data processing. The method comprises: performing a conversion between the visual data and the codestream with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

[0009] In a fifth aspect, a method for storing a codestream of visual data is proposed. The method comprises: performing a conversion between the visual data and the codestream with a neural network (NN)-based model; and storing the codestream in a non-transitory computer-readable recording medium, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

[0010] 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

[0011] 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.

[0012] Fig. 1A illustrates a block diagram that illustrates an example visual data coding system, in accordance with some embodiments of the present disclosure;

[0013] Fig. 1B is a schematic diagram illustrating an example transform coding scheme;

[0014] Fig. 2 illustrates example latent representations of an image;

[0015] Fig. 3 is a schematic diagram illustrating an example autoencoder implementing a hyperprior model;

[0016] Fig. 4 is a schematic diagram illustrating an example combined model configured to

5 jointly optimize a context model along with a hyperprior and the autoencoder;

[0017] Fig. 5 illustrates an example encoding process;

[0018] Fig. 6 illustrates an example decoding process;

[0019] Fig. 7 illustrates an example decoding process according to some embodiments of the present disclosure;

10 [0020] Fig. 8 illustrates an example learning-based image codec architecture;

[0021] Fig. 9 illustrates an example synthesis transform for learning based image coding;

[0022] Fig. 10 illustrates an example leaky Rectified Linear Unit (ReLU) activation function;

[0023] Fig. 11 illustrates an example ReLU activation function;

[0024] Fig. 12 illustrates latent tiles in synthesis transform;

15 [0025] Fig. 13 illustrates a bitstream layout;

[0026] Fig. 14 illustrates an example decoder structure;

[0027] Fig. 15 illustrates an example hyper scale decoder;

[0028] Fig. 16 illustrates an example hyper decoder;

[0029] Fig. 17 illustrates a diagram of an example multistage context modelling (MCM)

20 structure;

[0030] Fig. 18 illustrates an example implementation of primary component guided adaptive up-sampling filter;

[0031] Fig. 19 illustrates two example partition patterns, wherein equipartitioned regions shown on the left, and shifted regions shown on the right;

25 [0032] Figs. 20A-20C illustrate three example alignment schemes;

[0033] Fig. 21 illustrates latent tiles in synthesis transform;

[0034] Fig. 22 illustrates an example bitstream layout;

[0035] Fig. 23 illustrates an example decoder structure;

[0036] Fig. 24 illustrates an example hyper scale decoder;

30 [0037] Fig. 25 illustrates an example hyper decoder;

[0038] Fig. 26 illustrates a diagram of an example MCM structure;

[0039] Fig. 27 illustrates an example implementation of primary component guided adaptive up-sampling filter;

[0040] Fig. 28 illustrates a flowchart of a method for visual data processing in accordance

with embodiments of the present disclosure; and

[0041] Fig. 29 illustrates a block diagram of a computing device in which various embodiments of the present disclosure can be implemented.

[0042] Throughout the drawings, the same or similar reference numerals usually refer to the same or similar elements.

DETAILED DESCRIPTION

[0043] 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.

[0044] 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.

[0045] 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.

[0046] 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.

[0047] 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

5 [0048] Fig. 1A is a block diagram that illustrates an example visual data coding system 100 that may utilize the techniques of this disclosure. As shown, the visual data 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 visual data encoding device, and the destination device 120 can be also referred to as a visual data decoding device. In operation, the source device 110 can
10 be configured to generate encoded visual data and the destination device 120 can be configured to decode the encoded visual data generated by the source device 110. The source device 110 may include a visual data source 112, a visual data encoder 114, and an input/output (I/O) interface 116.

[0049] The visual data source 112 may include a source such as a visual data capture device.
15 Examples of the visual data capture device include, but are not limited to, an interface to receive visual data from a visual data provider, a computer graphics system for generating visual data, and/or a combination thereof.

[0050] The visual data may comprise one or more pictures of a video or one or more images. The visual data encoder 114 encodes the visual data from the visual data source 112 to
20 generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the visual data. The bitstream may include coded pictures and associated visual data. The coded picture is a coded representation of a picture. The associated visual 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
25 encoded visual data may be transmitted directly to destination device 120 via the I/O interface 116 through the network 130A. The encoded visual data may also be stored onto a storage medium/server 130B for access by destination device 120.

[0051] The destination device 120 may include an I/O interface 126, a visual data decoder 124, and a display device 122. The I/O interface 126 may include a receiver and/or a modem.
30 The I/O interface 126 may acquire encoded visual data from the source device 110 or the storage medium/server 130B. The visual data decoder 124 may decode the encoded visual data. The display device 122 may display the decoded visual 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.

[0052] The visual data encoder 114 and the visual data decoder 124 may operate according to a visual data coding standard, such as video coding standard or still picture coding standard and other current and/or further standards.

[0053] 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 visual data codecs, the disclosed techniques are applicable to other coding technologies also. Furthermore, while some embodiments describe 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 visual data processing encompasses visual data coding or compression, visual data decoding or decompression and visual data transcoding in which visual data are represented from one compressed format into another compressed format or at a different compressed bitrate.

1. Brief Summary

This disclosure is related to neural network (NN)-based image and video coding. Specifically, it is related to support of regional accessibility, wherein regional accessibility refers to the capability of correctly decoding only a regional part of an image (also referred to as a picture) or a video. The ideas may be applied individually or in various combinations, for image and/or video coding methods and specifications.

2. Introduction

The past decade has witnessed the rapid development of deep learning in a variety of areas, especially in computer vision and image processing. Inspired from the great success of deep learning technology to computer vision areas, many researchers have shifted their attention from conventional image/video compression techniques to neural image/video compression technologies. Neural network was invented originally with the interdisciplinary research of neuroscience and mathematics. It has shown strong capabilities in the context of non-linear transform and classification. Neural network-based image/video compression technology has gained significant progress during the past half decade. It is reported that the latest neural network-based image compression algorithm achieves comparable R-D performance with Versatile Video Coding (VVC), the latest video coding standard developed by Joint Video Experts Team (JVET) with experts from MPEG and VCEG. With the performance of neural image compression continually being improved, neural network-based video compression has become an actively developing research area. However, neural network-based video coding still remains in its infancy

due to the inherent difficulty of the problem.

1.1 Image/video compression

Image/video compression (also referred to as image/video coding) usually refers to the computing technology that compresses image/video into binary code to facilitate storage and transmission.

5 The binary codes may or may not support losslessly reconstructing the original image/video, termed lossless compression and lossy compression. Most of the efforts are devoted to lossy compression since lossless reconstruction is not necessary in most scenarios. Usually the performance of image/video compression algorithms is evaluated from two aspects, *i.e.* compression ratio and reconstruction quality. Compression ratio is directly related to the number
10 of binary codes, the less the better; Reconstruction quality is measured by comparing the reconstructed image/video with the original image/video, the higher the better.

Image/video compression techniques can be divided into two branches, the classical video coding methods and the neural-network-based video compression methods. Classical video coding schemes adopt transform-based solutions, in which researchers have exploited statistical
15 dependency in the latent variables (*e.g.*, DCT or wavelet coefficients) by carefully hand-engineering entropy codes modeling the dependencies in the quantized regime. Neural network-based video compression is in two flavors, neural network-based coding tools and end-to-end neural network-based video compression. The former is embedded into existing classical video codecs as coding tools and only serves as part of the framework, while the latter is a separate
20 framework developed based on neural networks without depending on classical video codecs.

In the last three decades, a series of classical video coding standards have been developed to accommodate the increasing visual content. The international standardization organizations ISO/IEC has two expert groups namely Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MPEG), and ITU-T also has its own Video Coding Experts Group (VCEG)
25 which is for standardization of image/video coding technology. The influential video coding standards published by these organizations include JPEG, JPEG 2000, H.262, H.264/AVC and H.265/HEVC. After H.265/HEVC, the Joint Video Experts Team (JVET) formed by MPEG and VCEG has been working on a new video coding standard Versatile Video Coding (VVC). The first version of VVC was released in July 2020. An average of 50% bitrate reduction is reported
30 by VVC under the same visual quality compared with HEVC.

Neural network-based image/video compression is not a new invention since there were a number of researchers working on neural network-based image coding. But the network architectures were relatively shallow, and the performance was not satisfactory. Benefit from the abundance of data and the support of powerful computing resources, neural network-based methods are better

exploited in a variety of applications. At present, neural network-based image/video compression has shown promising improvements, confirmed its feasibility. Nevertheless, this technology is still far from mature and a lot of challenges need to be addressed.

1.2 Neural networks

5 Neural networks, also known as artificial neural networks (ANN), are the computational models used in machine learning technology which 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 believed to be the capacity for processing data with multiple levels of abstraction and converting data into different kinds of representations. Note that these
10 representations 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, and thus is regarded useful especially for processing natively unstructured data, such as acoustic and visual signal, whilst processing such data has been a longstanding difficulty in the artificial intelligence field.

15 1.3 Neural networks for image compression

Existing neural networks for image compression methods can be classified in two categories, *i.e.*, pixel probability modeling and auto-encoder. The former one belongs to the predictive coding strategy, while the latter one is the transform-based solution. Sometimes, these two methods are combined together in literature.

20 1.3.1 Pixel probability modeling

According to Shannon's information theory, the optimal method for lossless coding can reach the minimal coding rate $-\log_2 p(x)$ where $p(x)$ is the probability of symbol x . A number of lossless coding methods were developed in literature and among them arithmetic coding is believed to be among the optimal ones. Given a probability distribution $p(x)$, arithmetic coding ensures that the
25 coding rate to be as close as possible to its theoretical limit $-\log_2 p(x)$ without considering the rounding error. Therefore, the remaining problem is to how to determine the probability, which is however very challenging for natural image/video due to the curse of dimensionality.

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.

$$30 \quad p(x) = p(x_1)p(x_2|x_1) \dots p(x_i|x_1, \dots, x_{i-1}) \dots p(x_{m \times n}|x_1, \dots, x_{m \times n-1}) \quad (1)$$

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, it can be difficult to estimate the conditional probability, thereby a simplified method is to limit the range of its context.

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

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

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

Neural networks were originally introduced for computer vision tasks and have been proven to be effective in regression and classification problems. Therefore, it has been proposed using neural networks to estimate the probability of $p(x_i)$ given its context $x_1, x_2, \dots, x_{i-1}$.

Most of the methods directly model the probability distribution in the pixel domain. Some researchers also attempt to model the probability distribution as a conditional one upon explicit or latent representations. That being said, it may be estimated that

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

where $\mathbf{h}$ is the additional condition and $p(\mathbf{x}) = p(\mathbf{h})p(\mathbf{x}|\mathbf{h})$, meaning the modeling is split into an unconditional one and a conditional one. The additional condition can be image label information or high-level representations.

1.3.2 Auto-encoder

Auto-encoder originates from the well-known work proposed by Hinton and Salakhutdinov. The method is trained for dimensionality reduction and consists of two parts: encoding and decoding. The encoding part converts the high-dimension input signal to low-dimension representations, typically with reduced spatial size but a greater number of channels. The decoding part attempts to recover the high-dimension input from the low-dimension representation. 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.

Fig. 1B illustrates a typical transform coding scheme. The original image $\mathbf{x}$ is transformed by the analysis network $\mathbf{g}_a$ to achieve the latent representation $\mathbf{y}$. The latent representation $\mathbf{y}$ is quantized and compressed into bits. The number of bits R is used to measure the coding rate. The quantized latent representation $\hat{\mathbf{y}}$ is then inversely transformed by a synthesis network $\mathbf{g}_s$ to obtain the reconstructed image $\hat{\mathbf{x}}$. The distortion is calculated in a perceptual space by transforming $\mathbf{x}$ and $\hat{\mathbf{x}}$ with the function $\mathbf{g}_p$.

It is intuitive to apply auto-encoder network to lossy image compression. It is only needed to encode the learned latent representation from the well-trained neural networks. However, it is not trivial to adapt auto-encoder to image compression since the original auto-encoder is not optimized

for compression thereby not efficient by directly using a trained auto-encoder. In addition, there exist other major challenges: First, the low-dimension representation should be quantized before being encoded, but the quantization is not differentiable, which is required in backpropagation while training the neural networks. Second, the objective under 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 needs to support variable rate, scalability, encoding/decoding speed, interoperability. In response to these challenges, a number of researchers have been actively contributing to this area.

The prototype auto-encoder for image compression is in Fig. 1B, which can be regarded as a *transform coding* strategy. The original image $\mathbf{x}$ is transformed with the *analysis* network $\mathbf{y} = g_a(\mathbf{x})$, where $\mathbf{y}$ is the latent representation which will be quantized and coded. The *synthesis* network will inversely transform the quantized latent representation $\hat{\mathbf{y}}$ back to obtain the reconstructed image $\hat{\mathbf{x}} = g_s(\hat{\mathbf{y}})$. The framework is trained with the rate-distortion loss function, *i.e.*, $\mathcal{L} = D + \lambda R$, where D is the distortion between $\mathbf{x}$ and $\hat{\mathbf{x}}$, R is the rate calculated or estimated from the quantized representation $\hat{\mathbf{y}}$, and λ is the Lagrange multiplier. It should be noted that D can be calculated in either pixel domain or perceptual domain. All existing research works follow this prototype and the difference might only be the network structure or loss function.

1.3.3 Hyper prior model

In the transform coding approach to image compression, the encoder subnetwork (section 2.3.2) transforms the image vector $\mathbf{x}$ using a parametric analysis transform $g_a(\mathbf{x}, \Phi_g)$ into a latent representation $\mathbf{y}$, which is then quantized to form $\hat{\mathbf{y}}$. Because $\hat{\mathbf{y}}$ is discrete-valued, it can be losslessly compressed using entropy coding techniques such as arithmetic coding and transmitted as a sequence of bits.

As evident from the middle left and middle right image of Fig. 2, there are significant spatial dependencies among the elements of $\hat{\mathbf{y}}$. Notably, their scales (middle right image) appear to be coupled spatially. An additional set of random variables $\hat{\mathbf{z}}$ can be introduced to capture the spatial dependencies and to further reduce the redundancies. In this case the image compression network is depicted in Fig. 3.

In Fig 3, the left hand of the models is the encoder g_a and decoder g_s (explained in section 2.3.2). The right-hand side is the additional hyper encoder h_a and hyper decoder h_s networks that are used to obtain $\hat{\mathbf{z}}$. In this architecture the encoder subjects the input image $\mathbf{x}$ to g_a , yielding the responses $\mathbf{y}$ with spatially varying standard deviations. The responses $\mathbf{y}$ are fed into h_a , summarizing the distribution of standard deviations in $\mathbf{z}$. $\mathbf{z}$ is then quantized ($\hat{\mathbf{z}}$), compressed, and transmitted as side information. The encoder then uses the quantized vector $\hat{\mathbf{z}}$ to estimate σ , the

spatial distribution of standard deviations, and uses it to compress and transmit the quantized image representation $\hat{\mathbf{y}}$. The decoder first recovers $\hat{\mathbf{z}}$ from the compressed signal. It then uses h_s to obtain σ , which provides it with the correct probability estimates to successfully recover $\hat{\mathbf{y}}$ as well. It then feeds $\hat{\mathbf{y}}$ into g_s to obtain the reconstructed image.

5 When the hyper encoder and hyper decoder are added to the image compression network, the spatial redundancies of the quantized latent $\hat{\mathbf{y}}$ are reduced. The rightmost image in Fig. 2 correspond to the quantized latent when hyper encoder/decoder are used. Compared to middle right image, the spatial redundancies are significantly reduced, as the samples of the quantized latent are less correlated.

10 In Fig. 2: Left: an image from the Kodak dataset. Middle left: visualization of a latent representation $\mathbf{y}$ of that image. Middle right: standard deviations σ of the latent. Right: latents $\mathbf{y}$ after the hyper prior (hyper encoder and decoder) network is introduced.

Fig. 3 illustrates Network architecture of a autoencoder implementing the hyperprior model. The left side shows an image autoencoder network, the right side corresponds to the hyperprior subnetwork. The analysis and synthesis transforms are denoted as g_a and g_s . Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder, respectively. The hyperprior model consists of two subnetworks, hyper encoder (denoted with h_a) and hyper decoder (denoted with h_s). The hyper prior model generates a quantized hyper latent ($\hat{\mathbf{z}}$) which comprises information about the probability distribution of the samples of the quantized latent $\hat{\mathbf{y}}$.
 15 $\hat{\mathbf{z}}$ is included in the bitstream and transmitted to the receiver (decoder) along with $\hat{\mathbf{y}}$.

1.3.4 Context model

Although the hyper prior model improves the modelling of the probability distribution of the quantized latent $\hat{\mathbf{y}}$, additional improvement can be obtained by utilizing an autoregressive model that predicts quantized latents from their causal context (Context Model).

25 The term auto-regressive means that the output of a process is later used as input to it. For example the context model subnetwork generates one sample of a latent, which is later used as input to obtain the next sample.

Fig. 4 is a schematic diagram illustrating an example combined model configured to jointly optimize a context model along with a hyperprior and the autoencoder. The following Table 1
 30 illustrates meaning of different symbols.

Table 1 – Illustration of symbols

Component	Symbol
Input Image	$\mathbf{x}$
Encoder	$f(\mathbf{x}; \theta_e)$

Latents	$\mathbf{y}$
Latents (quantized)	$\hat{\mathbf{y}}$
Decoder	$g(\hat{\mathbf{y}}; \boldsymbol{\theta}_d)$
Hyper Encoder	$f_h(\mathbf{y}; \boldsymbol{\theta}_{he})$
Hyper-Latents	$\mathbf{z}$
Hyper-Latents (quantized)	$\hat{\mathbf{z}}$
Hyper Decoder	$g_h(\hat{\mathbf{z}}; \boldsymbol{\theta}_{hd})$
Context Model	$g_{cm}(\mathbf{y}_{<i}; \boldsymbol{\theta}_{cm})$
Entropy Parameters	$g_{ep}(\cdot; \boldsymbol{\theta}_{ep})$
Reconstruction	$\hat{\mathbf{x}}$

A joint architecture can be utilized where both hyper prior model subnetwork (hyper encoder and hyper decoder) and a context model subnetwork are utilized. The hyper prior and the context model are combined to learn a probabilistic model over quantized latents $\hat{\mathbf{y}}$, which is then used for entropy coding. As depicted in Fig. 4, the outputs of 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 $\hat{\mathbf{y}}$ from the bitstream by arithmetic decoder (AD) module.

Fig 4 illustrates 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 ($\hat{\mathbf{y}}$) and quantized hyper-latents ($\hat{\mathbf{z}}$), which are compressed into a bitstream using an arithmetic encoder (AE) and decompressed by an arithmetic decoder (AD). The highlighted region corresponds to the components that are executed by the receiver (i.e. a decoder) to recover an image from a compressed bitstream.

Typically the latent samples are modeled as gaussian distribution or gaussian mixture models (not limited to). According to Fig. 4, 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 $\boldsymbol{\sigma}$).

1.3.5 Gained variational autoencoders (G-VAE)

Typically, neural network-based image/video compression methodologies need to train multiple models to adapt to different rates. Gained variational autoencoders (G-VAE) is the variational

autoencoder with a pair of gain units , which is designed to achieve continuously variable rate adaptation using a single model. It comprises of a pair of gain units, which are typically inserted to the output of encoder and input of decoder. The output of the encoder is defined as the latent representation $y \in R^{c \times h \times w}$, where c, h, w represent the number of channels, the height and width of the latent representation. Each channel of the latent representation is denoted as $y_{(i)} \in R^{h \times w}$, where $i = 0, 1, \dots, c - 1$. A pair of gain units include a gain matrix $M \in R^{c \times n}$ and an inverse gain matrix, where n is the number of gain vectors. The gain vector can be denoted as $m_s = \{\alpha_{s(0)}, \alpha_{s(1)}, \dots, \alpha_{s(c-1)}\}$, $\alpha_{s(i)} \in R$ where s denotes the index of the gain vectors in the gain matrix.

The motivation of gain matrix is similar to the quantization table in JPEG by controlling the quantization loss based on the characteristics of different channels. To apply the gain matrix to the latent representation, each channel is multiplied with the corresponding value in a gain vector.

$$\bar{y}_s = y \odot m_s$$

where $\odot$ is channel-wise multiplication, i.e., $\bar{y}_{s(i)} = y_{(i)} \times \alpha_{s(i)}$, and $\alpha_{s(i)}$ is the i -th gain value in the gain vector m_s . The inverse gain matrix used at the decoder side can be denoted as $M' \in R^{c \times n}$, which consists of n inverse gain vectors, i.e., $M' = \{\delta_{s(0)}, \delta_{s(1)}, \dots, \delta_{s(c-1)}\}$, $\delta_{s(i)} \in R$. The inverse gain process is expressed as:

$$y'_s = \hat{y} \odot m'_s$$

where $\hat{y}$ is the decoded quantized latent representation and y'_s is the inversely gained quantized latent representation, which will be fed into the synthesis network.

To achieve continuous variable rate adjustment, interpolation is used between vectors. Given two pairs of gain vectors $\{m_t, m'_t\}$ and $\{m_r, m'_r\}$, the interpolated gain vector can be obtained via the following equations.

$$m_v = [(m_r)^l \cdot (m_t)^{1-l}]$$

$$m'_v = [(m'_r)^l \cdot (m'_t)^{1-l}]$$

where $l \in R$ is an interpolation coefficient, which controls the corresponding bit rate of the generated gain vector pair. Since l is a real number, an arbitrary bit rate between the given two gain vector pairs can be achieved.

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

The fig 4. corresponds to the state of the art compression method. In this section and the next, the encoding and decoding processes will be described separately.

The Fig. 5 depicts the encoding process. The input image is first processed with an encoder subnetwork. The encoder transforms the input image into a transformed representation called latent, denoted by y . y is then input to a quantizer block, denoted by Q , to obtain the quantized

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

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 $\hat{\mathbf{y}}$. the latent $\mathbf{y}$ is input to hyper encoder, which outputs the hyper latent (denoted by $\mathbf{z}$). The hyper latent is then quantized ($\hat{\mathbf{z}}$) 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 ($\hat{\mathbf{y}}$).

The Entropy Parameters subnetwork generates the probability distribution estimations, that are used to encode the quantized latent $\hat{\mathbf{y}}$. The information that is generated by the Entropy Parameters typically include a mean μ and scale (or variance) σ parameters, that are together used to obtain a gaussian probability distribution. A gaussian distribution of a random variable x is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

wherein the parameter μ is the mean or expectation of the distribution (and also its median and mode), while the parameter σ is its standard deviation (or variance, or scale). In order to define a gaussian distribution, the mean and the variance need to be determined. In an existing design, the entropy parameters module are used to estimate the mean and the variance values.

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 $\hat{\mathbf{y}}$ is typically a matrix composed of many samples. The samples can be indicated using indices, such as $\hat{\mathbf{y}}[i,j,k]$ or $\hat{\mathbf{y}}[i,j]$ depending on the dimensions of the matrix $\hat{\mathbf{y}}$. The samples $\hat{\mathbf{y}}[i,j]$ 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, wherein the samples in a row are processed from left to right. In such a scenario (wherein 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 $\hat{\mathbf{y}}[i,j]$, 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 $\hat{\mathbf{y}}$ into bitstream (bits1).

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.

In the above description, the all of the elements in Fig. 5 are collectively called encoder. The analysis transform that converts the input image into latent representation is also called an encoder (or auto-encoder).

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

The Fig. 6 depicts the decoding process separately.

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 $\hat{\mathbf{z}}$, 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 $\hat{\mathbf{z}}$ that was generated by the encoder can be reconstructed at the decoder without any change. After obtaining of $\hat{\mathbf{z}}$, 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 $\hat{\mathbf{y}}$ without any loss. As a result the identical version of the quantized latent $\hat{\mathbf{y}}$ that was obtained in the encoder can be obtained in the decoder.

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 $\hat{\mathbf{y}}$ is input to the synthesis transform (denoted as decoder in Fig. 6) module to obtain the reconstructed image.

In the above description, the all of the elements in Fig. 6 are collectively called decoder. The synthesis transform that converts the quantized latent into reconstructed image is also called a decoder (or auto-decoder).

1.4 Neural networks for video compression

Similar to conventional 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 comes later than neural network-based image compression but needs far more efforts to solve the challenges due to its complexity. Starting from 2017, a few researchers have been working on neural network-based video compression schemes. Compared with image compression, video compression needs efficient methods to remove inter-picture redundancy. Inter-picture prediction is then a crucial step in these works. Motion estimation and compensation is widely adopted but is not implemented by trained neural networks until recently.

Studies on 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, it requires the decoding can be started from any point of the sequence, typically divides the entire sequence into multiple individual segments and each segment can be decoded independently. In low-latency case, it aims at reducing decoding time thereby usually merely temporally previous frames can be used as reference frames to decode subsequent frames.

1.5 Preliminaries

Almost all the natural image/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, $\mathbb{D} = \{0, 1, 2, \dots, 255\}$ is a common 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.

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 traditional codecs typically use YUV color space to represent the video sequences. In YUV color space, an image is decomposed into three channels, namely Y, Cb and Cr, where Y is the luminance component and Cb/Cr are the chroma components. The benefits come from that Cb and Cr are typically down sampled to achieve pre-compression since human vision system is less sensitive to chroma components.

A color video sequence is composed of multiple color images, called frames, to record scenes at different timestamps. For example, in the RGB color space, a color video can be denoted by $X = \{x_0, x_1, \dots, x_t, \dots, x_{T-1}\}$ where T is the number of frames in this video sequence, $x \in \mathbb{D}^{m \times n}$. 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), about 2.32 Gbps, which needs a lot storage thereby definitely needs to be compressed before transmission over the internet.

- 5 Usually the lossless methods can achieve compression ratio of about 1.5 to 3 for natural images, which is clearly below requirement. Therefore, lossy compression is developed to achieve further 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, *i.e.*, mean-squared-error (MSE). For a grayscale image, MSE can be calculated with the following
10 equation.

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

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

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

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).

- To compare different lossless compression schemes, it is sufficient to compare either the compression ratio given the resulting rate or vice versa. However, to compare different lossy compression methods, it has to take into account both the rate and reconstructed quality. For example, to calculate the relative rates at several different quality levels, and then to average the
15 rates, is a commonly adopted method; the average relative rate is known as Bjontegaard's delta-rate (BD-rate). There are other important aspects to evaluate image/video coding schemes, including encoding/decoding complexity, scalability, robustness, and so on.

1.6 Separate processing of luma and chroma components of an image

- Fig. 7 illustrates the decoding process according to some example embodiments of the present
20 disclosure. According to one implementation, the luma and chroma components of an image can be decoded using separate subnetworks. In Fig. 7, the luma component of the image is processed by the subnetworks "Synthesis", "Prediction fusion", "Mask Conv", "Hyper Decoder", "Hyper scale decoder" etc. Whereas the chroma components are processed by the subnetworks: "Synthesis UV", "Prediction fusion UV", "Mask Conv UV", "Hyper Decoder UV", "Hyper
25 scale decoder UV" etc.

A benefit of the above separate processing is that the computational complexity of the processing

of an image is reduced by application of separate processing. Typically in neural network based image and video decoding, the computational complexity is proportional to the square of the number of feature maps. If the number of total feature maps is equal to 192 for example, computational complexity will be proportional to 192×192 . On the other hand if the feature maps are divided into 128 for luma and 64 for chroma (in the case of separate processing), the computational complexity is proportional to $128 \times 128 + 64 \times 64$, which corresponds to a reduction in complexity by 45%. Typically the separate processing of luma and chroma components of an image does not result in a prohibitive reduction in performance, as the correlation between the luma and chroma components are typically very small.

10 The processing (Decoding process) in Fig. 7 can be explained below:

1. Firstly, the factorized entropy model is used to decode the quantized latents for luma and chroma, i.e., $\hat{\mathbf{z}}$ and $\hat{\mathbf{z}}_{uv}$ in Fig. 7.
2. The probability parameters (e.g. **variance**) generated by the second network are used to generate a quantized residual latent by performing the arithmetic decoding process.
- 15 3. The quantized residual latent is inversely gained with the inverse gain unit (iGain) as shown in orange color in Fig. 7. The outputs of the inverse gain units are denoted as $\hat{\mathbf{w}}$ and $\hat{\mathbf{w}}_{uv}$ for luma and chroma components, respectively.
4. For the luma component, the following steps are performed in a loop until all elements of $\hat{\mathbf{y}}$ are obtained:
 - 20 a. A first subnetwork is used to estimate a **mean** value parameter of a quantized latent ($\hat{\mathbf{y}}$), using the already obtained samples of $\hat{\mathbf{y}}$.
 - b. The quantized residual latent $\hat{\mathbf{w}}$ and the **mean** value are used to obtain the next element of $\hat{\mathbf{y}}$.
5. After all of the samples of $\hat{\mathbf{y}}$ are obtained, a synthesis transform can be applied to obtain the reconstructed image.
- 25 6. For chroma component, step 4 and 5 are the same but with a separate set of networks.
7. The decoded luma component is used as additional information to obtain the chroma component. Specifically, the Inter Channel Correlation Information filter sub-network (ICCI) is used for chroma component restoration. The luma is fed into the ICCI sub-network as additional information to assist the chroma component decoding.
- 30 8. Adaptive color transform (ACT) is performed after the luma and chroma components are reconstructed.

The module named ICCI is a neural-network based postprocessing module. The example embodiments of the present disclosure are not limited to the UCCI subnetwork, any other neural

network based postprocessing module might also be used.

An exemplary implementation of some example embodiments of the present disclosure is depicted in Fig. 7 (the decoding process). The framework comprises two branches for luma and chroma components respectively. In each of the branch, the first subnetwork comprises the context, prediction and optionally the hyper decoder modules. The second network comprises the hyper scale decoder module. The quantized hyper latent are $\hat{\mathbf{z}}$ and $\hat{\mathbf{z}}_{uv}$. The arithmetic decoding process generates the quantized residual latents, which are further fed into the iGain units to obtain the gained quantized residual latents $\hat{\mathbf{w}}$ and $\hat{\mathbf{w}}_{uv}$.

After the residual latent is obtained, a recursive prediction operation is performed to obtain the latent $\hat{\mathbf{y}}$ and $\hat{\mathbf{y}}_{uv}$. The following steps describe how to obtain the samples of latent $\hat{\mathbf{y}}[:, \mathbf{i}, \mathbf{j}]$, and the chroma component is processed in the same way but with different networks.

1. An autoregressive context module is used to generate first input of a prediction module using the samples $\hat{\mathbf{y}}[:, \mathbf{m}, \mathbf{n}]$ where the (m, n) pair are the indices of the samples of the latent that are already obtained.
2. Optionally the second input of the prediction module is obtained by using a hyper decoder and a quantized hyper latent $\hat{\mathbf{z}}_1$.
3. Using the first input and the second input, the prediction module generates the mean value $\mathbf{mean}[:, \mathbf{i}, \mathbf{j}]$.
4. The mean value $\mathbf{mean}[:, \mathbf{i}, \mathbf{j}]$ and the quantized residual latent $\hat{\mathbf{w}}[:, \mathbf{i}, \mathbf{j}]$ are added together to obtain the latent $\hat{\mathbf{y}}[:, \mathbf{i}, \mathbf{j}]$.
5. The steps 1-4 are repeated for the next sample.

Whether to and/or how to apply at least one method disclosed in the document may be signaled from the encoder to the decoder, e.g. in the bitstream.

Alternatively, whether to and/or how to apply at least one method disclosed in the document may be determined by the decoder based on coding information, such as dimensions, color format, etc. Alternative or additionally, the modules named MS1, MS2 or MS3+O (in Fig. 7), might be included in the processing flow. The said modules might perform an operation to their input by multiplying the input with a scalar or adding an additive component to the input to obtain the output. The scalar or the additive component that are used by the said modules might be indicated in a bitstream.

The module named RD or the module named AD in Fig. 7 might be an entropy decoding module. It might be a range decoder or an arithmetic decoder or the like.

The example embodiments of the present disclosure described herein are not limited to the specific combination of the units exemplified in Fig. 7. Some of the modules might be missing and some

of the modules might be displaced in processing order. Also additional modules might be included. For example:

1. The ICCI module might be removed. In that case the output of the Synthesis module and the Synthesis UV module might be combined by means of another module, that might be based on neural networks.
2. One or more of the modules named MS1, MS2 or MS3+O might be removed. The core of the proposed solution is not affected by the removing of one or more of the said scaling and adding modules.

In Fig. 7, other operations that are performed during the processing of the luma and chroma components are also indicated using the star symbol. These processes are denoted as MS1, MS2, MS3+O. These processing might be, but not limited to, adaptive quantization, latent sample scaling, and latent sample offsetting operations. For example, in an adaptive quantization process might correspond to scaling of a sample with multiplier before the prediction process, wherein the multiplier is predefined or whose value is indicated in the bitstream. The latent scaling process might correspond to the process where a sample is scaled with a multiplier after the prediction process, wherein the value of the multiplier is either predefined or indicated in the bitstream. The offsetting operation might correspond to adding an additive element to the sample, again wherein the value of the additive element might be indicated in the bitstream or inferred or predetermined. Another operation might be tiling operation, wherein samples are first tiled (grouped) into overlapping or non-overlapping regions, wherein each region is processed independently. For example the samples corresponding to the luma component might be divided into tiles with a tile height of 20 samples, whereas the chroma components might be divided into tiles with a tile height of 10 samples for processing.

Another operation might be application of wavefront parallel processing. In wavefront parallel processing, a number of samples might be processed in parallel, and the amount of samples that can be processed in parallel might be indicated by a control parameter. The said control parameter might be indicated in the bitstream, be inferred, or can be predetermined. In the case of separate luma and chroma processing, the number of samples that can be processed in parallel might be different, hence different indicators can be signalled in the bitstream to control the operation of luma and chrome processing separately.

1.7 Colors separation and conditional coding

In one example the primary and secondary color components of an image are coded separately, using networks with similar architecture, but different number of channels as shown in 8. All boxes with same names are sub-networks with the similar architecture, only input-output tensor size and

number of channels are different. Number of channels for primary component is $C_p = 128$, for secondary components is $C_s = 64$. The vertical arrows (with arrowhead pointing downwards) indicate data flow related to secondary color components coding. Vertical arrows show data exchange between primary and secondary components pipelines.

- 5 The input signal to be encoded is notated as x , latent space tensor in bottleneck of variational auto-encoder is y . Subscript “Y” indicates primary component, subscript “UV” is used for concatenated secondary components, there are chroma components.

Fig. 8 illustrates learning-based image codec architecture.

- First the input image that has RGB color format is converted to primary (Y) and secondary components(UV). The primary component x_Y is coded independently from secondary components x_{UV} and the coded picture size is equal to input/decoded picture size. The secondary components are coded conditionally, using x_Y as auxiliary information from primary component for encoding x_{UV} and using $\hat{y}_Y$ as a latent tensor with auxiliary information from primary component for decoding $\hat{y}_{UV}$ reconstruction. The codec structure for primary component and secondary components are almost identical except the number of channels, size of the channels and the several entropy models for transforming latent tensor to bitstream, therefore primary and secondary latent tensor will generate two different bitstream based on two different entropy models. Prior to the encoding x_Y , x_{UV} goes through a module which adjusts the sample location by down-sampling (marked as “s↓” on Fig. 1), this essentially means that coded picture size for secondary component is different from the coded picture size for primary component. The scaling factor s is variable, but the default scaling factor is $s = 2$. The size of auxiliary input tensor in conditional coding is adjusted in order the encoder receives primary and secondary components tensor with the same picture size. After reconstruction, the secondary component is rescaled to the original picture size with a neural-network based upsampling filter module (“NN-color filter s↑” on Fig. 1), which outputs secondary components up-sampled with factor s .

- The example in Fig. 8 exemplifies an image coding system, where the input image is first transformed into primary (Y) and secondary components (UV). The outputs $\hat{x}_Y$, $\hat{x}_{UV}$ are the reconstructed outputs corresponding to the primary and secondary components. At the end of the processing, $\hat{x}_Y$, $\hat{x}_{UV}$ are converted back to RGB color format. Typically the x_{UV} is downsampled (resized) before processing with the encoding and decoding modules (neural networks). For example the size of the x_{UV} might be reduced by a factor of 50% in each of the vertical and horizontal dimensions. Therefore the processing of the secondary component includes approximately $50\% \times 50\% = 25\%$ less samples, therefore it is computationally less complex.

1.8 Cropping operation in neural network based coding

Fig. 9 illustrates synthesis transform example for learning based image coding. The example synthesis transform above includes a sequence of 4 convolutions with up-sampling with stride of 2. The synthesis transform sub-Net is depicted on Fig. 9. The size of the tensor in different parts of synthesis transform before cropping layer is the diagram on Fig. 9.

- 5 The cropping layer changes tensor size $h_d \times w_d$ to $h_{d-1} \times w_{d-1}$, where $h_d = 2 \cdot \text{ceil}(H/2^d)$; $w_d = 2 \cdot \text{ceil}(W/2^d)$; here d is the depth of proceeding convolution in the codec architecture. For primary component **Synthesis Transform** receives input tensor with size $h \times w$; $h = \text{ceil}(H/16)$; $w = \text{ceil}(W/16)$. The output of **Synthesis Transform** for primary component is $1 \times h_0 \times w_0$, where $h_0 = H$; $w_0 = W$.
- 10 For secondary component **Synthesis Transform** receives input tensor with size $h_{UV} \times w_{UV}$; $h_{UV} = \text{ceil}(\text{ceil}(H/s)/16)$; $w_{UV} = \text{ceil}(\text{ceil}(W/s)/16)$. The output of the **Synthesis Transform** for primary component is $2 \times h_{UV0} \times w_{UV0}$, where $h_{UV0} = \text{ceil}(H/s)$; $w_{UV0} = \text{ceil}(W/s)$. For secondary components input sizes are $h_0 = \text{ceil}(H/s)$; $w_0 = \text{ceil}(W/s)$, where s is the scale factor. The scale factor might be 2 for example, wherein the secondary component is
- 15 downsampled by a factor of 2.

Based on the above explanation, the operation of the cropping layers depend on the output size H, W and the depth of the cropping layer. The depth of the left-most cropping layer in Fig. 9 is equal to 0. The output of this cropping layer must be equal to H, W (the output size), if the size of the input of this cropping layer is greater than H or W in horizontal or vertical dimension

20 respectively, cropping needs to be performed in that dimension. The second cropping layer counting from left to right has a depth of 1. The output of the second cropping layer must be equal to $h_1 = 2 \cdot \text{ceil}(H/2^1)$; $w_1 = 2 \cdot \text{ceil}(W/2^1)$, which means if the input of this second cropping layer is greater than h_1, w_1 in any dimension, than cropping is applied in that dimension. In summary, the operation of cropping layers are controlled by the output size H, W . In one example

25 if H and W are both equal to 16, then the cropping layers do not perform any cropping. On the other hand if H and W are both equal to 17, then all 4 cropping layers are going to perform cropping.

1.9 Bitwise shifting

The bitwise shift operator can be represented using the function $\text{bitshift}(x, n)$, where n is an integer number. If n is greater than 0, it corresponds to right-shift operator ($>>$), which moves

30 the bits of of the input to the right, and the left-shift operator ($<<$), which moves the bits to the left. In another words the $\text{bitshift}(x, n)$ operation corresponds to:

$$\begin{aligned}\text{bitshift}(x, n) &= x * 2^n, \text{ or} \\ \text{bitshift}(x, n) &= \text{floor}(x * 2^n), \text{ or} \\ \text{bitshift}(x, n) &= x / 2^n.\end{aligned}$$

The output of the bitshift operation is an integer value. In some implementations, the floor() function might be added to the definition.

Floor(x) is equal to the largest integer less than or equal to x.

The “/” operator or the integer division operator: It is an operation that comprises division and truncation of the result toward zero. For example, 7 / 4 and -7 / -4 are truncated to 1 and -7 / 4 and 7 / -4 are truncated to -1.

$$\text{rightshift}(x, n) = x \gg n \text{ or}$$

$$\text{leftshift}(x, n) = x \ll n$$

Equation 3: alternative implementation of the bitshift operator as rightshift or leftshift.

10 $x \gg y$ Arithmetic right shift of a two's complement integer representation of x by y binary digits.

This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

$x \ll y$ Arithmetic left shift of a two's complement integer representation of x by y binary digits.

15 This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

1.10 Convolution operation

The convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel. In some cases the convolution operation might comprise a “bias”, which is added to the output of the elementwise multiplication operation.

The convolution operation might be described by the following mathematical formula. An output out1 can be obtained as:

$$25 \quad \text{out1}[x, y] = \text{conv1}(I) = \sum_{k=0}^M \sum_{i=0}^N \sum_{j=0}^P w1_k[i, j] \times I_k[x + i, y + j] + K1$$

wherein w1 are the multiplication factors, K1 is called a bias (an additive term) and I_k is the k^{th} input, and N is the kernel size in one direction and P is the kernel size in another direction. The convolution layer might consist of convolution operations wherein more than one output might be generated. Other equivalent depictions of the convolution operation might be found below:

$$30 \quad \text{out1}[x, y] = \text{conv1}(I) = \sum_{k=0}^M \sum_{i=0}^N \sum_{j=0}^P w1[k, i, j] \times I[k, x + i, y + j] + K1$$

$$out[c, x, y] = conv(I) = \sum_{k=0}^M \sum_{i=0}^N \sum_{j=0}^P w[c, k, i, j] \times I[k, x + i, y + j] + K[c]$$

In the above equations “c” indicates the channel number. It is equivalent to output number, out[1,x,y] is one output and out[2,x,y] is a second output. Wherein the k is the input number, I[1, x, y] is one input and I[2, x, y] is a second input.

5 The w1, or w describe weights of the convolution operation.

1.11 Leaky_Relu activation function

The leaky relu activation function is depicted in Fig.10. According to the function, if the input is a positive value, the output is equal to the input. If the input (y) is a negative value, the output is equal to a*y. The a is typically (not limited to) a value that is smaller than 1 and greater than 0.

10 Since the multiplier a is smaller than 1, it can be implemented either as a multiplication with a non-integer number, or with a division operation. The multiplier a might be called the negative slope of the leaky relu function.

1.12 Relu activation function

The relu activation function is depicted in Fig.11. According to the function, if the input is a positive value, the output is equal to the input. If the input (y) is a negative value, the output is equal to 0.

1.13 The JPEG AI image coding standard

The existing design utilizes some NN-based image coding methods described mentioned above. Some of the features are described or summarized below. The numbers in parentheses are section
20 numbers for reference.

1. (3.5.20) padding layer

padding layer is denoted as $Padd(H_{in}, W_{in}, d, s_d)$, where H_{in}, W_{in} are height and width of tensor – input to Analysis transform, s_d is stride of proceeding convolution, d is depth of convolution layer in deep learnable encoder. Padding layer receives tensor of size $[C, h_{d-1}, w_{d-1}]$ and
25 outputs tensor of size $[C, s_d h_d, s_d w_d]$, where $h_d = \text{ceil}(h_{d-1}/s_d)$; $w_d = \text{ceil}(w_{d-1}/s_d)$, $h_0 = H_{in}$, $w_0 = W_{in}$. By default padding is performed by replication. Different model of padding can be specified (for example, padding by zeros).

2.(3.5.22) cropping layer

cropping layer is denoted as $Crop(H_{in}, W_{in}, d, s_d)$, where H_{in}, W_{in} are height and width of
30 tensor – output to Synthesis transform, s_d is stride of proceeding transposed convolution, d is depth of convolution layer in deep learnable reconstruction process. Cropping layer outputs a tensor of size $[C, h_{d-1}, w_{d-1}]$, where h_{d-1}, w_{d-1} defined in Table 2. Note, that $h_0 = H_{in}$, $w_0 = W_{in}$. Cropping is performed by discarding redundant elements.

Table 2 Tensor size parameters for primary and secondary components decoding.

	Primary “Y” component	Secondary “UV” component
H_{in}	H	$\text{ceil}(H/(3 - c_{ver}))$
W_{in}	W	$\text{ceil}(W/(3 - c_{hor}))$
C_{in}	1	2
$h_d, d = 0, \dots, 6$	$\text{ceil}(H_{in}/2^d)$	$\text{ceil}(H_{in}/2^{(d<4)?d:d-1})$
$w_d, d = 0, \dots, 6$	$\text{ceil}(W_{in}/2^d)$	$\text{ceil}(W_{in}/2^{(d<4)?d:d-1})$
C	$C_p = 160$	$C_s = 32$
C_3	$96 + 32 * opIdx$	64
$C_2 = C_1$	$64 + 64 * opIdx$	64
C_d	0	160

Table 3 Supported combinations of output picture format and scaling factors

output picture color sampling mode	color sampling mode coded picture	c_{ver}	c_{hor}	s_{ver}	s_{hor}
4:4:4 $H_Y = H_{UV} = H$ $W_Y = W_{UV} = W$	4:4:4	2	2	2	2
	4:2:2	Not applicable			
	4:2:0	1	1	2	2
4:2:2 $H_Y = H_{UV} = H$ $W_Y = W; W_{UV} = W/2$	4:4:4	Not applicable			
	4:2:2	2	1	2	1
	4:2:0	1	1	2	1
4:2:0 $H_Y = H; H_{UV} = H/2$ $W_Y = W; W_{UV} = W/2$	4:4:4	Not applicable			
	4:2:2	Not applicable			
	4:2:0	1	1	1	1

5 3. (8.4) Latent space tiles

Latent space tiles decoding can be enabled for each component independently by flags signalled in picture header (section 9.3). If `tile_enable_Luma` or `tile_enable_Chroma` is true then corresponding component is decoded using latent tiling process illustrated on Fig. 12.

The number and location of tiles are determined by values tile size S_{tile} (equal to `tile_size_Luma` for primary component and `tile_size_Chroma` for secondary component) and tile overlap m_{tile} (`tile_overlap_Luma` for primary component `tile_overlap_Chroma` for secondary component) signalled in picture header (section 9.3). Further, the ratio $r = H_{in}/h_4 = w_{in}/w_4$ between signal domain size and latent tensor size is known, which is $r = 2^4$ for primary and $r = 2^3$ for secondary component.

15 The number of tiles vertically is $K = \text{ceil}((H_{in} - m_{tile})/(S_{tile} - m_{tile}))$ and horizontally

$L = \text{ceil}((W_{in} - m_{tile}) / (S_{tile} - m_{tile}))$, where H_{in}, W_{in} are height and width of output color plane.

H_{in}, W_{in} are hight and width of output color plane.

For $k = 0, \dots, K - 1$ and $l = 0, \dots, L - 1$

5 - Define tiles coordinates and sizes

- $i_0^{(k)} = k \cdot (S_{tile} - m_{tile})$ – vertical dimension tile start in signal domain;
- $i_4^{(k)} = i_0^{(k)} / r$ – vertical dimension tile start in latent space;
- $H_0^{(k)} = \left((i_0^{(k)} + S_{tile}) \leq H_{in} \right) ? S_{tile} : H_{in} - i_0^{(k)}$ – vertical dimension tile size in signal domain;

- 10 ○ $h_4^{(k)} = \text{ceil}(H_0^{(k)} / r)$ – vertical dimension tile size in latent space;
- $j_0^{(l)} = l \cdot (S_{tile} - m_{tile})$ – horizontal dimension tile start in signal domain;
 - $j_4^{(l)} = j_0^{(l)} / r$ – horizontal dimension tile start in latent space;
 - $W_0^{(l)} = \left((j_0^{(l)} + S_{tile}) \leq W_{in} \right) ? S_{tile} : W_{in} - j_0^{(l)}$ – horizontal dimension tile size in signal domain;

- 15 ○ $w_4^{(l)} = \text{ceil}(W_0^{(l)} / r)$ – horizontal dimension tile size in latent space.

- tile latent space tensor

- For $c = 0, \dots, C - 1$, $i = 0, \dots, h_4^{(k)} - 1$ and $j = 0, \dots, w_4^{(l)} - 1$

$$\hat{y}_{k,l}[c, i, j] = \hat{y}[c, i_4^{(k)} + i, j_4^{(l)} + j]$$

- For $c = 0, \dots, C_d - 1$, $i = 0, \dots, h_4^{(k)} - 1$ and $j = 0, \dots, w_4^{(l)} - 1$

20 ▪ $\tilde{y}_{k,l}[c, i, j] = \tilde{y}[c, i_4^{(k)} + i, j_4^{(l)} + j]$

- Synthesis transform for one tile

- Synthesis transform described in section 8.3 with $\hat{y}_{k,l}, \tilde{y}_{k,l}$ of sizes $h_4^{(k)}$ and $w_4^{(l)}$ as inputs and $\hat{x}_{k,l}$ of size $H_0^{(k)}$ and $W_0^{(l)}$ as output.

- Merging reconstructed parts of tensor into one

- 25 ○ $ov_l = (j_0^{(k)} == 0) ? 0 : m_{tile} / 2$ – overlap used on left boundary of the tile;
- $ov_t = (i_0^{(l)} == 0) ? 0 : m_{tile} / 2$ – overlap used on top boundary of the tile;
 - $ov_r = (j_0^{(k)} + W_0^{(l)} >= W_{in}) ? 0 : m_{tile} / 2$ – overlap used on right boundary of the tile;
 - $ov_b = (j_0^{(l)} + H_0^{(k)} >= H_{in}) ? 0 : m_{tile} / 2$ – overlap used on bottom boundary of

the tile;

- $H_1^{(k)} = H_0^{(k)} - \text{ov}_t - \text{ov}_b$ vertical dimension tile size in signal domain without tile overlap;
- $W_1^{(l)} = W_0^{(l)} - \text{ov}_l - \text{ov}_r$ horizontal dimension tile size in signal domain without tile overlap;
- For $c = 0, \dots, C_{in} - 1$, $i = 0, \dots, H_1^{(k)} - 1$ and $j = 0, \dots, W_1^{(l)} - 1$,
 - $\hat{x}[c, i_0^{(k)} + \text{ov}_t + i, j_0^{(l)} + \text{ov}_l + j] = \hat{x}_{k,l}[c, \text{ov}_t + i, \text{ov}_l + j]$.

Tile size location and overlap signalling in Picture Header is described in Annex C.

4. (9.2) Code stream layout

- 10 Code stream structure is depicted in Fig. 13. The code stream is composed of six parts with byte boundary, which are:
1. SOC - Start Of Codestream marker;
 2. PIH (Picture Header marker) followed by picture header (section 9.3);
 3. TOH (Tools Header marker) followed by tools information (section 9.3.1.1);
 - 15 4. SOQ (start of Quality map marker) followed by code stream;
 5. SOZ (Start of Z-stream marker) followed codestream of hyper tensor z, including $\hat{z}_Y$ ("*stream z_Y*" in Fig. 14, which illustrates a general JPEG AI decoder structure) and $\hat{z}_{UV}$ ("*stream z_{UV}*" in Fig. 14);
 6. SORp (Start of Residual stream for primary component marker) followed by codestream of primary component residual, which includes $\hat{r}_Y$ ("*stream y_Y*" in Fig. 14);
 - 20 7. SORs (Start of Residual stream for secondary component marker) followed by codestream of secondary component residual, which includes $\hat{r}_{UV}$ ("*stream y_{UV}*" in Fig. 14);
 8. EOC - End Of Codestream marker.

The overall syntax structure of an image is:

picture() {	Descriptor
SOC	u(16)
picture_header()	
tools_header()	
if (gain_3D_enable_flag)	
q_stream()	
z_stream()	
r_primary_stream()	
r_secondary_stream()	

EOC	u(16)
}	

Each code stream starts with a 16-bit marker. All markers used in this specification are as follows:

Code assignment	Symbol	Description	Mandatory/Optional
0xff80	SOC	Start of codestream	Mandatory
0xff81	EOC	End of codestream	Mandatory
0xff82	PIH	Picture header	Mandatory
0xff83	TOH	Tools header	Optional
0xff84	VUI	Reserved for rendering information	Optional
0xff85	xxx	Reserved	Optional
0xff86	xxx	Reserved	Optional
0xff87	xxx	Reserved	Optional
0xff88	SOZ	Start of z	Mandatory
0xff89	SORp	Start of residual (primary component)	Mandatory
0xff8a	SORs	Start of residual (secondary component)	Mandatory
0xff8b	SOQ	Start of quality map	Optional
0xff8c	xxx	Reserved	Optional
0xff8d	xxx	Reserved	Optional
0xff8e	xxx	Reserved	Optional
0xff8f	xxx	Reserved	Optional

5. (9.3.1) Syntax table

picture_header() {	Descriptor
PIH	u(16)
picture_header_size	u(16)
img_width	u(16)
img_height	u(16)
picture_format	u(2)
bit_depth	u(1)

bit_c_ver	u(1)
bit_c_hor	u(1)
bit_s_ver	u(1)
bit_s_hor	u(1)
color_transform_header()	u(1)
model_header()	
}	

6. (9.3.1.1) Model header

tile_header_Luma(tile_signaling_type) {	Descriptor
tile_enable_Luma	u(1)
if (tile_enable_Luma)	
tile_size_Luma	u(13)
tile_overlap_Luma	u(8)
}	

tile_header_Chroma(tile_signaling_type) {	Descriptor
tile_enable_Chroma	u(1)
tile_size_Chroma	u(13)
tile_overlap_Chroma	u(8)
}	
color_transform_header() {	Descriptor
color_transform_enable	u(1)
if(color_transform_enable)	
for (i=0;i<3;i++)	
for (j=0;j<3;j++)	
color_transform_matrix[i][j]	u(8)
for (i=0;i<3;i++)	
color_transform_offset[i]	u(8)
}	

5

7. (9.3.2) Picture header semantics

Following serves information is signalled:

picture_header_size is the number of bytes in the picture header excluding the first two-byte marker;

img_width plus 64 specifies width of an input picture (from 64 to 65600);

img_height plus 64 specifies height of the input picture (from 64 to 65600);

5 **picture_format** is a data format of the output picture (YUV420 = 0, YUV444 = 1, sRGB = 2, YUV422 = 3);

bit_depth is a bit-depth the output picture (“0” corresponds to 8 and “1” corresponds to 10);

bit_c_ver is one bit value which defines $c_ver = 1 + \text{bit_c_ver}$, c_ver controls internal subsampling mode of the secondary component in vertical direction as defined in Table 3 and Table 2. If

10 bit_c_ver is not present in bit-stream then c_ver is equal to 2.

bit_c_hor is one bit value which defines $c_hor = 1 + \text{bit_c_hor}$, c_hor controls internal subsampling mode of the secondary component in horizontal direction as defined in Table 3 and Table 2. If bit_c_hor is not present in bit-stream then c_hor is equal to 2.

bit_s_ver is one bit value which defines s_ver which is used for align coding subsampling mode
15 of the secondary component and subsampling mode in output picture format in vertical direction as defined in Table 3. Usage of s_ver is described in section 7.6 If bit_c_ver is not present in bit-stream then s_ver is equal to 1.

bit_s_hor is one bit value which defines $s_hor = 1 + \text{bit_s_hor}$, s_hor controls internal subsampling mode of the secondary component in horizontal direction as defined in Table 3 Usage of s_hor is
20 described in section 7.6 If bit_c_ver is not present in bit-stream then s_hor is equal to 1.

scale_comp[2] are vertical and horizontal ratio between primary and secondary components sizes of coded picture $s_ver = \text{scale_comp}[0]$ and $s_hor = \text{scale_comp}[1]$; if not present (res_changer_enable = false) then $s_ver = 2$ and $s_hor = 2$. Allowed values are listed in Table 3.

independent_beta_uv is a flag (false/true) which indicates do the rate control parameter (β) for
25 primary and secondary components are the same.

beta_displacement_log_y – parameter indicating ratio between rate control parameter beta selected by encoder for primary component and one used in the model training.

$\text{betaDisplacementLogY} = \text{beta_displacement_log_y} - 2^{11}$

beta_displacement_log_uv – parameter indicating ratio between rate control parameter beta
30 selected by encoder for secondary component and one used in the model training.

$\text{betaDisplacementLogUV} = \text{beta_displacement_log_uv} - 2^{11}$

model_id is an identifier of pre-stored checkpoint with model’s weights, $\text{model_id} = 0, 1, 2, 3$ or 4.

opIdx is an identifier for operation point, 0 means “base”, 1 means “high” operation point.

tile_enable_Luma and **tile_enable_Chroma** are enable flags for tiling of primary and secondary components.

tile_size_Luma and **tile_size_Chroma** are size of tiles for primary and secondary components.

tile_overlap_Luma and **tile_overlap_Chroma** are sizes of tiles overlapping areas for primary and secondary components.

cube_group_flag is 1-bit unsigned integer. **cube_group_flag**=0 indicates no **cube_flag** is signalled in one group, and cube flags of one group are set to be 1. **cube_group_flag**=1 indicates cube flags of one group are signaled.

cube_luma_flag is 1D array of size $((h_{4,Y} + 7) \gg 3) \cdot ((w_{4,Y} + 7) \gg 3)$, which contains cube flags for primary component. 1 indicates Skip Mode is applied to one cube of residual tensor of primary component $\hat{r}_Y$. 0 indicates Skip Mode is disable for one cube of residual tensor of primary component $\hat{r}_Y$.

cube_chroma_flag is 1D array of size $((h_{4,UV} + 7) \gg 3) \cdot ((w_{4,UV} + 7) \gg 3)$, which contains cube flags for secondary component. 1 indicates Skip Mode is applied to one cube of residual tensor of secondary component $\hat{r}_{UV}$. 0 indicates Skip Mode is disable for one cube of residual tensor of secondary component $\hat{r}_{UV}$.

color_transform_enable is an enable flag for color conversion module.

color_transform_matrix[i][j] is a matrix of color conversion. If not present (**color_transform_enable** is false) then default ITU-R BT.709 colour transform is used (refer to section 7.8).

color_transform_offset[i] is an offset for color transformation. If not present (**color_transform_enable** is false) then default ITU-R BT.709 colour transform is used (refer to section 7.8).

8. (9.5.1.1) Syntax table of hyper tensor

z_stream() {	Descriptor
SOZ	u(16)
z_stream_size	u(24)
num_threads_z	u(8)
for (i =1; i < num_threads_z; ++i) {	
thread_offsets_z[i]	u(24)
}	
for (i =0; i < num_threads_z; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_z, thread_offsets_z, z_stream_size)	

}	
for (ch =0; ch < z_primary_ch; ++ch) {	
for (i =0; i < num_threads_z; ++i) {	
decode_z_data(sub_stream[i], z_primary[ch][], ch, z_primary_width, z_primary_height, i, num_threads_z)	
}	
}	
for (ch =0; ch < z_secondary_ch; ++ch) {	
for (i =0; i < num_threads_z; ++i) {	
decode_z_data(sub_stream[i], z_secondary[ch][], ch, z_secondary_width, z_secondary_height, i, num_threads_z)	
}	
}	
}	

9. (9.5.2) Quality map information decoder

The input of this process is

- Codestream for quality_map;
- 5 - Indicators for sigma tables which are decoded in picture header:
 - o quality_map_entropy_index_y for primary component.
 - o quality_map_entropy_index_uv for secondary component.

The output of this process is

- quality_map_delta_Y is an array of size $[w_{4Y}, h_{4Y}]$ with information used for deriving
10 scaling factor (section 12.2) for primary component residual tensor.
- quality_map_delta_UV is an array of size $[w_{4UV}, h_{4UV}]$ with information used for
deriving scaling factor (section 12.2) for secondary component residual tensor.

Here sizes $[w_{4Y}, h_{4Y}]$ and $[w_{4UV}, h_{4UV}]$ are defined in Table 2.

- q_primary_sigma_idx is a 1D array of size $h_{4Y} \times w_{4Y}$, and each element in this array is equal to
15 q_sigma_idx[quality_map_entropy_index_y], where the q_sigma_idx[k] is derived according to
Table 4.

q_secondary_sigma_idx is a 1D array of size $h_{4UV} \times w_{4UV}$, and each element in this array is
equal to q_sigma_idx[quality_map_entropy_index_uv].

q_primary is a 1D array of size $h_{4Y} \times w_{4Y}$, and quality_map_delta_Y is determined by:

- 20 - quality_map_delta_Y[i, j] = q_primary[i $\times$ w_{4Y} + j], for j = 0.. w_{4Y} - 1, and i = 0.. h_{4Y} - 1.

$q_secondary$ is a 1D array of size $h_{4UV} \times w_{4UV}$, and $quality_map_delta_UV$ is determined by

- $quality_map_delta_UV[i, j] = q_secondary[i \times w_{4UV} + j]$, for $j = 0..w_{4UV} - 1$, and $i = 0..h_{4UV} - 1$.

- **Table 4 Determining q_sigma_Idx for quality map delta decoding**

k	0	1	2	3	4	5	6	7
$q_sigma_Idx[k]$	0	4	5	6	8	10	14	18

5

10. (9.5.2.1) Syntax table of quality map information tensor

$q_stream()$ {	Descriptor
SOQ	u(16)
q_stream_size	u(24)
$num_threads_q$	u(8)
for ($i=1; i < num_threads_q; ++i$) {	
$thread_offsets_q[i]$	u(24)
}	
for ($i=0; i < num_threads_q; ++i$) {	
$sub_stream[i]=sub_stream_init(i, num_threads_q, thread_offsets_q,$ $q_stream_size)$	
}	
for ($i=0; i < num_threads_q; ++i$) {	
$decode_r_data(sub_stream[i], q_primary[], q_primary_sigma_Idx[], i,$ $num_threads_q)$	
}	
for ($i=0; i < num_threads_q; ++i$) {	
$decode_r_data(sub_stream[i], q_secondary[], q_secondary_sigma_Idx[], i,$ $num_threads_q)$	
}	
}	

11. (9.5.3) Primary residual stream decoder

The input of this process is

- 10 - Primary component residual codestream;
- $mask_skip[C_p, h_{4Y}, w_{4Y}]$ generated by **Skip Mask generation** (section 13.3.2) for primary component;

- sigma_Idx_primary – tensor of size $[C_p, h_{4Y}, w_{4Y}]$ which is the output of **Sigma quantization** (section 10.8) for primary component.

The output of this process is

- r_primary – one dimensional array $\{s\}$ of residual tensor elements which is an input of

5 decoder skip process (section 13.3.4) for primary component.

Here sizes C_p , w_{4Y} , h_{4Y} are defined in Table 2. A one dimensional array

$\text{sigma_Idx_primary_1D}$ is determined through following steps:

- Set $k = 0$
- For $c = 0..C_p - 1$,
- 10 - For $i = 0..h_{4Y} - 1$,
- For $j = 0..w_{4Y} - 1$
 - if $\text{mask_skip}[c, i, j]$ is equal to True, $\text{sigma_Idx_primary_1D}[k]$ is set to equal to $\text{sigma_Idx_primary}[c, i, j]$ and increase k by 1.

15 12. (9.5.3.1) Syntax table of primary residual tensor

$\text{r_primary_stream}()$ {	Descriptor
SOR	u(16)
$\text{r_primary_stream_size}$	u(32)
$\text{num_threads_r_primary}$	u(8)
for ($i = 1$; $i < \text{num_threads_r_primary}$; $++i$) {	
$\text{thread_offsets_r_primary}[i]$	u(32)
}	
for ($i = 0$; $i < \text{num_threads_r_primary}$; $++i$) {	
$\text{sub_stream}[i] = \text{sub_stream_init}(i, \text{num_threads_r_primary},$ $\text{thread_offsets_r_primary}, \text{r_primary_stream_size})$	
$\text{decode_r_data}(\text{sub_stream}[i], \text{r_primary}[], \text{sigma_Idx_primary}[], i,$ $\text{num_threads_r_primary})$	
}	
}	

13. (9.5.4) Secondary residual stream decoder

The input of this process is

- Codestream for secondary component residual;

- $\text{mask_skip}[C_s, h_{4UV}, w_{4UV}]$ generated by **Skip Mask generation** (section 13.3.2) for secondary component;
- $\text{sigma_Idx_secondary}$ – tensor of size $[C_s, h_{4UV}, w_{4UV}]$ which is the output of **Sigma quantization** (section 10.8) for secondary component.

5 The output of this process is

- $r_secondary$ – one dimensional array $\{s\}$ of residual tensor elements which is an input of decoder skip process (section 13.3.4) for secondary component.

Here sizes C_s, h_{4UV}, w_{4UV} are defined in Table 2. A one dimensional array

$\text{sigma_Idx_secondary_1D}$ is determined through following steps:

- 10 – Set $k = 0$
- For $c = 0..C_p - 1$,
 - For $i = 0..h_{4Y} - 1$,
 - For $j = 0..w_{4Y} - 1$
 - if $\text{mask_skip}[c, i, j]$ is equal to True, $\text{sigma_Idx_secondary_1D}[k]$ is set
- 15 to equal to $\text{sigma_Idx_secondary}[c, i, j]$ and increase k by 1.

14. (9.5.4.1) Syntax table of secondary residual tensor

$r_secondary_stream() \{$	Descriptor
SOR	u(16)
$r_secondary_stream_size$	u(24)
$num_threads_r_secondary$	u(8)
for ($i=1; i < num_threads_r_secondary; ++i$) {	
$thread_offsets_r_secondary[i]$	u(24)
}	
for ($i=0; i < num_threads_r_secondary; ++i$) {	
$sub_stream[i]=sub_stream_init(i, num_threads_r_secondary,$ $thread_offsets_r_secondary, r_secondary_stream_size)$	
$decode_r_data(sub_stream[i], r_secondary[], sigma_Idx_secondary[], i,$ $num_threads_r_secondary)$	
}	
}	

15. (10.3) Hyper Scale Decoder

20 Hyper scale decoder net is depicted in Fig. 15.

The input of hyper scale decoder is

- $\hat{z}[C, h_6, w_6]$ reconstructed hyper tensor,
- sizes of input/output tensor H_{in}, W_{in} ,
- operation point indicator $opIdx$,
- 5 – model parameters for **Hyper Scale Decoder Net** defined by pair $(modelIdx, opIdx)$,
all multiplier parameters in those models are 8-bits integer.

The output of hyper scale decoder is standard deviation logarithm tensor $I_\sigma [C, h_4, w_4]$ with integer values in a range $0 \leq I_\sigma < ((N_\sigma - 1) \ll \text{sigmaPrecision})$, where $N_\sigma, \text{sigmaPrecision}$ are defined in section 0. Sizes of those tensors for primary and secondary components are listed in
10 Table 2.

All operation in scalable hyper decoder are integer, an accumulator in all computations is within 32 bits integer diapason, multipliers in model parameters are quantized to 8-bits integer. This guarantees bit-exact behaviour of this neural network module. Hyper scale decoder uses special type of operations quantized convolutions. For each quantized convolution in the process
15 the set of clipping values $\{d_k\}$ and de-scaling shifts parameters $\{p_k\}$ ($1 \leq k \leq 3$) are specified for each channel. All clipping values in quantized convolutions are $d_k = 2^7$. De-scaling shifts $\{p_k\}$ are provided in section 15.6.

NOTE – the magnitude of integerized weights in quantized model doesn't exceed 2^7 , shift and clipping value combination ensures the register of quantized convolution is within 32-bits.

20 The sequence of operations in this module doesn't depend on operation point indicator ($opIdx$), but model parameters are different for base and high operation point.

Hyper scale decoder starts with quantized convolution kernel size 1×1 , stride 1, followed by rectified linear unit. Then there is a stride 1 quantized convolution with kernel size 3×3 , followed by rectified linear unit. The next stride 1 quantized convolution (kernel size
25 1×1) increases the number of channels to $16C$. It is followed by pixel shuffle (stride 4), which brings number of channels back to C . The cropping layer (stride 4, depth 5) ensures the size of output tensor is $[C, h_4, w_4]$. The process concluded with abs operation.

Model with weights are storied in electronic attachment, in format specied in section 4.8 and locates in $\langle \text{oper_point} \rangle / \text{model_} \langle \text{MID} \rangle / \langle \text{COMP} \rangle / \text{hyper_scale_decoder.onnx}$, where

30 $\langle \text{oper_point} \rangle$ is "base" or "high", $\langle \text{MID} \rangle$ is an integer from 0 to 5, $\langle \text{COMP} \rangle$ is "primary" or "secondary".

16. (11.2) Hyper Decoder

The learning-based hyper decoder consists of two independent pipe-lines with identical

neural network architecture, except input size and number of channels.

The input of this process is

- $\hat{z} [C, h_6, w_6]$ reconstructed hyper latent tensor,
- model parameters for **Hyper Decoder Net** defined by (*modelIdx*),
- 5 – operation point indicator *opIdx*.

The output of this process is

- $p[C_p, h_4, w_4]$ is explicit prediction (part of prediction tensor derived from explicitly signalled information), with channels size $C_p = (opIdx + 1)C$.

Hyper decoder process is depicted in Fig. 16.

- 10 Hyper decoder starts stride 1 convolution with kernel size 1×1 , followed by inverse convolution (stride 2, kernel size 4×4), cropping layer (depth 6) and leaky rectified linear unit. Next step is stride 1 convolution with kernel size 3×3 , followed by inverse convolution (stride 2, kernel size 4×4), cropping layer (depth 5) and leaky rectified linear unit. Number of channels kept unchanged till this point (equal to number of channels C of input tensor). Hyper decoder concluded
- 15 by stride 1 convolution with kernel size 3×3 which increases number of channels to $2C$ for high operation point and keeps number of channels unchanged for base operation point followed by leaky rectified linear unit.

Model with weights are stored in electronic attachment, in format specified in section 4.8 and located in <oper_point>/model_<MID>/<COMP>/hyper_decoder.onnx, where <oper_point> is

- 20 “base” or “high”, <MID> is an integer from 0 to 5, <COMP> is “primary” or “secondary”.

17. (11.3) Latent tensor reconstruction process

The input of this process is

- operation point indicator *opIdx*,
- $\hat{r} [C, h_4, w_4]$ reconstructed residual tensor, which is an output of SKIP Model process (13.3.4),
- 25 – $p [C_p, h_4, w_4]$ explicit prediction, with channels size $C_p = (opIdx + 1)C$, which is an output of Hyper Decoder (11.2),

The output of this process is

- $\hat{y}' [C, h_4, w_4]$ reconstructed latent tensor.

- 30 The process is as follows.

If *opIdx* == 0 (base operation point) then multi-stage context modelling process is bypassed,

- explicit prediction is added to residual $\hat{y}' = \hat{r} + p [0: C - 1, h_4, w_4]$:

If $opIdx == 1$ (high operation point) then:

- **Multi-stage Context Modelling** process (section 11.4) is used.

18. (11.4) Multistage Context Modelling

The input of this process is

- 5 – $\hat{r}[C, h_4, w_4]$ reconstructed residual tensor, which is an output of SKIP Model process (13.3.4),
- $p[2C, h_4, w_4]$ explicit prediction, which is an output of Hyper Decoder (11.2),
- Eight MCM_k , $k=0, \dots, 7$ models with parameters defined by $(modelIdx, k)$,

The output of this process is

- 10 – $\hat{y}'[C, h_4, w_4]$ reconstructed latent tensor.

The process consists of following steps:

- **padding layer** (depth 5, stride 2) and **down-shuffle** (11.5.1) $M=2$ of explicit prediction tensor $p[2C, h_4, w_4]$ to $\check{p}[8C, h_5, w_5]$ re-shaped prediction tensor,
- **padding layer** (depth 5, stride 2) and **down-shuffle** (11.5.1) $M=1$ of reconstructed residual $\hat{r}[C, h_4, w_4]$ to $\check{r}[4C, h_5, w_5]$ re-shaped residual tensor,
- 15 – split $\check{p}[8C, h_5, w_5]$ into four parts $\check{p}_l = \check{p}[2lC : 2(l+1)C - 1, h_5, w_5]$, $l = 0, \dots, 3$ (each part consists of $2C$ out of $8C$ channels),
- split $\check{r}[4C, h_5, w_5]$ into eight parts $\check{r}_k = \check{r}[kC/2 : (k+1)C/2 - 1, h_5, w_5]$, $k = 0, \dots, 7$ (each part consists of $C/2$ out of $4C$ channels),
- 20 – For $k=0, \dots, 3$
 - **MCM(k)** process which
 - takes as an input
 - $\{\check{y}_m\}$, $m = 0, \dots, k-1$ previously reconstructed parts of re-shaped latent space tensor,
 - 25 • $\check{r}_k$ – collocated part of reconstructed residual tensor,
 - $\check{p}_{k\%4}$ – part of re-shaped explicit prediction tensor,
 - outputs
 - produces $\check{y}_k = \check{y}[kC/2 : (k+1)C/2 - 1, h_5, w_5]$.
 - **Channel net** process (11.5.4) over $\check{y}[0 : (2C-1), h_5, w_5]$ tensor.
 - 30 – For $k=3, \dots, 7$
 - **MCM(k)** process which
 - takes as an input
 - $\{\check{y}_m\}$, $m = 0, \dots, k-1$ previously reconstructed parts of re-shaped

latent space tensor,

- $\ddot{r}_k$ – collocated part of reconstructed residual tensor,
- $\ddot{p}_{k\%4}$ – part of re-shaped explicit prediction tensor.

▪ outputs

- produces $\ddot{y}_k = \ddot{y}[kC/2:(k+1)C/2-1, h_5, w_5]$.

- **up-shuffle** (11.5.2) $M=2$ and **cropping layer** (depth 5, stride 2) $\ddot{y}[4C, h_5, w_5]$ to $\hat{y}[C, h_4, w_4]$.
- **up-shuffle** (11.5.2) $M=2$ and **cropping layer** (depth 5, stride 2) $\ddot{\mu}[4C, h_5, w_5]$ to $\mu[C, h_4, w_4]$ (to be further used in **LSBS process** 13.4.2).

10 Multi-stage context modelling process is depicted in Fig. 17. This is a recurrent process: later stages uses previously obtained elements of output tensor as an input. Data flow which corresponds usage of previous stages output is marked with arrows in Fig. 17.

Models with weights are stored in electronic attachment, in format specified in section 4.8 and located in `high/model_<MID>/<COMP>/MCM/stage<N>.onnx`, where `<MID>` is an integer

15 from 0 to 5, `<COMP>` is “primary” or “secondary”, `<N>` is a number of MCM’s stage from 0 to 7.

19. (13.3.4) Decoder side SKIP operation

At the decoder, the inputs of skip mode process are

- 20 – 1D array $s[\text{num_res_elements}]$ after decoding by me-tANS (section 9.5.2) to from the “stream – y”,
- $\text{mask_skip}[C, h_4, w_4]$.

The output of this process is

- the residual tensor $\hat{r}[C, h_4, w_4]$.

25 The output of the lossless decoding process is a 1D array $\{s_k\}$, whose size is equal to the total number of “1”s in the $\text{mask_skip}[C, h_4, w_4]$ tensor.

In other words, the $\text{mask_skip}[C, h_4, w_4]$ 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.

30 The process of residual skip mode at the decoder is as follows:

- Dimensions $[C, h_4, w_4]$ are set equal to number of channels, height and width of the sigma tensor σ (Table 2).
- Tensors $\hat{r}[C, h_4, w_4]$ is initialized to be equal to all zeros.

- The counter $k = 0$.
- The following ordered steps are applied:
- For $c = 0..C - 1$,

- For $i = 0..h_4 - 1$

5 ▪ For $j = 0..w_4 - 1$

if mask_skip $[c, i, j]$ is equal to True, $\hat{r}[c, i, j] = \text{mask_skip}[c, i, j]$
 $\cdot (s[k] - 2^{16-1} + 1)$, and increase k by 1;

▪ otherwise, $\hat{r}[c, i, j] = 0$.

20. (14.1.1) Inter channel correlation information filter tiling and output selection

10 process

The ICCI filter processes the input in tiles using the same method described in Section 8.4. The tile size `icci_tile_size` and tile overlap `icci_tile_overlap` are same for the primary (Luma) and secondary (Chroma) components and signalled in picture header (Section 9.3). The ratio r between input domain size and output tensor size is fixed to 1 for each component. The minimum

15 tile size is limited to 176 for the stable computation of the model selection explained next.

In total, 10 ICCI filters are included into the design. In each tile *tileID* for each color component $comp=0..2$ (“0” – “y”, “1” – “u”, “2”- “v”) the filter is selected based on `icci_model_idx[comp][tileID]` value which is coded in picture header (Section 9.3). In case `icci_model_idx = 0`, the ICCI processing is bypassed.

20 21. (14.2) Adaptive upsampler

This section details the primary component guided adaptive upsampler process. This process provides enhancement of secondary components (colour information planes) of image utilising information from primary component.

This process is enabled if `EFE_upsampler_enabled_flag` is true.

25 The input of this process is

- $\hat{x}_Y [1, H_Y, W_Y]$ (output of synthesis transfor for primary component),
- $\hat{x}_{UV} [2, H_{inUV}, W_{inUV}]$ (output of synthesis transfor for secondary component),

The output of this process is

- enhanced secondary component $\hat{x}'_{UV} [2, H_{UV}, W_{UV}]$ which goes to the ICCI filter block
- 30 (section 14.1) and $\hat{x}''_{UV} [2, H_{UV}, W_{UV}]$ which goes to the non-linear filter block (section 14.3).

Fig. 18 illustrates an example implementation of primary component guided adaptive up-sampling filter.

If `EFE_upsampler_enabled_flag` is equal to 0, the $\hat{x}'_{UV}[2, H, W]$ is up-sampled by bi-cubic interpolation as described in section 7.6. Otherwise the following ordered steps are performed:

- The parsing process according to parsing table in **Section 9.4.1** is invoked to obtain W_{1A} , W_{1B} , W_{4A} , W_{4B} and B_1 .
- 5 – Tiling process is as described in section 14.2.2 is invoked with parsed syntax elements as inputs and `Tile1` tensor as output.
- Parameter update process as specified in section **Section 14.2.1**. is invoked with W_{4A} and W_{1A} as inputs and modified W_{4A} and W_{1A} as outputs.
- $B_1[2]$ is vector is subtracted channelwise from $\hat{x}'_{UV}[2, H_{inUV}, W_{inUV}]$.
- 10 – $U\left[2 \cdot scale_{ver} \cdot scale_{hor}, \frac{H+1}{2}, \frac{W+1}{2}\right]$ and $Y\left[4, \frac{H+1}{2}, \frac{W+1}{2}\right]$ are set equal to `pixelUnshuffle( $\hat{x}'_{UV}, scale_{ver}, scale_{hor}$ )` and `pixelUnshuffle( $\hat{x}'_Y, 2, 2$ )` respectively.
- For $x = 0..(W+1)/2-1$, $y = 0...(H+1)/2-1$, $k = 0..1$, $i = 0..o_{ver}-1$, $j = 0..o_{hor}-1$ the following is performed:
 - $ch = 4k + 2i + j$
 - 15 – $ch_{out} = k \cdot o_{ver} \cdot o_{hor} + o_{hor} \cdot i + j$
 - $ch_{in} = k \cdot scale_{ver} \cdot scale_{hor} + scale_{hor} \cdot (i \% scale_{ver}) + (j \% scale_{hor})$
 - $t_{idx} = Tile1[k, y, x]$
 - $U[ch_{out}, y, x] = U[ch_{in}, y, x] \star W_{4A}[t_{idx}, ch] + Y[2i + j, y, x] \star W_{4B}[t_{idx}, k] + B_1[k]$
 - 20 – $U[ch_{out}, y, x] = U[ch_{in}, y, x] \star W_{1A}[0, ch] + Y[2i + j, y, x] \star W_{1B}[0, k] + B_1[k]$
 - $\hat{x}'_{UV}[2, H_{UV}, W_{UV}]$ and $\hat{x}''_{UV}[2, H_{UV}, W_{UV}]$ are set equal to `pixelshuffle( $U', o_{ver}, o_{hor}$ )` and `pixelshuffle( $U'', o_{ver}, o_{hor}$ )` respectively.

22. (14.4.1) LEF general process

- 25 LEF process receives primary variance tensor $\hat{\sigma}_Y[C_p, h_{4,Y}, w_{4,Y}]$ and reconstructed primary $\hat{x}'_Y[1, H, W]$ as inputs. The output of LEF process is edge enhanced modified $\hat{x}'_Y[1, H, W]$.

The following ordered steps are performed:

- Picture header parsing process is invoked as described in section **Section 9.3**. to obtain `targetBppIdx`,
- 30 – `chIdx` is set equal to `reference_channel_idx_table[targetBppIdx]`,

- The $\hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y]$ tensor is obtained as follows, for $i = 0..H_Y - 1, j = 0..W_Y - 1$,
 - $\hat{\sigma}_{Y_channel}^1[0, i, j] = \hat{\sigma}_Y[chIdx, floor(i \cdot h_{4,Y} \div H_Y), floor(i \cdot w_{4,Y} \div W_Y)]$,
 - $thr[3] = luma_edge_filter_thr_list_table[targetBppIdx]$,
 - $intensity[4] = luma_edge_filter_intensity_list[targetBppIdx]$,
- 5 $\hat{x}'_Y = \hat{x}'_Y \div 255$,
- $\hat{x}_{blur} = \hat{x}'_Y * kernel$,
 - $\hat{x}_{edge} = \hat{x}'_Y - \hat{x}_{blur}$,
 - $\hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y] = \hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y] \div 8190$,
 - For $i = 0..H_Y-1, j = 0..W_Y-1$;
- 10 $tIdx = \begin{cases} 0, & \hat{\sigma}_{Y_channel}^1[0, i, j] < thr[0] \\ 1, & thr[0] \leq \hat{\sigma}_{Y_channel}^1[0, i, j] < thr[1] \\ 2, & thr[1] \leq \hat{\sigma}_{Y_channel}^1[0, i, j] < thr[2] \\ 3, & thr[2] \leq \hat{\sigma}_{Y_channel}^1[0, i, j] \end{cases}$
- $\hat{Y}[i, j] = \hat{x}_{edge}[0, i, j] * intensity[tIdx] + \hat{x}_{blur}[0, i, j]$,
 - $\hat{Y}$ is modified as follows, for $i = 0..H_Y-1, j = 0..W_Y-1$:
 - $\hat{Y}[i, j] = clip(0, 1, \hat{Y}[i, j]) \cdot 255$.

In this process zero padding is used when the indices of a tensor exceeds the tensor

15 boundaries.

The weights tensor $kernel$ is defined as follows:

$$kernel = \begin{bmatrix} 1 \div 13 & 1 \div 13 & 1 \div 13 \\ 1 \div 13 & 5 \div 13 & 1 \div 13 \\ 1 \div 13 & 1 \div 13 & 1 \div 13 \end{bmatrix}$$

The $reference_channel_idx_table$ is defined as follows:

$$reference_channel_idx_table = [23, 23, 65, 65, 65]$$

20 The $luma_edge_filter_intensity_list$ is as follows:

$luma_edge_filter_intensity_list[X][Y]$	Y=0	Y=1	Y=2	Y=3
X=0	1.00	1.10	1.25	1.40
X=1	1.00	1.08	1.12	1.16
X=2	1.00	1.05	1.06	1.07
X=3	1.00	1.02	1.03	1.04
X=4	1.00	1.01	1.02	1.03

The $luma_edge_filter_thr_list_table$ is defined as follows:

<i>luma_edge_filter_thr_list_table</i> [X][Y]	Y=0	Y=1	Y=2
X=0	0.3	2.0	6.0
X=1	0.4	3.0	7.0
X=2	0.5	6.0	10.0
X=3	0.6	7.0	12.0
X=4	0.7	8.0	14.0

3. Problems

The existing design has the following problems:

- 1) Per the current codestream structure, in an JPEG AI codestream containing a coded picture, a type-based bits organization solution is used. That is, all coded bits of the hyper tensor (denoted as 1st bit type) for the entire picture precede all coded bits of the primary component residual (denoted as 2nd bit type) for the entire picture, and all coded bits of the primary component residual for the entire picture precede all coded bits of the second component residual (denoted as 3rd bit type) for the entire picture. Therefore, even when the picture is split into multiple regions by having `num_threads_z`, `num_threads_r_primary`, and `num_threads_r_secondary` greater than 1, in order to decode only one region, the decoder still needs to parse and/or decode the different types of coded bits of all previous regions in raster-scan order. Furthermore, due to that there is no JPEG marker between the coded bits of different regions, it is not easy for the systems layer to encapsulate the coded bits of different regions into different data structures or data units for region-based transmission and/or processing. An example of such systems-layer data structure or data unit is a file format sub-sample. Another example is a file format sample in a track or item carrying only a subset of the regions. Yet another example is a real-time transport protocol (RTP) packet.
- 2) Currently it is possible for `num_threads_z`, `num_threads_r_primary`, and `num_threads_r_secondary` to have different values, which seems not working at all.
- 3) Currently, it is possible to partition a picture into multiple horizontally-split regions by having `num_threads_z`, `num_threads_r_primary`, and `num_threads_r_secondary` greater than 1. However, it is not possible to partition a picture into multiple vertically-split regions or into multiple horizontally-and-vertically-split regions.

A use case for partitioning a picture into multiple vertically-split regions is a single-picture to cover a multi-page document, which is often used nowadays in document sharing in social media, wherein the picture has a normal height but a very large value of width, and in this case, the picture

is rendered piece by piece, each piece corresponding to one page of the original multi-page document, and the user turns pages back and forth by sliding on the screen to the right and to the left, respectively.

A use case for partitioning a picture into multiple vertically-split regions or into multiple horizontally-and-vertically-split regions is virtual reality or 360 degree image, wherein it is ideal for the client to be able to only receive and decode the minimum set of regions that fully covers the current field of view of the user. Similarly, this capability can also be used for traditional region of interest (ROI) coding, which enables the client to be able to only receive and decode the minimum set of regions that fully covers the ROI of the user.

4) Currently, it is possible to partition a picture into multiple regions, and at the same time enable the correct decoding of a subset of each region independently from other regions by having tiles being subsets of regions and alignment of tiling between luma and chroma. However, there lacks a high-level indication of this regional access capability, e.g., in the picture header.

5) Currently, it is possible to partition a picture into multiple regions, and at the same time enable the correct decoding of a subset of each region independently from other regions. However, it is not possible to enable correct decoding of the entire regions.

4. Detailed Solutions

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

1) It is proposed that a subset (such as a subpicture, or a tile, or a slice, or a region) of samples in a picture can be reconstructed independently from samples in the picture out of the subset, with a NN-based decoder such as a JPEG-AI decoder.

2) To solve problems 1 and 2, one or more of the following methods apply:

a. Instead of using the current type-based bits organization solution for a whole picture, it is proposed to restrict the type-based bits organization solution to be region-based instead of picture-based.

i. Alternatively, furthermore, the bits of all types for a first region are coded firstly, followed by the bits of a second region.

ii. Alternatively, furthermore, the bits for a first region are organized following the type-based solution, that is, bits for one type in the first region are coded firstly, followed by bits for another type in the first region.

- iii. Alternatively, furthermore, an indicator (e.g., 1-bit flag or a starting code or an ending code) may be signalled after all bits of all types for a first region before coding the 2nd region.
- iv. Alternatively, furthermore, an indicator (e.g., 1-bit flag or a starting code or an ending code) may be signalled before all bits of all types for a region.
- v. Alternatively, furthermore, an indicator (e.g., 1-bit flag or a starting code or an ending code) may be signalled before/after all bits of one type for a region.
- b. Coded bits of different types (e.g., the hyper tensor, primary component residual, and secondary component residual) for the same region are placed together.
- i. For example, coded bits of different types for the same region are not separated by a coded bit of any other region.
- ii. Additionally, one or multiple indications are signalled to control whether such a coded bit placement is applied or not.
- c. A JPEG marker is placed at the beginning of coded bits for at least one region.
- i. A JPEG marker is placed at the beginning of coded bits for each region.
- ii. In one example, for a region, a JPEG marker is placed at the beginning of each type (hyper tensor, primary component residual, or secondary component residual) of coded bits.
- iii. In another example, for a region, a JPEG marker is placed at the beginning the hyper tensor coded bits, but not at the beginning of the coded bits of the other types (i.e., primary component residual and secondary component residual).
- d. A first number of regions is applied on the hyper tensor coded bits, a second number of regions is applied on the primary component residual coded bits, and a third number of eregions is applied on the secondary component residual coded bits, and the numbers may satisfy:
- i. It may be required that the first number should be equal to the second number.
- ii. It may be required that the first number should be equal to the third number.
- iii. It may be required that the second number should be equal to the third number.
- iv. A single indication may be signaled to indicate the number of regions.
1. The first number, and/or the second number and/or the third number

may be set equal to the signaled number of regions, without being signaled individually.

3) To solve problem 3, one or more of the following NN-based image and/or video coding methods apply:

- 5 a. Enable partitioning a picture into multiple vertically-split regions.
 - i. E.g. the number of vertical splits is indicated.
 - ii. In one example, the number of vertical splits minus 1 is signalled.
 - iii. In one example, the number of vertical splits is signalled and the value is constrained to be greater than 0.
- 10 b. Enable partitioning a picture into multiple horizontally-and-vertically-split regions.
 - i. E.g. the number of vertical splits and/or the number of horizontal splits are indicated.
 - ii. In one example, the number of vertical splits minus 1 and the number of horizontal splits minus 1 are signalled.
 - 15 iii. In one example, the number of vertical splits and the number of horizontal splits are signalled and the values are both constrained to be greater than 0.
- c. Additionally, the maximum number of vertically-split and/or horizontally-split regions is constrained.
 - i. In one example, the maximum number of regions is equal to a positive integer N, which may depend on the profile and level.
 - 20 d. Additionally, the position of vertically-split and/or horizontally-split regions is constrained.
 - i. In one example, the upper-left luma position of a region shall be located at $(2^X, 2^Y)$ relative to the upper-left luma position of the picture, where X and Y are non-negative integers.
 - 25 1. In one example, both X and Y are equal to 4.
 2. In one example, both X and Y are equal to 5.
 3. In one example, both X and Y are equal to 6.
- e. Alternatively, the minimal number of luma samples and/or chroma samples contained in one region is constrained.
 - i. In one example, the number of luma samples in one region shall be no smaller than 2.
 - 30 ii. In one example, the number of luma samples in one region shall be no smaller than 4.

iii. In one example, the number of luma samples in one region shall be no smaller than 16.

iv. In one example, the number of luma samples in one region shall be no smaller than 64.

5 f. Alternatively, the maximum number of luma samples and/or chroma samples contained in one region is constrained.

i. In one example, the number of luma samples in one region shall be smaller than a positive integer N, which may depend on the profile and level.

10 4) To solve problem 4, an indication of regional access capability is signalled in a bitstream coded by an NN-based coder, e.g., to an NN-based decoder such as a JPEG-AI decoder, e.g., in the picture header.

a. In one example, the indication indicates that it is possible to decode a region independently from other regions.

15 i. In one example, the indication may indicate that it is possible to decode each region independently from other regions.

b. In one example, the indication indicates that it is possible to correctly decode a region independently from other regions.

i. In one example, the indication may indicate that it is possible to correctly decode each region independently from other regions.

20 c. In one example, when it indicated that regional access is enabled, one or more of the following constraints are imposed:

i. Each luma or chroma sample on a region boundary shall also be on a tile boundary.

25 ii. Each tile is a subset of a region. This implies that tile width is less than or equal to region width, and tile height is less than or equal to region height.

iii. Tiles for luma and for chroma are aligned. This implies that $(\text{tile_size_Luma_ver} + s_{\text{ver}} - 1) / s_{\text{ver}}$ shall be equal to $\text{tile_size_Chroma_ver}$, and $(\text{tile_size_Luma_hor} + s_{\text{hor}} - 1) / s_{\text{hor}}$ shall be equal to $\text{tile_size_Chroma_hor}$.

30 5) To solve problem 5, one or more of the following methods apply:

a. In one example, an indication is signalled for a region to indicate whether the entire region can be correctly decoded independently from other regions.

i. In one example, an indication may be signaled for each region individually.

b. In one example, an indication is signalled for a region to indicate whether the entire

region can be decoded independently from other regions.

- i. In one example, an indication may be signaled for each region individually.
- c. In one example, for any particular region, when it is indicated that the entire region can be correctly decoded independently from other regions, regardless of the tile overlap value signalled for the picture, zero overlap is applied when the boundaries of the region are processed in the decoding process.

- i. In one example, alternatively, for any particular region, when it is indicated that the entire region can be correctly decoded independently from other regions, in the decoding process the boundaries of the region are handled in the same manner as picture boundaries.

- d. For a particular region, when it is indicated that the entire region can be decoded independently from other regions, regardless of the tile overlap value signalled for the picture, zero overlap is applied when at least one boundary of the region is processed in the decoding process.

- i. In one example, alternatively, for any particular region, when it is indicated that the entire region can be correctly decoded independently from other regions, in the decoding process at least one boundary of the region are handled in the same manner as picture boundaries.

- e. For item 5.c., 5.c.i, 5.d, or 5.d.i, one or more of the following apply:

- i. Additionally or alternatively, padding might be applied at at least one region boundary if it is indicated that the said region can be decoded independently.

- 1. The padding amount might depend on the size of the region.
- 2. The padding amount might be determined according to the modulo of the region size.

- 3. The padding might be repetitive padding or padding with a constant value.

- ii. Additionally or alternatively, the region size might be multiple of M samples.

- 1. The M might be of the form 2^N , wherein N might be a non-negative integer.

- 2. The M might be 64, or 128, or 256.

- 3. When the region size is determined to be multiple of M samples, all of the regions that are not at the right or bottom image boundary might be restricted to have a size that is multiple of M.

4. As an example if the region size is determined to be multiple of 64, no padding might be necessary for processing of the said region. Since padding requires extra processing, restricting the region sizes to be multiple of 64 eliminates the extra processing.

- 5 a. The padding operation might be necessary only at the regions that coincide with right or bottom image boundary.
- b. The padding operation might be performed in a way to make the region size multiple of 64.

10 6) To solve problem 5, according to some example embodiments of the present disclosure, the following might be performed.

- a. The process of entropy decoding of a region might be performed independently from other regions.
- b. The process of sample prediction of latent tensor reconstruction of a region might be performed independently from other regions.
- 15 c. The process of filtering of a region might be performed independently from other regions.
- i. The filtering process might comprise a convolution operation.
- ii. The convolution process might comprise padding (repetitive padding or padding with a constant), if the convolution process requires samples
- 20 outside of a region.
- iii. The convolution process might be restricted to use samples inside of a region.
- iv. The convolution process might be implemented as cross-correlation process.
- d. The process of tensor reconstruction of a region might be performed independently
- 25 from other regions.
- i. A bitstream might be decoded according to an order, wherein all samples corresponding to a region might be decoded before decoding of any samples corresponding to another region.
1. A hyper latent (or a hyper tensor substream) of a region might be
- 30 decoded first,
2. A primary residual stream of a region might be decoded secondly, and
3. A secondary residual stream of a region might be decoded thirdly, wherein a sample corresponding to a second region is not decoded after

the decoding of the first sample of the hyper latent and before the end of decoding of the last sample of the region.

ii. A residual tensor or a latent or a hyper latent tensor might be reconstructed according to an order, wherein samples of each region might be grouped together.

iii. A residual tensor or a latent or a hyper latent tensor might be reconstructed according to an order, wherein all samples of a first region is reconstructed first, then all samples of a second region are reconstructed.

e. Any of the above mentioned processed might comprise a neural network that might be applied to process a region independently from other regions.

i. The said neural network might be a hyper decoder or a hyper scale decoder or alike.

ii. The output of the neural network might be standard deviation logarithm tensor, standard deviation tensor or alike.

iii. The output of the neural network might be prediction tensor, explicit prediction tensor or alike.

iv. The said neural network might be performed multiple times on each of the regions in decoding of an image.

v. A cropping might be applied during or at the end of the processing with the neural network, wherein the cropping among is determined according to the region size.

1. Cropping might be applied only if a region shares a boundary with the bottom or right boundary of an image.

2. Cropping might not be applied if a region does not share any boundary with the bottom or right boundary of an image.

f. A lossless decoder, (arithmetic decoder, asymmetric numeral systems or alike) might be applied to obtain the residuals corresponding to a region independently of other regions.

g. The probability parameters of a region might be obtained independently of other regions.

i. Probability parameters (e.g. sigma parameters, gaussian sigma parameters etc.) corresponding to a region might be used to decode the samples of only one region and not used to obtain (i.e. decode) the samples of a second region.

h. The sample prediction of latent sample prediction process might comprise a multi-state context model process.

i. The coordinates of a region (e.g. vertical coordinate, horizontal coordinate) and/or the size of a region might be determined according to image size and a depth parameter.

i. Vertical coordinate of a region might be obtained according to $i * \text{VerRegionSize} / 2^d$.

ii. Vertical ending coordinate of a region might be obtained according to $(i == \text{NumVerSplits} - 1) ? \text{ceil}((\text{img_height} + 64) \div 2^d) : (i + 1) * \text{VerRegionSize} / 2^d$.

iii. Horizontal coordinate of a region might be obtained according to $i * \text{HorRegionSize} / 2^d$.

iv. Horizontal ending coordinate of a region might be obtained according to $(j == \text{NumHorSplits} - 1) ? \text{ceil}((\text{img_width} + 64) \div 2^d) : (j + 1) * \text{HorRegionSize} / 2^d$.

Wherein the I and j are the indices of the region, img_width , img_height are the width and height of the image and “ d ” is the depth of the tensor.

A neural network used to process the regions might comprise multiple processing layers, each one processing an input tensor and outputting an output tensor. The depth parameters might correspond to the position of a tensor output.

7) In one example, a first bitstream coded with at least one NN may be processed to get a second bitstream and the second bitstream may be decoded.

a. In one example, a sub-stream denoted as the second bitstream may be extracted from the first bitstream, and the sub-stream may be decoded.

i. In one example, the sub-stream may be a subset of the first bitstream.

1. In one example, the sub-stream may be a modified subset of the first bitstream.

ii. In one example, a sub-stream may correspond to a region in the picture corresponding to the first bitstream.

b. In one example, whether and/or how the first bitstream is processed to generate the second bitstream may be indicated in the first and/or second bitstream.

8) According to some example embodiments of the present disclosure, data corresponding to a region might be removed and/or extracted from the bitstream.

a. The data might be a substream, a bitstream, a syntax element or like.

- i. The data might comprise quality map information, a residual data, luma residual data, chroma residual data, a hyper latent data, residual latent data or alike.
- b. An indication might be included in the bitstream to indicate data corresponding to which region is present (or absent) in a bitstream.
 - i. The indication might be an index.
 - 1. The indication might indicate a region index or a substream index.
 - 2. The indication might be a list of indices, wherein each entry of the list might indicate the regions that are removed from the bitstream.
 - 3. The indication might be a list of indices, wherein each entry of the list might indicate the regions that are present in the bitstream.
 - ii. The indication might be a flag.
 - 1. The flag might indicate if a region is present in the bitstream or absent in the bitstream.
 - iii. The indication might indicate the number of regions that are present or absent in the bitstream.
 - 1. The indication might indicate the number of regions in a vertical dimension or a horizontal dimension that are present or absent in a bitstream.
 - 2. The indication might be `num_present_hor_splits_minus1` or `num_present_ver_splits_minus1`.
 - 3. The indication might indicate total number of regions present or absent in the bitstream.
- c. An indication might be included in the bitstream to indicate if data corresponding to a region is present (or absent) in a bitstream.
 - i. The data might comprise information related to;
 - 1. part of a region.
 - 2. residual information of a region.
 - 3. hyper latent information of a region.
 - 4. Quality map information of a region
 - ii. A first indication might be included in the bitstream to indicate the size of the data.
 - 1. The first indication might indicate the number of bits or number of bytes.

2. The first indication might be coded using predictive coding.
3. The first indication might be coded using fixed length or variable length coding.
4. A second indication might be included in the bitstream to indicate the length of the first indication.

- a. The second indication might indicate the number of bits or number of bytes that is used to encode/decode the first indication.

- b. The second indication might be `q_ls`, `z_ls`, `r_primary_ls` or `r_secondary_ls` as exemplified in embodiment 2.

- c. The second indication might be included in the bitstream (or obtained from the bitstream) based on a third indication.

- i. The third indication might be a flag.

5. The first indication might be `z_stream_size`, `q_stream_size`, `r_primary_stream_size`, or `r_secondary_stream_size` as exemplified in embodiment 1 or 2.

- d. An indication might be included in the bitstream to indicate if all regions are included in the bitstream or if a region is removed and/or extracted from the bitstream.

- i. The indication might be a flag.

- ii. The indication might be a `region_set_extracted_flag` flag.

- iii. If the value of the indication is true (or false), a second indication might be included in the bitstream to indicate:

1. number of regions that are removed (or present).

2. The indices of regions that are removed or present.

3. A flag to indicate if a region is removed or present.

- e. An indication might be included in the bitstream to indicate an area corresponding to regions that are removed or present.

- i. The area information might include a start coordinate and a size information.

- ii. The area information might include a number of regions that are removed or present.

- iii. The area information might include indices of a regions that are removed or present.

9) According to some example embodiments of the present disclosure, the regions might be

encoded/decoded independently or dependently.

- a. An indication might be included in the bitstream to indicate if a region might be decoded independently or dependently.

- b. If a region is indicated to be decoded independently:

- i. Samples corresponding to a neighbor region of the said independent region are not used in the processing of at least one processing module.

- 1. The processing module might be a hyper decoder, a hyper scale decoder, a sample prediction module, a latent prediction module, an MCM (multi-stage-context) module, an entropy decoding module or a synthesis transform module, etc.

- 2. The input of the processing module is set to comprise only the samples corresponding to said independent region.

- 3. The output of the processing module is assigned to the said independent region.

- ii. Samples corresponding to none of the neighbor regions of the said independent region are used in the processing of at least one processing module.

- c. If a region is indicated to be decoded dependently:

- i. At least one sample corresponding to a neighbor region of the said dependent region is used in the processing of at least one processing module.

- 1. The processing module might be a hyper decoder, a hyper scale decoder, a sample prediction module, a latent prediction module, an MCM (multi-stage-context) module, an entropy decoding module or a synthesis transform module, etc.

- 2. An extension size is used to determine the amount of samples to be used from a neighbor region.

- a. The extension size might be used to determine the size of the input of the processing module.

- b. The size might indicate the extension of the input of the processing module in either one of the horizontal or vertical direction.

- i. The size might indicate the extension amount in left/right/bottom or top direction.

- c. The extension size might be used to determine the cropping

amount of the output of a processing module (how much of the samples to be discarded from the output of the processing module).

d. The extension size might be different for different processing modules.

i. The extension size might be 1 for hyper decoder and hyper scale decoder.

ii. The extension size might be 4 or 8 for latent reconstruction process or MCM process or sample prediction process.

iii. The extension size might be indicated in the bitstream.

1. An indication might be included in the bitstream to determine the amount of extension in the input and output.

ii. The dependent regions might be removed from the bitstream.

iii. An independent region cannot be removed from the bitstream if a dependent region is present in the bitstream that requires samples from the said independent region.

iv. A sample corresponding to an independent region or a dependent region might be used in the processing of a dependent region.

10) The bitstream corresponding to a region might be encoded/decoded using an entropy coder in parallel.

a. A data corresponding to a region might be coded/decoded using an entropy coder/decoder in parallel.

i. The data might comprise information related to;

1. part of a region.

2. residual information of a region.

3. hyper latent information of a region.

4. Quality map information of a region.

b. An indication might be included in the bitstream to indicate the number of parallel processing units (e.g. threads) can be used.

i. The indication might indicate the number of threads.

ii. The indication might be:

1. num_threads_q_minus1,
2. num_threads_z_minus1,
3. num_threads_r_primary_minus1,
4. num_threads_r_secondary_minus1.

5 iii. The indication might be different for different data types. For example the number of parallel processing threads might be different for data corresponding to luma information or chroma information. The number might be different for hyper latent information (e.g. hyper tensor) or residual information or quality map information or picture header.

10 iv. The indication might be same for different data types.

c. The above mentioned parallel processing units (e.g. number of threads) might be predefined or inferred.

d. The parallel processing units (e.g. number of threads), or the maximum number of parallel processing units (maximum number of threads) might be determined based

15 on:

- i. A profile indication,
- ii. A picture size indication.
- iii. A number of regions indication,
- iv. Etc.

20 11) According to some example embodiments of the present disclosure, the maximum number or maximum size or minimum size of a region might be determined based on:

- a. A profile indication,
- b. A picture size indication.
- c. A number of regions indication,
- 25 d. Etc.

12) According to some example embodiments of the present disclosure, the regions might be centrally aligned, equipartitioned or shifted.

- a. An indication might be included in the bitstream to indicate that region coordinates are shifted.
- 30 b. The regions might be equipartitioned. E.g. the regions might have same size except for the right or bottom boundary.
 - i. An example is depicted in Fig. 19 on the left. The 4 regions (except for the regions at the right and bottom boundary have equal sizes).
- c. The regions might be shifted. Compared to the equipartitioned case, the regions

might be shifted to:

- i. Extend the size of the regions on the left and/or top boundary, and/or
- ii. Reduce the size of the regions at the right and/or bottom boundary.
- iii. An example is depicted in Fig. 19 on the right.

5 d. The processing areas corresponding to a region might be centrally aligned or shifted;
i. During the processing of a region the tensor used as input to a first processing module might be centrally aligned with the tensor used as input to a second processing module.

10 ii. During the processing of a region the tensor used as input to a first processing module might be shifted with respect to the tensor used as input to a second processing module.

15 iii. This is exemplified in Figs. 20A-20C, which illustrate three example alignment schemes. More specifically, Fig. 20A illustrates a scheme with centrally aligned processing areas corresponding to a region, Fig. 20B illustrates an example of shifted areas, and Fig. 20C illustrates a further example of shifted areas.

e. In one example the processing of a synthesis transform might be performed according to centrally aligned regions. And at least one processing unit is performed using non-aligned (e.g. shifted) regions.

20 f. In one example the processing of a synthesis transform might be performed according to centrally aligned regions. And at least one processing unit is performed using either the centrally aligned or non-aligned (e.g. shifted) regions.

i. The at least one processing unit might be one or more of the following:

- 25
1. Hyper decoder,
 2. hyper scale decoder,
 3. entropy decoder,
 4. sample prediction unit,
 5. MCM (multi-stage context model),
 6. latent sample reconstruction unit.

30 ii. The determination of if centrally aligned regions are used or not is determined based on an indication.

1. The indication might be included in a bitstream.
2. The indication might be inferred based on one or more of the following:

- a. Picture size,
- b. profile indication,
- c. level indication,
- d. etc.

5 13) The correspondence between an index of a sub-stream and a region in the original picture may depend on at least one signaled indication.

- a. In one example, the location (such as the coordinate of the top-left position) and/or the size (such as width and height) of the region corresponding with an index may be signaled.

10 b. Alternatively, the correspondence between an index of a sub-stream and a region in the original picture may be derived.

- i. In one example, the location (such as the coordinate of the top-left position) and/or the size (such as width and height) of the region corresponding with an index may be derived based on the region partitioning way of the whole picture and/or the index scanning order of regions.

15 14) Constrains may be applied to regions in a picture coded with at least one NN.

- a. For example, two regions should not be overlapped.
- b. For example, all regions should cover the whole picture.
- c. For example, the region location (such as the coordinate of the top-left position denoted as (X, Y)) may be constrained.

20 i. For example, X and/or Y must be in a form of $k \times N$, where k is an integer and N is a predefined integer such as 2, 4, 8, 16.

- d. The region size (such as width W and height H) may be constrained.
 - i. For example, X and/or Y must be in a form of $k \times N$, where k is an integer and N is a predefined integer such as 2, 4, 8, 16.

25 e. The number of total regions may be no larger than a predefined integer such as 16.

- f. The number of rows of regions may be no larger than a predefined integer such as 8.

- g. The number of columns of regions may be no larger than a predefined integer such as 8.

30 15) In one example, a syntax element signaled related to region based coding with at least one NN may be coded in a predictive way.

General aspects

16) Additional operations may be applied to or with the proposed method.

- a) A syntax element (a.k.a. an indication) 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.
- b) A syntax element representing a coding tool or a coding method may not be signalled and implicitly determined to be unused, if the coding tool or the coding method is regarded as not applicable or cannot be used.
- c) A syntax element disclosed above may be coded with at least one context model. Or it may be bypass coded.
- d) A syntax element disclosed above may be signaled in a conditional way.
- e) A syntax element disclosed above may be signaled at block level/ sequence level/group of pictures level/picture level/slice level/tile group level.
- f) 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.
- g) Whether to and/or how to apply the disclosed methods above may be dependent on coded information, such as colour format, colour component, slice/picture type.
- 17) The proposed methods may be applied to other image/video compression solutions with NN-based coding tools involved.

5. Embodiments

Below are some example embodiments for the detailed solutions aspects summarized above in Section 4.

Most relevant parts that have been added or modified are shown by using bolded words (e.g., **this format indicates added text**), and some of the deleted parts are shown by using words in italics between double curly brackets (e.g., *{{this format indicates deleted text}}*). There may be some other changes that are editorial in nature and thus not highlighted. It should be understood that only markings in this section are intended to represent changes.

5.1. Embodiment 1

This embodiment is for some of the detailed solution items summarized above in Section 4.

1. (3.5.20) padding layer

padding layer is denoted as $Padd(H_{in}, W_{in}, d, s_d)$, where H_{in}, W_{in} are height and width of tensor – input to Analysis transform, s_d is stride of proceeding convolution, d is depth of convolution layer in deep learnable encoder. Padding layer receives tensor of size $[C, h_{d-1}, w_{d-1}]$ and outputs tensor of size $[C, s_d h_d, s_d w_d]$, where $h_d = \text{ceil}(h_{d-1}/s_d)$; $w_d = \text{ceil}(w_{d-1}/s_d)$, $h_0 = H_{in}$, $w_0 = W_{in}$. By default padding is performed by replication. Different model of padding can be specified (for example, padding by zeros).

padding layer is denoted as $Padd(s)$, where h, w are height and width of the input tensor and s is stride. Padding layer receives tensor of size $[C, h, w]$ and outputs tensor of size $[C, h + h\%s, w + w\%s]$. Default padding is performed by replication when the padding mode is not explicitly specified. Different model of padding can be specified (for example, padding by zeros).

2. (3.5.22) cropping layer

cropping layer is denoted as $Crop(H_{in}, W_{in}, d, s_d)$, where H_{in}, W_{in} are height and width of tensor – output to Synthesis transform, s_d is stride of proceeding transposed convolution, d is depth of convolution layer in deep learnable reconstruction process. Cropping layer outputs a tensor of size $[C, h_{d-1}, w_{d-1}]$, where h_{d-1}, w_{d-1} defined in Table 2. Note, that $h_0 = H_{in}, w_0 = W_{in}$. Cropping is performed by discarding redundant elements.

cropping layer is alternatively denoted as $Crop(h, w)$, where h and w are height and width of the output tensor. Cropping layer outputs a tensor of size $[C, h, w]$ by discarding redundant elements.

3. (8.4) Latent space tiles

Latent space tiles decoding can be enabled for each component independently by flags signalled in picture header (section 9.3). If `tile_enable_Luma` or `tile_enable_Chroma` is true then corresponding component is decoded using latent tiling process illustrated on Fig. 21.

The number and location of tiles are determined by values tile size $S_{tileHor}$ and $S_{tileVer}$ (equal to `tile_size_Luma_hor` and `tile_size_Luma_ver` for primary component and `tile_size_Chroma_hor` and `tile_size_Chroma_ver` for secondary component respectively) and tile overlap m_{tile} (`tile_overlap_Luma` for primary component `tile_overlap_Chroma` for secondary component) signalled in picture header (section 9.3). Further, the ratio $r = H_{in}/h_4 = w_{in}/w_4$ between signal domain size and latent tensor size is known, which is $r = 2^4$ for

primary and $r = 2^3$ for secondary component.

The number of tiles vertically is $K = \text{ceil}((H_{in} - m_{tile}) / (S_{tileVer} - m_{tile}))$ and horizontally $L = \text{ceil}((W_{in} - m_{tile}) / (S_{tileHor} - m_{tile}))$, where H_{in}, W_{in} are height and width of output color plane.

H_{in}, W_{in} are height and width of output color plane.

For $k = 0, \dots, K - 1$ and $l = 0, \dots, L - 1$,

- Define tiles coordinates and sizes

- $i_0^{(k)} = k \cdot (S_{tileVer} - m_{tile})$ – vertical dimension tile start in signal domain;
- $i_0^{(k)} = i_0^{(k)} - m_{tile}/2$ if
 - top tile boundary is not at image boundary and

- top tile boundary is not at a region boundary with independent_region_flag equal to 1.
- $i_4^{(k)} = i_0^{(k)} / r$ – vertical dimension tile start in latent space;
- $H_0^{(k)} = (k < K - 1) ? S_{tileVer-m_{tile}} : H_{in} - i_0^{(k)}$ – vertical dimension tile size in signal domain;
- $H_0^{(k)} = H_0^{(k)} + m_{tile} / 2$ if
 - bottom tile boundary is not at image boundary and
 - bottom tile boundary is not at a region boundary with independent_region_flag equal to 1.
- $h_4^{(k)} = \text{ceil}(H_0^{(k)} / r)$ – vertical dimension tile size in latent space;
- $j_0^{(l)} = l \cdot (S_{tileHor} - m_{tile})$ – horizontal dimension tile start in signal domain;
- $j_0^{(k)} = j_0^{(k)} - m_{tile} / 2$ if
 - left tile boundary is not at image boundary and
 - left tile boundary is not at a region boundary with independent_region_flag equal to 1.
- $j_4^{(l)} = j_0^{(l)} / r$ – horizontal dimension tile start in latent space;
- $W_0^{(l)} = (l < L - 1) ? S_{tileHor-m_{tile}} : W_{in} - j_0^{(l)}$ – horizontal dimension tile size in signal domain;
- $W_0^{(k)} = W_0^{(k)} + m_{tile} / 2$ if
 - right tile boundary is not at image boundary and
 - right tile boundary is not at a region boundary with independent_region_flag equal to 1.
- $w_4^{(l)} = \text{ceil}(W_0^{(l)} / r)$ – horizontal dimension tile size in latent space.
- tile latent space tensor
 - For $c = 0, \dots, C - 1$, $i = 0, \dots, h_4^{(k)} - 1$ and $j = 0, \dots, w_4^{(l)} - 1$
 - $\hat{y}_{k,l}[c, i, j] = \hat{y}[c, i_4^{(k)} + i, j_4^{(l)} + j]$
 - For $c = 0, \dots, C_d - 1$, $i = 0, \dots, h_4^{(k)} - 1$ and $j = 0, \dots, w_4^{(l)} - 1$
 - $\tilde{y}_{k,l}[c, i, j] = \tilde{y}[c, i_4^{(k)} + i, j_4^{(l)} + j]$
- Synthesis transform for one tile
 - Synthesis transform described in section 8.3 with $\hat{y}_{k,l}$, $\tilde{y}_{k,l}$ of sizes $h_4^{(k)}$ and $w_4^{(l)}$ as inputs and $\hat{x}_{k,l}$ of size $H_0^{(k)}$ and $W_0^{(l)}$ as output.

- Merging reconstructed parts of tensor into one

$$\begin{aligned} \circ \text{ov}_l &= \\ &\begin{cases} 0, \text{ if } i_0^{(k)} == 0 \text{ or left tile boundary coincides with a region boundary} \\ \text{and independent_region_flag is equal to 1 for the region which comprises the tile.} \\ m_{\text{tile}}/2, \text{ otherwise} \end{cases} \end{aligned}$$

- overlap used on left boundary of the tile

$$\begin{aligned} 5 \quad \circ \text{ov}_t &= \\ &\begin{cases} 0, \text{ if } j_0^{(k)} == 0 \text{ or top tile boundary coincides with a region boundary} \\ \text{and independent_region_flag is equal to 1 for the region which comprises the tile.} \\ m_{\text{tile}}/2, \text{ otherwise} \end{cases} \end{aligned}$$

- overlap used on top boundary of the tile

$$\begin{aligned} \circ \text{ov}_r &= \\ &\begin{cases} 0, \text{ if } i_0^{(k)} + W_0^{(l)} \geq W_{in} \text{ or right tile boundary coincides with a region boundary} \\ \text{and independent_region_flag is equal to 1 for the region which comprises the tile.} \\ m_{\text{tile}} / 2 \end{cases} \end{aligned}$$

10 - overlap used on right boundary of the tile

$$\begin{aligned} \circ \text{ov}_b &= \\ &\begin{cases} 0, \text{ if } j_0^{(l)} + H_0^{(k)} \geq H_{in} \text{ or right tile boundary coincides with a region boundary} \\ \text{and independent_region_flag is equal to 1 for the region which comprises the tile} \\ m_{\text{tile}} / 2 \end{cases} \end{aligned}$$

- overlap used on bottom boundary of the tile

15 $\circ H_1^{(k)} = H_0^{(k)} - \text{ov}_t - \text{ov}_b$ vertical dimension tile size in signal domain without tile overlap;

$\circ W_1^{(l)} = W_0^{(l)} - \text{ov}_l - \text{ov}_r$ horizontal dimension tile size in signal domain without tile overlap;

$\circ \text{For } c = 0, \dots, C_{in} - 1, i = 0, \dots, H_1^{(k)} - 1 \text{ and } j = 0, \dots, W_1^{(l)} - 1,$
 $\quad \hat{x}[c, i_0^{(k)} + \text{ov}_t + i, j_0^{(l)} + \text{ov}_l + j] = \hat{x}_{k,l}[c, \text{ov}_t + i, \text{ov}_l + j].$

20 Tile size location and overlap signalling in Picture Header is described in Annex C.

4. (9.2) Code stream layout

Code stream structure is depicted in Fig. 22. The code stream is composed of the following parts with byte boundary, which are:

9. SOC - Start Of Codestream marker;

25 10. PIH (Picture Header marker) followed by picture header (section 9.3);

11. TOH (Tools Header marker) followed by tools information (section 9.3.1.1);

12. SOR (Start of region-stream marker) followed by codestream of hyper tensor z, including

$\hat{z}_Y$ ("*stream z_Y*" in Fig. 23) and $\hat{z}_{UV}$ ("*stream z_{UV}*" in Fig. 23), codestream of primary component residual, which includes $\hat{r}_Y$ ("*stream y_Y*" in Fig. 23), and codestream of secondary component residual, which includes $\hat{r}_{UV}$ ("*stream y_{UV}*" in Fig. 23), for one region.

5 13. EOC - End Of Codestream marker.

The overall syntax structure of an image is:

picture() {	Descriptor
SOC	u(16)
picture_header()	
tools_header()	
region_set_extracted_flag	u(1)
if(region_set_extracted_flag) {	
num_present_ver_splits_minus1	u(7)
num_present_hor_splits_minus1	u(7)
}	
for(i = 0; i < NumVerSplits; i++) {	
for(j = 0; j < NumHorSplits; j++) {	
SOR	u(16)
subsIdx	u(8)
q_stream_for_one_region(subsIdx)	
z_stream_for_one_region(subsIdx)	
r_primary_stream_for_one_region(subsIdx)	
r_secondary_stream_for_one_region(subsIdx)	
}	
}	
EOC	u(16)
}	

Each code stream starts with a 16-bit marker. All markers used in this specification are as follows:

Code assignment	Symbol	Description	Mandatory/Optional
0xff80	SOC	Start of codestream	Mandatory
0xff81	EOC	End of codestream	Mandatory

0xff82	PIH	Picture header	Mandatory
0xff83	TOH	Tools header	Optional
0xff84	VUI	Reserved for rendering information	Optional
0xff85	xxx	Reserved	Optional
0xff86	xxx	Reserved	Optional
0xff87	xxx	Reserved	Optional
0xff88	SOR	Start of region substream	Mandatory
0xff8c	xxx	Reserved	Optional
0xff8d	xxx	Reserved	Optional
0xff8e	xxx	Reserved	Optional
0xff8f	xxx	Reserved	Optional

region_set_extracted_flag equal to 1 specifies that the picture() syntax structure contains a set of extracted regions that forms a rectangular region of the full picture.

region_set_extracted_flag equal to 0 specifies that the picture() syntax structure contains the full picture.

num_present_ver_splits_minus1 plus 1 specifies the number of columns of regions contained in the picture() syntax structure. The value of **num_present_ver_splits_minus1** shall be less than or equal to **num_ver_splits_minus1**.

num_present_hor_splits_minus1 plus 1 specifies the number of rows of regions contained in the picture() syntax structure. The value of **num_present_hor_splits_minus1** shall be less than or equal to **num_hor_splits_minus1**.

When **region_set_extracted_flag** equal to 1, the value of the variable **NumVerSplits** is updated to be equal to **num_present_ver_splits_minus1+1**, and the value of the variable **NumHorSplits** is updated to be equal to **num_present_hor_splits_minus1+1**.

subIdx indicates the region substream index.

5. (9.3.1) Syntax table

picture_header() {	Descriptor
PIH	u(16)
picture_header_size	u(16)
img_width	u(16)
img_height	u(16)

picture_format	u(2)
bit_depth	u(1)
bit_c_ver	u(1)
bit_c_hor	u(1)
bit_s_ver	u(1)
bit_s_hor	u(1)
num_ver_splits_minus1	u(7)
num_hor_splits_minus1	u(7)
if(NumVerSplits > 1 NumHorSplits > 1) {	
regional_access_enabled_flag	u(1)
if(regional_access_enabled_flag)	
for(i = 0; i < NumVerSplits; i++)	
for(j = 0; j < NumHorSplits; j++)	
independent_region_flag[i][j]	u(1)
}	
num_threads_q_minus1	u(7)
num_threads_z_minus1	u(7)
num_threads_r_primary_minus1	u(7)
num_threads_r_secondary_minus1	u(7)
color_transform_header()	u(1)
model_header()	
}	

6. (9.3.1.1.) Model header

tile_header_Luma(tile_signaling_type) {	Descriptor
tile_enable_Luma	u(1)
if(tile_enable_Luma) {	
tile_size_Luma_hor	u(13)
tile_size_Luma_ver	u(13)
tile_overlap_Luma	u(8)
}	
}	

tile_header_Chroma(tile_signaling_type) {	Descriptor
tile_enable_Chroma	u(1)
if (tile_enable_Chroma) {	
tile_size_Chroma_hor	u(13)
tile_size_Chroma_ver	u(13)
tile_overlap_Chroma	u(8)
}	
}	
color_transform_header() {	Descriptor
color_transform_enable	u(1)
if(color_transform_enable)	
for (i=0;i<3;i++)	
for (j=0;j<3;j++)	
color_transform_matrix[i][j]	u(8)
for (i=0;i<3;i++)	
color_transform_offset[i]	u(8)
}	

7. (9.3.2) Picture header semantics

Following serves information is signalled:

picture_header_size is the number of bytes in the picture header excluding the first two-byte marker;

img_width plus 128 specifies width of an input picture (from 128 to 65600);

img_height plus 128 specifies height of the input picture (from 128 to 65600);

picture_format is a data format of the output picture (YUV420 = 0, YUV444 = 1, sRGB = 2, YUV422 = 3);

10 bit_depth is a bit-depth the output picture (“0” corresponds to 8 and “1” corresponds to 10);

bit_c_ver is one bit value which defines $c_ver = 1 + \text{bit_c_ver}$, c_ver controls internal subsampling mode of the secondary component in vertical direction as defined in Table 3 and Table 2. If

bit_c_ver is not present in bit-stream then c_ver is equal to 2.

bit_c_hor is one bit value which defines $c_hor = 1 + \text{bit_c_hor}$, c_hor controls internal

15 subsampling mode of the secondary component in vertical direction as defined in Table 3 and Table 2. If bit_c_hor is not present in bit-stream then c_hor is equal to 2.

bit_s_ver is one bit value which defines s_ver which is used for align coding subsampling mode

of the secondary component and subsampling mode in output picture format in vertical direction as defined in Table 3. Usage of s_ver is described in section 7.6 If bit_c_ver is not present in bit-stream then s_ver is equal to 1.

bit_s_hor is one bit value which defines $c_hor = 1 + bit_c_hor$, c_ver controls internal

- 5 subsampling mode of the secondary component in vertical direction as defined in Table 3 Usage of s_ver is described in section 7.6 If bit_c_ver is not present in bit-stream then s_hor is equal to 1.

$num_ver_splits_minus1$ plus 1 specifies the number of vertical splits. The variable

$NumVerSplits$ is derived to be equal to $num_ver_splits_minus1 + 1$. The maximum value of

- 10 $NumVerSplits$ is constrained such that the width of each vertical split shall be greater than or equal to 128 pixels. The maximum value of $NumVerSplits$ may also be further constrained depending on the profile and the level to which the codestream conforms.

The variable $VerRegionSize$ is set equal to $((img_height + 127)/128)/NumHorSplits * 128$.

$num_hor_splits_minus1$ plus 1 specifies the number of horizontal splits. The variable

- 15 $NumHorSplits$ is derived to be equal to $num_hor_splits_minus1 + 1$. The maximum value of $NumHorSplits$ is constrained such that the height of each horizontal split shall be greater than or equal to 128 pixels. The maximum value of $NumHorSplits$ may also be further constrained depending on the profile and the level to which the codestream conforms.

The picture is partitioned into $NumVerSplits * NumHorSplits$ rectangular regions.

- 20 The variable $HorRegionSize$ is set equal to $((img_width + 127)/128)/NumVerSplits * 128$.

The tensor $RC[7][NumHorSplits][NumVerSplits][6]$ array is set as follows:

For $d = 0...6, i = 0...NumHorSplits-1, j = 0...NumVerSplits-1$;

- $RC[d][i][j][0] = i * VerRegionSize / 2^d$
- $RC[d][i][j][1] = (i == NumHorSplits - 1) ? ceil(H_{in} / 2^d) : (i + 1) * VerRegionSize / 2^d$
- 25 – $RC[d][i][j][2] = j * HorRegionSize / 2^d$
- $RC[d][i][j][3] = (j == NumHorSplits - 1) ? ceil(W_{in} / 2^d) : (j + 1) * HorRegionSize / 2^d$
- $RC[d][i][j][4] = RC[d][i][j][1] - RC[d][i][j][0]$
- $RC[d][i][j][5] = RC[d][i][j][3] - RC[d][i][j][2]$

For $d = 0...6, y = 0...ceil(H_{in} / 2^d) - 1, x = 0...ceil(W_{in} / 2^d) - 1$;

- 30
- $i = 0...NumHorSplits-1, j = 0...NumVerSplits-1$;
 - if $y \geq RC[d][i][j][0]$ and $y < RC[d][i][j][1]$ and $x \geq RC[d][i][j][2]$ and $x < RC[d][i][j][3]$;
 - $sample2substrIdx[d][y][x]$ is set equal to $i * NumVerSplits + j$

$regional_access_enabled_flag$ equal to 1 specifies that a subset of each rectangular region

can be correctly decoded without the presence of coded data of other rectangular regions, where the subset is obtained by shifting the boundaries of the rectangular region towards the inside by the tile overlap size. `regional_access_enabled_flag` equal to 0 specifies that there may or may not be such a subset of each rectangular region that can be correctly

5 decoded without the presence of coded data of other rectangular regions.

When `regional_access_enabled_flag` is equal to 1, the following constraints apply:

- The value of `tile_size_Luma_ver` shall be less than or equal to `VerRegionSize`.
- The value of `tile_size_Luma_hor` shall be less than or equal to `HorRegionSize`.
- $(\text{tile_size_Luma_ver} + s_{ver} - 1) / s_{ver}$ shall be equal to `tile_size_Chroma_ver`.
- 10 - $(\text{tile_size_Luma_hor} + s_{hor} - 1) / s_{hor}$ shall be equal to `tile_size_Chroma_hor`.
- Each luma or chroma sample on a region boundary shall also be on a tile boundary.

`independent_region_flag[i][j]` equal to 1 specifies that the rectangular region on the i-th row and the j-th column can be correctly decoded without the presence of coded data of other rectangular regions. `independent_region_flag[i][j]` equal to 0 specifies that the

15 rectangular region on the i-th row and the j-th column may or may not be correctly decoded without the presence of coded data of other rectangular regions.

`num_threads_z_minus1` plus 1 is the number of parallelly decodable sub-stream in the hyper tensor codestream; the maximum value of `num_threads_z_minus1` is 127.

`num_threads_q_minus1` plus 1 is the number of parallelly decodable sub-stream in the

20 quality map codestream; the maximum value of `num_threads_q_minus1` is 127.

`num_threads_r_primary_minus1` plus 1 is the number of parallelly decodable sub-stream in primary component residual tensor codestream; the maximum value of `num_threads_r_primary_minus1` is 127 and is depends on profiles and levels.

`num_threads_r_secondary_minus1` plus 1 is the number of parallelly decodable sub-stream

25 in secondary component residual tensor codestream; the maximum value of `num_threads_r_secondary_minus1` is 127 and is depends on profiles and levels.

`scale_comp[2]` are vertical and horizontal ratio between primary and secondary components sizes of coded picture $s_{ver} = \text{scale_comp}[0]$ and $s_{hor} = \text{scale_comp}[1]$; if not present

(`res_changer_enable` = false) then $s_{ver} = 2$ and $s_{hor} = 2$. Allowed values are listed in Table 3.

30 `independent_beta_uv` is a flag (false/true) which indicates do the rate control parameter (β) for primary and secondary components are the same.

`beta_displacement_log_y` – parameter indicating ratio between rate control parameter beta selected by encoder for primary component and one used in the model training.

$\text{betaDisplacementLogY} = \text{beta_displacement_log_y} - 2^{11}$

beta_displacement_log_uv – parameter indicating ratio between rate control parameter beta selected by encoder for secondary component and one used in the model training.

betaDisplacementLogUV = beta_displacement_log_uv – 2¹¹

model_id is an identifier of pre-stored checkpoint with model's weights, model_id = 0,1,2,3 or

5 4.

opIdx is an identifier for operation point, 0 means “base”, 1 means “high” operation point.

tile_enable_Luma and tile_enable_Chroma are enable flags for tiling of primary and secondary components.

tile_size_Luma_ver and tile_size_Chroma_ver are the sizes of tiles for primary and secondary
10 components in the vertical direction.

tile_size_Luma_hor and tile_size_Chroma_hor are the sizes of tiles for primary and secondary components in the horizontal direction.

tile_overlap_Luma and tile_overlap_Chroma are sizes of tiles overlapping areas for primary and secondary components.

15 cube_group_flag is 1-bit unsigned integer. cube_group_flag=0 indicates no cube_flag is signalled in one group, and cube flags of one group are set to be 1. cube_group_flag=1 indicates cube flags of one group are signaled.

cube_luma_flag is 1D array of size $((h_{4,Y} + 7) \gg 3) \cdot ((w_{4,Y} + 7) \gg 3)$, which contains cube flags for primary component. 1 indicates Skip Mode is applied to one cube of residual tensor of
20 primary component $\hat{r}_Y$. 0 indicates Skip Mode is disable for one cube of residual tensor of primary component $\hat{r}_Y$.

cube_chroma_flag is 1D array of size $((h_{4,UV} + 7) \gg 3) \cdot ((w_{4,UV} + 7) \gg 3)$, which contains cube flags for secondary component. 1 indicates Skip Mode is applied to one cube of residual tensor of secondary component $\hat{r}_{UV}$. 0 indicates Skip Mode is disable for one cube of residual
25 tensor of secondary component $\hat{r}_{UV}$.

color_transform_enable is an enable flag for color conversion module.

color_transform_matrix[i][j] is a matrix of color conversion. If not present

(color_transform_enable is false) then default ITU-R BT.709 colour transform is used (refer to section 7.8).

30 color_transform_offset[i] is an offset for color transformation If not present

(color_transform_enable is false) then default ITU-R BT.709 colour transform is used (refer to section 7.8).

8. (9.5.1.1) Syntax table of hyper tensor

z_stream_for_one_region(substrIdx) {	Descriptor
z_stream_size	u(24)
for (i =1; i <= num_threads_z_minus1; ++i) {	
thread_offsets_z[i]	u(24)
}	
for (i =0; i <= num_threads_z_minus1; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_z_minus1+1, thread_offsets_z, z_stream_size)	
}	
for (ch =0; ch < z_primary_ch; ++ch) {	
for (i =0; i <= num_threads_z_minus1; ++i) {	
decode_z_data(sub_stream[i], z_primary[substrIdx][ch][], ch, RC[6][i][j][5], RC[6][i][j][4], i, num_threads_z_minus1+1)	
}	
}	
for (ch =0; ch < z_secondary_ch; ++ch) {	
for (i =0; i <= num_threads_z_minus1; ++i) {	
decode_z_data(sub_stream[i], z_secondary[substrIdx][ch][], ch, RC[6][i][j][5], RC[6][i][j][4], i, num_threads_z_minus1+1)	
}	
}	
}	

9. (9.5.2) Quality map information decoder

The input of this process is

- Codestream for quality_map;
- 5 - Indicators for sigma tables which are decoded in picture header:
 - o quality_map_entropy_index_y for primary component.
 - o quality_map_entropy_index_uv for secondary component.

The output of this process is

- quality_map_delta_Y is an array of size $[w_{4Y}, h_{4Y}]$ with information used for deriving scaling factor (section 12.2) for primary component residual tensor.
- 10 - quality_map_delta_UV is an array of size $[w_{4UV}, h_{4UV}]$ with information used for deriving scaling factor (section 12.2) for secondary component residual tensor.

Here sizes $[w_{4Y}, h_{4Y}]$ and $[w_{4UV}, h_{4UV}]$ are defined in Table 2.

$q_primary_sigma_Idx$ is a 2D array of size $[numHorSplits * numVerSplits][]$, and each element in this array is equal to $q_sigma_Idx[quality_map_entropy_index_y]$, where the $q_sigma_Idx[k]$ is derived according to Table 4.

- 5 $q_secondary_sigma_Idx$ is a **2D** array of size $[numHorSplits * numVerSplits][]$, and each element in this array is equal to $q_sigma_Idx[quality_map_entropy_index_uv]$.

$q_primary$ is a **2D** array of size $[numHorSplits * numVerSplits][]$, $q_secondary$ is a **2D** array of size $[numHorSplits * numVerSplits][]$.

$quality_map_delta_Y$ and $quality_map_delta_UV$ are determined by determined by:

- 10 – **for** $y = 0 \dots NumHorSplits$ and $x = 0 \dots NumVerSplits$
- **k** is set equal to 0.
 - **For** $i = RC[4][y][x][0] \dots RC[4][y][x][1] - 1$,
 - $j = RC[4][y][x][2] \dots RC[4][y][x][3] - 1$
 - $quality_map_delta_UV[i, j] = q_secondary[y * NumVerSplits + x][k]$
 - $quality_map_delta_Y[i, j] = q_primary[y * NumVerSplits + x][k]$
 - **k** = **k**+1
- 15

Table 4 Determining q_sigma_Idx for quality map delta decoding

k	0	1	2	3	4	5	6	7
$q_sigma_Idx[k]$	0	4	5	6	8	10	14	18

- 20 10. (9.5.2.1) Syntax table of quality map information tensor

$q_stream_for_one_region(substrIdx)$	Descriptor
q_stream_size	u(24)
for ($i = 1; i \leq num_threads_q_minus1; ++i$) {	
$thread_offsets_q[i]$	u(24)
}	
for ($i = 0; i \leq num_threads_q_minus1; ++i$) {	
$sub_stream[i] = sub_stream_init(i, num_threads_q_minus1 + 1, thread_offsets_q, q_stream_size)$	
}	
for ($i = 0; i \leq num_threads_q_minus1; ++i$) {	

decode_r_data(sub_stream[i], q_primary[substrIdx][], q_primary_sigma_idx_2D[substrIdx][], i, num_threads_q_minus1+1)	
}	
for (i =0; i <= num_threads_q_minus1; ++i) {	
decode_r_data(sub_stream[i], q_secondary[substrIdx][], q_secondary_sigma_idx_2D [substrIdx][], i, num_threads_q_minus1+1)	
}	
}	

11. (9.5.3) Primary residual stream decoder

The input of this process is

- Primary component residual codestream;
- 5 - mask_skip[C_p, h_{4Y}, w_{4Y}] generated by Skip Mask generation (section 13.3.2) for primary component;
- sigma_idx_primary – tensor of size $[C_p, h_{4Y}, w_{4Y}]$ which is the output of Sigma quantization (section 10.8) for primary component.

The output of this process is

- 10 - r_primary – one dimensional array {s} of residual tensor elements which is an input of decoder skip process (section 13.3.4) for primary component.

Here sizes C_p, w_{4Y}, h_{4Y} are defined in Table 2. A one dimensional array sigma_idx_primary_1D is determined though following steps:

- Set count[NumVerSplits*NumHorSplits] equal to all zeros,
- 15 - For $c = 0..C_p - 1$,
 - For $i = 0..h_{4Y} - 1$,
 - For $j = 0..w_{4Y} - 1$
 - subIdx = sample2SubstrIdx(4, i, j)
 - if mask_skip[c, i, j] is equal to True,
 - 20 sigma_idx_primary_2D[subIdx][count[subIdx]] is set to equal to sigma_idx_primary[c, i, j] and increase count[subIdx] by 1.

12. (9.5.3.1) Syntax table of primary residual tensor

r_primary_stream_for_one_region(substrIdx) {	Descriptor
r_primary_stream_size	u(32)

for (i = 1; i <= num_threads_r_primary_minus1; ++i) {	
thread_offsets_r_primary[i]	u(32)
}	
for (i = 0; i <= num_threads_r_primary_minus1; ++i) {	
sub_stream[i] = sub_stream_init(i, num_threads_r_primary_minus1 + 1, thread_offsets_r_primary, r_primary_stream_size)	
decode_r_data(sub_stream[i], r_primary[substrIdx][i], sigma_idx_primary_2D[substrIdx][i], i, num_threads_r_primary_minus1 + 1)	
}	
}	

13. (9.5.4) Secondary residual stream decoder

The input of this process is

- Codestream for secondary component residual;
- 5 - mask_skip[C_s, h_{4UV}, w_{4UV}] generated by Skip Mask generation (section 13.3.2) for secondary component;
- sigma_idx_secondary – tensor of size $[C_s, h_{4UV}, w_{4UV}]$ which is the output of Sigma quantization (section 10.8) for secondary component.

The output of this process is

- 10 - r_secondary – one dimensional array {s} of residual tensor elements which is an input of decoder skip process (section 13.3.4) for secondary component.

Here sizes C_s, h_{4UV}, w_{4UV} are defined in Table 2. A one dimensional array sigma_idx_secondary_1D is determined through following steps:

- **Set count[NumVerSplits*NumHorSplits] equal to all zeros**
- 15 - **For $c = 0..C_p - 1$,**
 - **For $i = 0..h_{4Y} - 1$,**
 - **For $j = 0..w_{4Y} - 1$**
 - **subIdx = sample2SubstrIdx(4, i, j)**
 - **if mask_skip[c, i, j] is equal to True,**
 - 20 **sigma_idx_secondary_2D[subIdx][count[subIdx]] is set to equal to**
 - sigma_idx_secondary[c, i, j] and increase count[subIdx] by 1.**

14. (9.5.4.1) Syntax table of secondary residual tensor

r_secondary_stream_for_one_region (substrIdx) {	Descriptor
r_secondary_stream_size	u(24)
for (i =1; i <= num_threads_r_secondary_minus1; ++i) {	
thread_offsets_r_secondary[i]	u(24)
}	
for (i=0; i <= num_threads_r_secondary_minus1; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_r_secondary_minus1+1, thread_offsets_r_secondary, r_secondary_stream_size)	
decode_r_data(sub_stream[i], r_secondary[substrIdx] [], sigma_idx_secondary_2D[substrIdx] [], i, num_threads_r_secondary_minus1+1)	
}	
}	

15. (10.3) Hyper Latent Tensor Reconstruction

Input of this process is reconstructed $z_primary$ and $z_secondary$ tensors.

Output of this process is $\hat{z}[C, h_6, w_6]$ reconstructed hyper tensor.

- 5 – The following ordered steps are applied:
 - s tensor is set equal to $z_primary$ if primary component is being processed, or $z_secondary$ tensor if secondary component is being processed.
 - k is set equal to 0.
 - for $y = 0 \dots \text{NumHorSplits}$ and $x = 0 \dots \text{NumVerSplits}$
 - 10 – $k = 0$
 - For $c = 0 \dots C - 1$
 - For $i = RC[6][y][x][0] \dots RC[6][y][x][1] - 1$,
 - $j = RC[6][y][x][2] \dots RC[6][y][x][3] - 1$
 - $\hat{z}[c, i, j] = s[y * \text{NumVerSplits} + x][c][k]$ and $k = k+1$

15 16. (10.4) Hyper Scale Decoder

Hyper scale decoder net is depicted in Fig. 24.

The input of hyper scale decoder is

- $\hat{z}[C, h_6, w_6]$ reconstructed hyper tensor,
- sizes of input/output tensor H_{in}, W_{in} ,
- 20 – operation point indicator $opIdx$,
- model parameters for Hyper Scale Decoder Net defined by pair $(modelIdx, opIdx)$,

all multiplier parameters in those models are 8-bits integer.

The output of hyper scale decoder is standard deviation logarithm tensor $I_\sigma [C, h_4, w_4]$ with integer values in a range $0 \leq I_\sigma < ((N_\sigma - 1) \ll \text{sigmaPrecision})$, where N_σ , sigmaPrecision are defined in section 0. Sizes of those tensors for primary and secondary components are listed in Table 2.

All operation in scalable hyper decoder are integer, an accumulator in all computations is within 32 bits integer diapason, multipliers in model parameters are quantized to 8-bits integer. This guarantees bit-exact behaviour of this neural network module. Hyper scale decoder uses special type of operations quantized convolutions. For each quantized convolution in the process the set of clipping values $\{d_k\}$ and de-scaling shifts parameters $\{p_k\}$ ($1 \leq k \leq 3$) are specified for each channel. All clipping values in quantized convolutions are $d_k = 2^7$. De-scaling shifts $\{p_k\}$ are provided in section 15.6.

NOTE – the magnitude of integerized weights in quantized model doesn't exceed 2^7 , shift and clipping value combination ensures the register of quantized convolution is within 32-bits.

The sequence of operations in this module doesn't depend on operation point indicator (*opIdx*), but model parameters are different for base and high operation point.

Hyper scale decoder starts with quantized convolution kernel size 1×1 , stride 1, followed by rectified linear unit. Then there is a stride 1 quantized convolution with kernel size 3×3 , followed by rectified linear unit. The next stride 1 quantized convolution (kernel size 1×1) increases the number of channels to $16C$. It is followed by pixel shuffle (stride 4), which brings number of channels back to C . The cropping layer (stride 4, depth 5) ensures the size of output tensor is $[C, h_4, w_4]$. The process concluded with abs operation.

The hyper decoder operation is applied in tiled manner. The following are the processing:

```

25  – for Vtile = 0...NumHorSplits-1 and Htile = 0...NumVerSplits-1
      – ext = independent_region_flag[ Vtile ][ Htile ] ? 1 : 0
      – padleft = ext if Htile > 0 else 0
      – padright = ext if Htile < NumVerSplits – 1 else 0
      – padtop = ext if Vtile > 0 else 0
30  – padbottom = ext if Vtile < NumHorSplits – 1 else 0
      – y = RC[6][Vtile][Htile][0] - padtop
      – x = RC[6][Vtile][Htile][1] - padleft
      – yO = RC[4][Vtile][Htile][0]
```

- $\mathbf{xO} = RC[4][Vtile][Htile][1]$
- $\mathbf{yE} = RC[4][Vtile][Htile][2]$
- $\mathbf{xE} = RC[4][Vtile][Htile][3]$
- $\mathbf{h} = RC[6][Vtile][Htile][4] + \mathbf{pad}_{top} + \mathbf{pad}_{bottom}$
- 5 – $\mathbf{w} = RC[6][Vtile][Htile][5] + \mathbf{pad}_{left} + \mathbf{pad}_{right}$
- $\mathbf{hO} = RC[4][Vtile][Htile][4] + \mathbf{pad}_{top} * 4 + \mathbf{pad}_{bottom} * 4$
- $\mathbf{wO} = RC[4][Vtile][Htile][5] + \mathbf{pad}_{left} * 4 + \mathbf{pad}_{right} * 4$
- $\mathbf{hImO} = RC[0][Vtile][Htile][4] + \mathbf{pad}_{top} * 64 + \mathbf{pad}_{bottom} * 64$
- $\mathbf{wImO} = RC[0][Vtile][Htile][5] + \mathbf{pad}_{left} * 64 + \mathbf{pad}_{right} * 64$
- 10 – the tensor part $\hat{\mathbf{z}}[C, y \dots y + \mathbf{h} - 1, x \dots x + \mathbf{w} - 1]$ is fed into the hyper scale decoder to obtain the tensor $Iq_{tile}[C, hO, wO]$
- $I_{\sigma}[C, yO \dots yE - 1, xO \dots xE - 1] = Iq_{tile}[C, \mathbf{pad}_{top} * 4 \dots RC[4][Vtile][Htile][4] + \mathbf{pad}_{top} - 1, \mathbf{pad}_{left} * 4 \dots RC[4][Vtile][Htile][5] + \mathbf{pad}_{left} + 1]$.
- 15 Model with weights are storied in electronic attachment, in format specied in section 4.8 and locates in <oper_point>/model_<MID>/<COMP>/hyper_scale_decoder.onnx, where <oper_point> is “base” or “high”, <MID> is an integer from 0 to 5, <COMP> is “primary” or “secondary”.

20 17. (11.2) Hyper Decoder

The learning-based hyper decoder consists of two independent pipe-lines with identical neural network architecture, except input size and number of channels.

The input of this process is

- $\hat{\mathbf{z}}[C, h_6, w_6]$ reconstructed hyper latent tensor,
- 25 – model parameters for Hyper Decoder Net defined by (*modelldx*),
- operation point indicator *opldx*.

The output of this process is

- $p[C_p, h_4, w_4]$ is explicit prediction (part of predicton tensor derived from explicitly signalled information), with channels size $C_p = (opldx + 1)C$.
- 30 Hyper decoder process is depicted in Fig. 25.

Hyper decoder starts stride 1 convolution with kernel size 1×1 , followed by inverse convolution (stride 2, kernel size 4×4), cropping layer (depth 6) and leaky rectified linear unit. Next step is stride 1 convolution with kernel size 3×3 , followed by inverse convolution (stride 2, kernel size

4×4), cropping layer (depth 5) and leaky rectified linear unit. Number of channels keep unchanged till this point (equal to number of channels C of input tensor). Hyper decoder concluded by stride 1 convolution with kernel size 3×3 which increases number of channels to $2C$ for high operation point and keeps number of channels unchanged for base operation point followed by leaky rectified linear unit.

The hyper decoder operation is applied in tiled manner. The following are the processing:

```

– for Vtile = 0...NumHorSplits-1 and Htile = 0...NumVerSplits-1
    – ext = independent_region_flag[ Vtile ][ Htile ] ? 1 : 0
    – padleft = ext if Htile > 0 else 0
    – padright = ext if Htile < NumVerSplits – 1 else 0
    – padtop = ext if Vtile > 0 else 0
    – padbottom = ext if Vtile < NumHorSplits – 1 else 0
    – y = RC[6][Vtile][Htile][0] - pad_top
    – x = RC[6][Vtile][Htile][1] - pad_left
    – yO = RC[4][Vtile][Htile][0]
    – xO = RC[4][Vtile][Htile][1]
    – h = RC[6][Vtile][Htile][4] + pad_top + pad_bottom
    – w = RC[6][Vtile][Htile][5] + pad_left + pad_right
    – hO = RC[4][Vtile][Htile][4] + pad_top * 4 + pad_bottom * 4
    – wO = RC[4][Vtile][Htile][5] + pad_left * 4 + pad_right * 4
    – hImO = RC[0][Vtile][Htile][4] + pad_top * 64 + pad_bottom * 64
    – wImO = RC[0][Vtile][Htile][5] + pad_left * 64 + pad_right * 64
    – the tensor part  $\hat{z}[C, y \dots y + h - 1, x \dots x + w - 1]$  is fed into the hyper decoder to
      obtain the tensor p_tile[Cp, hO, wO]
    – p[Cp, yO ... RC[4][Vtile][Htile][1] – 1, xO ... RC[4][Vtile][Htile][3] – 1] =
      p_tile[Cp, padtop * 4 ... RC[4][Vtile][Htile][4] + padtop – 1, padleft *
      4 ... RC[4][Vtile][Htile][5] + padleft + 1].

```

Model with weights are stored in electronic attachment, in format specified in section 4.8 and located in <oper_point>/model_<MID>/<COMP>/hyper_decoder.onnx, where <oper_point> is “base” or “high”, <MID> is an integer from 0 to 5, <COMP> is “primary” or “secondary”.

18. (11.3) Latent tensor reconstruction process

The input of this process is

- operation point indicator $opIdx$,
- $\hat{r} [C, h_4, w_4]$ reconstructed residual tensor, which is an out out of SKIP Model process (13.3.4),
- $p [C_p, h_4, w_4]$ explicit prediction, with channels size $C_p = (opIdx + 1)C$, which is an

5

The output of this process is

- $\hat{y}' [C, h_4, w_4]$ reconstructed latent tensor.

The process is as follows.

If $opIdx == 0$ (base operation point) then multi-stage context modelling process is by-

10

passed,

- explicit prediction is added to residual $\hat{y}' = \hat{r} + p [0: C - 1, h_4, w_4]$:

If $opIdx == 1$ (high operation point) then:

- Multi-stage Context Modelling process (section 11.4) is used.

19. (11.4) Tiled Multistage Context Modelling

15

The input of this process is

- $\hat{r} [C, h_4, w_4]$ reconstructed residual tensor, which is an out out of SKIP Model process (13.3.4),
- $p [2C, h_4, w_4]$ explicit prediction, which is an output of Hyper Decoder (11.2),
- Eight MCM_k , $k=0,...,7$ models with parameters defined by ($modelIdx, k$).

20

The output of this process is

- $\hat{y}' [C, h_4, w_4]$ reconstructed latent tensor.
- $\mu [C, h_4, w_4]$ reconstructed latent tensor.

The following steps are applied:

- for $Vtile = 0...NumHorSplits-1$ and $Htile = 0...NumVerSplits-1$

25

○ $ext = independent_region_flag [Vtile][Htile] ? 8 : 0$

○ $pad_{left} = ext$ if $Htile > 0$ else 0

○ $pad_{right} = ext$ if $Htile < NumVerSplits - 1$ else 0

○ $pad_{top} = ext$ if $Vtile > 0$ else 0

○ $pad_{bottom} = ext$ if $Vtile < NumHorSplits - 1$ else 0

30

○ $y = RC[4][Vtile][Htile][0] - pad_{top}$

○ $x = RC[4][Vtile][Htile][1] - pad_{left}$

○ $h = RC[4][Vtile][Htile][4] + pad_{top} + pad_{bottom}$

- $w = RC[4][Vtile][Htile][5] + pad_left + pad_right$
- Multistage Context Modelling subsection is invoked with $\hat{r}[C, y \dots y + h - 1, x \dots x + w - 1]$ and $p[2C, y \dots y + h - 1, x \dots x + w - 1]$ as inputs and $\hat{y}'_{tile}[C, h, w]$ and $\mu_{tile}[C, h, w]$ as outputs.
- 5 ○ $\hat{y}'[C, y \dots RC[4][Vtile][Htile][1] - 1, x \dots RC[4][Vtile][Htile][3] - 1] = \hat{y}'_{tile}[C, pad_{top} \dots RC[4][Vtile][Htile][4] + pad_{top} - 1, pad_{left} \dots RC[4][Vtile][Htile][5] + pad_{left} - 1]$
- $\mu[C, y \dots RC[4][Vtile][Htile][1] - 1, x \dots RC[4][Vtile][Htile][3] - 1] = \mu_{tile}[C, pad_{top} \dots RC[4][Vtile][Htile][4] + pad_{top} - 1, pad_{left} \dots RC[4][Vtile][Htile][5] + pad_{left} - 1]$.
- 10

20. (11.5) Multistage Context Modelling

The input of this process is

- $\hat{r}[h, w]$ reconstructed residual tensor, which is an out out of SKIP Model process (13.3.4),
- 15 – $p[2C, h, w]$ explicit prediction, which is an output of Hyper Decoder (11.2),
- Eight MCM_k , $k=0, \dots, 7$ models with parameters defined by $(modelIdx, k)$.

The output of this process is

- $\hat{y}'_{tile}[C, h, w]$ reconstructed latent tensor.

The process consists of following steps:

- 20 – **$h2$ and $w2$ are set equal to $(h + 1)/2, (w + 1)/2$ respectively.**
- padding layer (depth 5, stride 2) and down-shuffle (11.5.1) $M=2$ of explicit prediction tensor $p[2C, h, w]$ to $\ddot{p}[8C, h2, w2]$ re-shaped prediction tensor,
- padding layer (depth 5, stride 2) and down-shuffle (11.5.1) $M=1$ of reconstructed residual $\hat{r}[C, h, w]$ to $\ddot{r}[4C, h2, w2]$ re-shaped residual tensor,
- 25 – split $\ddot{p}[8C, h2, w2]$ into four parts $\ddot{p}_l = \ddot{p}[2lC: 2(l + 1)C - 1, h2, w2], l = 0, \dots, 3$ (each parts consists of $2C$ out of $8C$ channels),
- split $\ddot{r}[4C, h2, w2]$ into eight parts $\ddot{r}_k = \ddot{r}[kC/2: (k + 1)C/2 - 1, h2, w2], k = 0, \dots, 7$ (each parts consists of $C/2$ out of $4C$ channels),
- For $k=0, \dots, 3$,
- 30 ○ $MCM(k)$ process which
 - takes as an input
 - $\{\ddot{y}_m\}, m = 0, \dots, k - 1$ previously reconstructed parts of re-shaped

The output of the lossless decoding process is a 1D array $\{s_k\}$, whose size is equal to the total number of “1”s in the mask_skip $[C, h_4, w_4]$ tensor.

In other words, the mask_skip $[C, h_4, w_4]$ 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

5 inferred to be equal to zero.

The process of residual skip mode at the decoder is as follows:

- Dimensions $[C, h_4, w_4]$ are set equal to number of channels, height and width of the sigma tensor σ (Table 2).

- Tensors $\hat{r} [C, h_4, w_4]$ is initialized to be equal to all zeros.

10 – The counter $k = 0$.

- **The following ordered steps are applied:**

- **for Vtile = 0...NumHorSplits-1 and Htile = 0...NumVerSplits-1**

- **k = 0**

- **$i = RC[4][Vtile][Htile][0] \dots RC[4][Vtile][Htile][1] - 1$,**

15 ▪ **$j = RC[4][Vtile][Htile][2] \dots RC[4][Vtile][Htile][3] - 1$**

- **For $c = 0 \dots C - 1$,**

- **If mask_skip $[c, i, j]$ is equal to 1, $\hat{r}[c, i, j] = (s[sample2SubStrIdx[4, i, j]][k] - 2^{16-1} + 1)$, $k=k+1$.**

- **Otherwise $\hat{r}[c, i, j]=0$.**

20

22. (14.1.1)) Inter channel correlation information filter tiling and output selection process

The ICCI filter processes the input in tiles using the same method described in Section 8.4. The

tile size icci_tile_size and tile overlap icci_tile_overlap are same for the primary (Luma) and

25 secondary (Chroma) components and signalled in picture header (Section 9.3). The ratio r

between input domain size and output tensor size is fixed to 1 for each component. The minimum tile size is limited to 176 for the stable computation of the model selection explained next.

In total, 10 ICCI filters are included into the design. In each tile *tileID* for each color component *comp*=0..2 (“0” – “y”, “1” – “u”, “2” – “v”) the filter is selected based on

30 icci_model_idx[*comp*][*tileID*] value which is coded in picture header (Section 9.3). In case icci_model_idx = 0, the ICCI processing is bypassed.

The ICCI process is applied in a region based manner. When processing a sample (y,x) firstly the following parameters are set:

- **Vtile and Htile are set equal to the vertical and horizontal coordinates of the**

region that the sample at (y,x) belongs.

- If `independent_region_flag[Vtile][Htile]` equal to 1;
 - $ymin = RC[2 - s_{ver}][Vtile][Htile][0]$
 - $ymax = RC[2 - s_{ver}][Vtile][Htile][2] - 1$
 - 5 ▪ $xmin = RC[2 - s_{hor}][Vtile][Htile][1]$
 - $xmax = RC[2 - s_{hor}][Vtile][Htile][3] - 1$
- Else;
 - $ymin$ and $xmin$ are set equal to zero, $ymax$ and $xmax$ are set equal to $H_{UV} - 1$ and $W_{UV} - 1$ respectively.

10 **When processing the sample at (y,x), only the input samples within the boundaries of xmin, xmax, ymin and ymax are used. The region is extended at the boundaries using replication padding.**

23. (14.2) Adaptive upsampler

15 This section details the primary component guided adaptive upsampler process. This process provides enhancement of secondary components (colour information planes) of image utilising information from primary component.

This process is enabled if `EFE_upsampler_enabled_flag` is true.

The input of this process is

- 20 – $\hat{x}_Y [1, H_Y, W_Y]$ (output of synthesis transform for primary component),
- $\hat{x}_{UV} [2, H_{inUV}, W_{inUV}]$ (output of synthesis transform for secondary component),

The output of this process is

- enhanced secondary component $\hat{x}'_{UV} [2, H_{UV}, W_{UV}]$ which goes to the ICCI filter block (section 14.1) and $\hat{x}''_{UV} [2, H_{UV}, W_{UV}]$ which goes to the non-linear filter block (section 25 14.3).

Fig. 27 illustrates an example implementation of primary component guided adaptive up-sampling filter.

If `EFE_upsampler_enabled_flag` is equal to 0, the $\hat{x}'_{UV} [2, H, W]$ is up-sampled by bi-cubic interpolation as described in section 7.6. Otherwise the following ordered steps are performed:

- 30 – The parsing process according to parsing table in section 9.4.1. is invoked to obtain W_{1A} , W_{1B} , W_{4A} , W_{4B} and B_1 .
- Tiling process is as described in section 14.2.2 is invoked with parsed syntax elements as

inputs and Tile1 tensor as output.

- Parameter update process as specified in section Section 14.2.1. is invoked with W_{4A} and W_{1A} as inputs and modified W_{4A} and W_{1A} as outputs.
- $B_1[2]$ is vector is subtracted channelwise from $\hat{x}'_{UV}[2, H_{inUV}, W_{inUV}]$.
- 5 – $U\left[2 \cdot scale_{ver} \cdot scale_{hor}, \frac{H+1}{2}, \frac{W+1}{2}\right]$ and $Y\left[4, \frac{H+1}{2}, \frac{W+1}{2}\right]$ are set equal to $pixelUnshuffle(\hat{x}'_{UV}, scale_{ver}, scale_{hor})$ and $pixelUnshuffle(\hat{x}'_Y, 2, 2)$ respectively.
- for Vtile = 0...NumHorSplits-1 and Htile = 0...NumVerSplits-1
 - for $x = RC[1][Vtile][Htile][1] \dots RC[1][Vtile][Htile][3] - 1$, $y = RC[1][Vtile][Htile][0] \dots RC[1][Vtile][Htile][2] - 1$, $k = 0..1$, $i=0 \dots o_{ver} - 1$,
 10 $j=0 \dots o_{hor} - 1$ the following is performed:
 - $ch = 4k + 2i + j$
 - $ch_{out} = k \cdot o_{ver} \cdot o_{hor} + o_{hor} \cdot i + j$
 - $ch_{in} = k \cdot scale_{ver} \cdot scale_{hor} + scale_{hor} \cdot (i \% scale_{ver}) + (j \% scale_{hor})$
 - $t_{idx} = Tile1[k, y, x]$
 - 15 – If indepdent_region_flag[Vtile][Htile] equal to 1;
 - $ymin = RC[1][Vtile][Htile][0]$
 - $ymax = RC[1][Vtile][Htile][2] - 1$
 - $xmin = RC[1][Vtile][Htile][1]$
 - $xmax = RC[1][Vtile][Htile][3] - 1$
 - 20 – Else;
 - $ymin$ and $xmin$ are set equal to zero, $ymax$ and $xmax$ are set equal to $\frac{H+1}{2} - 1$ and $\frac{W+1}{2} - 1$ respectively.
 - $U[ch_{out}, y, x] = U[ch_{in}, y, x] \star W_{4A}[t_{idx}, ch] + Y[2i + j, y, x] \star W_{4B}[t_{idx}, k] + B_1[k]$
 - 25 – $U[ch_{out}, y, x] = U[ch_{in}, y, x] \star W_{1A}[0, ch] + Y[2i + j, y, x] \star W_{1B}[0, k] + B_1[k]$
 - $\hat{x}'_{UV}[2, H_{UV}, W_{UV}]$ and $\hat{x}''_{UV}[2, H_{UV}, W_{UV}]$ are set equal to $pixelshuffle(U', o_{ver}, o_{hor})$ and $pixelshuffle(U'', o_{ver}, o_{hor})$ respectively.

Where “ $\star$ ” is 2D cross-correlation operator with kernel size 4×4 : $b \star a = \sum_{j=-1}^{j=2} \sum_{i=-1}^{i=2} b[clip(ymin, ymax - 1, y + i), clip(xmin, xmax, x + j)]a[i + 1, j + 1]$.

30 24. (14.4.1) LEF general process

LEF process receives primary variance tensor $\hat{\sigma}_Y[C_p, h_{4,Y}, w_{4,Y}]$ and reconstructed primary

$\hat{x}'_Y[1, H, W]$ as inputs. The output of LEF process is edge enhanced modified $\hat{x}'_Y[1, H, W]$.

The following ordered steps are performed:

- Picture header parsing process is invoked as described in section Section 9.3. to obtain *targetBppIdx*.
- 5 – *chIdx* is set equal to *reference_channel_idx_table[targetBppIdx]*.
- The $\hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y]$ tensor is obtained as follows, for $i = 0..H_Y - 1, j = 0..W_Y - 1$;
 - $\hat{\sigma}_{Y_channel}^1[0, i, j] = \hat{\sigma}_Y[chIdx, floor(i \cdot h_{4,Y} \div H_Y), floor(j \cdot w_{4,Y} \div W_Y)]$
- *thr[3]=luma_edge_filter_thr_list_table[targetBppIdx]*
- *intensity[4]=luma_edge_filter_intensity_list[targetBppIdx]*
- 10 – $\hat{x}'_Y = \hat{x}'_Y \div 255$
- **for** *Vtile* = 0...NumHorSplits-1 and *Htile* = 0...NumVerSplits-1
 - If *independent_region_flag[Vtile][Htile]* equal to 1;
 - *ymin* = *RC[0][Vtile][Htile][0]*
 - *ymax* = *RC[0][Vtile][Htile][2] - 1*
 - 15 ▪ *xmin* = *RC[0][Vtile][Htile][1]*
 - *xmax* = *RC[0][Vtile][Htile][3] - 1*
 - Else;
 - *ymin* and *xmin* are set equal to zero, *ymax* and *xmax* are set equal to $\frac{H+1}{2} - 1$ and $\frac{W+1}{2} - 1$ respectively.
 - 20 ○ **for** *x* = *RC[0][Vtile][Htile][1] ... RC[0][Vtile][Htile][3] - 1*, *y* = *RC[0][Vtile][Htile][0] ... RC[0][Vtile][Htile][2] - 1*, *k* = 0..1, *i*=0...*o_{ver}*-1, *j*=0...*o_{hor}* - 1 the following is performed:
 - $\hat{x}_{blur}[0, y, x] = \hat{x}'_Y \star kernel$

Where “ $\star$ ” is 2D cross-correlation operator with kernel size 3×3 : $b \star a =$

$$25 \sum_{j=-1}^{j=2} \sum_{i=-1}^{i=2} b[clip(ymin, ymax - 1, y + i), clip(xmin, xmax, x + j)] a[i + 1, j + 1].$$

- $\hat{x}_{edge} = \hat{x}'_Y - \hat{x}_{blur}$
- $\hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y] = \hat{\sigma}_{Y_channel}^1[1, H_Y, W_Y] \div 8190$
- For $i = 0..H_Y-1, j = 0..W_Y-1$;

$$- \quad tIdx = \begin{cases} 0, & \hat{\sigma}_{Y_{channel}}^1[0, i, j] < thr[0] \\ 1, & thr[0] \leq \hat{\sigma}_{Y_{channel}}^1[0, i, j] < thr[1] \\ 2, & thr[1] \leq \hat{\sigma}_{Y_{channel}}^1[0, i, j] < thr[2] \\ 3, & thr[2] \leq \hat{\sigma}_{Y_{channel}}^1[0, i, j] \end{cases}$$

$$- \quad \hat{Y}[i, j] = \hat{x}_{edge}[0, i, j] * intensity[tIdx] + \hat{x}_{blur}[0, i, j]$$

– $\hat{Y}$ is modified as follows, for $i = 0..H_Y-1, j = 0..W_Y-1$:

$$- \quad \hat{Y}[i, j] = clip(0, 1, \hat{Y}[i, j]) \cdot 255$$

5 In this process zero padding is used when the indices of a tensor exceeds the tensor boundaries.

The weights tensor *kernel* is defined as follows:

$$kernel = \begin{bmatrix} 1 \div 13 & 1 \div 13 & 1 \div 13 \\ 1 \div 13 & 5 \div 13 & 1 \div 13 \\ 1 \div 13 & 1 \div 13 & 1 \div 13 \end{bmatrix}$$

The *reference_channel_idx_table* is defined as follows:

$$10 \quad \quad \quad reference_channel_idx_table = [23, 23, 65, 65, 65]$$

The *luma_edge_filter_intensity_list* is as follows:

<i>luma_edge_filter_intensity_list</i> [X][Y]	Y=0	Y=1	Y=2	Y=3
X=0	1.00	1.10	1.25	1.40
X=1	1.00	1.08	1.12	1.16
X=2	1.00	1.05	1.06	1.07
X=3	1.00	1.02	1.03	1.04
X=4	1.00	1.01	1.02	1.03

The *luma_edge_filter_thr_list_table* is defined as follows:

<i>luma_edge_filter_thr_list_table</i> [X][Y]	Y=0	Y=1	Y=2
X=0	0.3	2.0	6.0
X=1	0.4	3.0	7.0
X=2	0.5	6.0	10.0
X=3	0.6	7.0	12.0
X=4	0.7	8.0	14.0

5.2 Embodiment 2

15 Alternatively or additionally to the embodiment 1, the following might be a further implementation of the present disclosure.

1. (9.3.1) Syntax table

picture_header() {	Descriptor
PIH	u(16)
picture_header_size	u(16)
img_width	u(16)
img_height	u(16)
picture_format	u(2)
bit_depth	u(1)
bit_c_ver	u(1)
bit_c_hor	u(1)
bit_s_ver	u(1)
bit_s_hor	u(1)
num_ver_splits_minus1	u(7)
num_hor_splits_minus1	u(7)
if(NumVerSplits > 1 NumHorSplits > 1) {	
regional_access_enabled_flag	u(1)
if(regional_access_enabled_flag)	
for(i = 0; i < NumVerSplits; i++)	
for(j = 0; j < NumHorSplits; j++)	
independent_region_flag[i][j]	u(1)
}	
num_threads_q_minus1	u(7)
num_threads_z_minus1	u(7)
num_threads_r_primary_minus1	u(7)
num_threads_r_secondary_minus1	u(7)
q_ls	u(8)
z_ls	u(8)
r_primary_ls	u(8)
q_stream_ls	u(8)
color_transform_header()	u(1)
model_header()	
}	

q_ls indicates the number of bytes used to code the size of the **q_stream**.

z_ls indicates the number of bytes used to code the size of the **z_stream**.

r_primary_ls indicates the number of bytes used to code the size of the **r_primary_stream**.

r_secondary_ls indicates the number of bytes used to code the size of the **r_secondary_stream**.

2. (9.5.1.1) Syntax table of hyper tensor

z_stream_for_one_region(substrIdx) {	Descriptor
z_stream_size	u(8*z_ls)
for (i =1; i <= num_threads_z_minus1; ++i) {	
thread_offsets_z[i]	u(24)
}	
for (i =0; i <= num_threads_z_minus1; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_z_minus1+1, thread_offsets_z, z_stream_size)	
}	
for (ch =0; ch < z_primary_ch; ++ch) {	
for (i =0; i <= num_threads_z_minus1; ++i) {	
decode_z_data(sub_stream[i], z_primary[substrIdx][ch][], ch, RC[6][i][j][5], RC[6][i][j][4], i, num_threads_z_minus1+1)	
}	
}	
for (ch =0; ch < z_secondary_ch; ++ch) {	
for (i =0; i <= num_threads_z_minus1; ++i) {	
decode_z_data(sub_stream[i], z_secondary[substrIdx][ch][], ch, RC[6][i][j][5], RC[6][i][j][4], i, num_threads_z_minus1+1)	
}	
}	
}	

5 3. (9.5.2.1) Syntax table of quality map information tensor

q_stream_for_one_region(substrIdx)	Descriptor
q_stream_size	u(8*q_ls)
for (i =1; i <= num_threads_q_minus1; ++i) {	
thread_offsets_q[i]	u(24)
}	
for (i =0; i <= num_threads_q_minus1; ++i) {	

sub_stream[i]=sub_stream_init(i, num_threads_q_minus1+1, thread_offsets_q, q_stream_size)	
}	
for (i =0; i <= num_threads_q_minus1; ++i) {	
decode_r_data(sub_stream[i], q_primary[substrIdx][], q_primary_sigma_idx_2D[substrIdx][], i, num_threads_q_minus1+1)	
}	
for (i =0; i <= num_threads_q_minus1; ++i) {	
decode_r_data(sub_stream[i], q_secondary[substrIdx][], q_secondary_sigma_idx_2D [substrIdx][], i, num_threads_q_minus1+1)	
}	
}	

4. (9.5.3.1) Syntax table of primary residual tensor

r_primary_stream_for_one_region(substrIdx) {	Descriptor
r_primary_stream_size	u(8*r_prim ary_ls)
for (i =1; i <= num_threads_r_primary_minus1; ++i) {	
thread_offsets_r_primary[i]	u(32)
}	
for (i=0; i <= num_threads_r_primary_minus1; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_r_primary_minus1+1, thread_offsets_r_primary, r_primary_stream_size)	
decode_r_data(sub_stream[i], r_primary[substrIdx][], sigma_idx_primary_2D[substrIdx][], i, num_threads_r_primary_minus1+1)	
}	
}	

5. (9.5.4.1) Syntax table of secondary residual tensor

r_secondary_stream_for_one_region (substrIdx) {	Descriptor
r_secondary_stream_size	u(8*r_secon dary_ls)
for (i =1; i <= num_threads_r_secondary_minus1; ++i) {	

thread_offsets_r_secondary[i]	u(24)
}	
for (i=0; i <= num_threads_r_secondary_minus1; ++i) {	
sub_stream[i]=sub_stream_init(i, num_threads_r_secondary_minus1+1, thread_offsets_r_secondary, r_secondary_stream_size)	
decode_r_data(sub_stream[i], r_secondary[substrIdx] [], sigma_idx_secondary_2D[substrIdx] [], i, num_threads_r_secondary_minus1+1)	
}	
}	

[0054] More details of the embodiments of the present disclosure will be described below which are related to neural network-based visual data coding. As used herein, the term “visual data” may refer to an image, a picture in a video, or any other visual data suitable to be coded. To solve the above problems and some other problems not mentioned, visual data processing solutions as described below are disclosed. The embodiments of the present disclosure should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these embodiments can be applied individually or combined in any manner.

[0055] Fig. 28 illustrates a flowchart of a method 2800 for visual data processing in accordance with some embodiments of the present disclosure. At 2802, a conversion between the visual data and a codestream of the visual data is performed with a neural network (NN)-based model. For example, a codestream may comprise a sequence of bits. In addition, the codestream may further comprise associated codes which are used as markers. As used herein, the codestream may also be referred to as a bitstream.

[0056] In some embodiments, the conversion may include encoding the visual data into the codestream. Additionally or alternatively, the conversion may include decoding the visual data from the codestream. By way of example rather than limitation, the decoding model shown in Fig. 6 may be employed for decoding the visual data from the bitstream.

[0057] As used herein, an NN-based model may be a model based on neural network technologies. For example, an NN-based model may specify sequence of neural network modules (also called architecture) and model parameters. The neural network module may comprise a set of neural network layers. Each neural network layer specifies a tensor operation which receives and outputs tensor, and each layer has trainable parameters. It should be understood that the possible implementations of the NN-based model described

here are merely illustrative and therefore should not be construed as limiting the present disclosure in any way.

[0058] In addition, a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions. The
5 codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region. For example, each subset of residual samples corresponds to a set of samples covering a region of the visual data, and the region may be rectangular. In some embodiments, the set of residual samples may be a residual tensor, and each of the plurality of subsets of residual
10 samples may be a residual sub-tensor.

[0059] By way of example, the first indication equal to a first value (such as 1, or the like) may indicate that primary or secondary residual data for each region is in a substream. That is, residual data for different regions are encapsulated in different substreams. In this case, a region may be coded independently from other regions. the first indication equal to a
15 second value (such as 0, or the like) may indicate that there is only one substream for primary residual data and only one substream for secondary residual data. That is, residual data for a component of different regions are encapsulated in the same substream.

[0060] In view of the above, the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that
20 corresponds to the region. Compared with the conventional solution lacking such an indication, the proposed method can better support independent coding of different regions, and thus the coding efficiency can be improved.

[0061] In some embodiments, if the first indication indicates that is only one substream for primary residual samples for the plurality of regions and only one substream for secondary
25 residual samples for the plurality of regions, for a processing with a first processing module in the NN-based model, the plurality of regions may be allowed to overlap. By way of example, the first processing module may comprise at least one of the following: a hyper decoding module, a multistage context modelling (MCM) module, a synthesis transform module, a latent sample reconstruction module, a sample prediction module, an analysis
30 transform module, a hyper scale decoding module, or the like.

[0062] In some embodiments, the plurality of regions may comprise a first region, and at least one padding amount for the first region may be determined based on an overlap amount between the first region and one or more regions neighboring to the first region. For example, the at least one padding amount may comprise a padding amount in a left direction, a padding

amount in a right direction, a padding amount in a bottom direction, a padding amount in a top direction, and/or the like.

[0063] In some embodiments, the number of samples to be discarded from an output of the first processing module may be determined based on the at least one padding amount. For example, in a process of merging samples of a set of regions into an output tensor of the first processing module, several samples of each region may be omitted. By way of example, 9 regions, each of a size of 14×14 , may be merged into an output tensor of 30×30 . In this case, 2 samples at left, 2 samples at right, 2 samples at top, and 2 samples at bottom may be discarded for each region. It should be understood that the specific values recited herein are intended to be examples rather than limiting the scope of the present disclosure.

[0064] In some embodiments, an overlap amount between two of the plurality of regions may be dependent on the first processing module. For example, the codestream may further comprise at least one of the following: an indication indicating the overlap amount or a padding amount used in a hyper decoding module, or an indication indicating the overlap amount or a padding amount used in an MCM module. In some embodiments, the overlap amount may be different for different processing modules.

[0065] In some embodiments, if the first indication indicates that residual samples for each of the plurality of regions are in a substream of the codestream, for a processing with a second processing module in the NN-based model, the plurality of regions do not overlap with each other. By way of example, the second processing module may comprise at least one of the following: a synthesis transform module, an analysis transform module, an MCM module, a sample prediction module, a latent sample reconstruction module, a hyper decoding module, a hyper scale decoding module, or the like. In this case, residual samples for each of the plurality of regions may be coded independently from residual samples for the rest of the plurality of regions.

[0066] In some embodiments, if the first indication indicates that residual samples for each of the plurality of regions are in a substream of the codestream, for a processing with a synthesis transform module, at least one region among the plurality of regions may be partitioned into a plurality of tiles, and the number of the plurality of tiles may be an integer. For example, a tile within a region may be allowed to overlap with a further tile within the same region and may be not allowed to overlap with a tile within a further region. In other words, tiles comprised in a same region may overlap while tiles from different are not allowed to overlap. Thereby, it can be ensured that a region can be coded without samples from other regions, and thus the independent coding can be ensured.

[0067] In some embodiments, if the first indication indicates that residual samples for each of the plurality of regions are in a substream of the codestream, the codestream may comprise a plurality of substreams, and each of the plurality of substreams corresponds to one of the plurality of regions. For example, residual data of different regions may be encapsulated into different substreams. In this case, samples corresponding to one of the plurality of regions may be reconstructed independently from substreams corresponding to the rest of the plurality of regions. In some embodiments, a substream corresponding to one of the plurality of regions may be allowed to be discarded. For example, a decoder may read the marker ID in a substream and if the decoder doesn't support the marker, the decoder may discard the remaining bytes of the substream. In this case, the discarding of a substream will not affect the correct coding of the other substreams, since they are coded independently.

[0068] In some embodiments, data corresponding to one of the plurality of regions may be allowed to be extracted from the codestream. In this case, at least one of the following may be indicated in the codestream: whether to extract the data from the codestream, or how to extract the data from the codestream. For example, the data may comprise a substream, quality map information, residual data, luma residual data, chroma residual data, hyper latent data, residual latent data, and/or the like. It should be noted that discarding a part of data may also be regarded as extracting the rest of the data.

[0069] In some embodiments, the codestream comprising a second indication of a region indicates that residual data for the region is present in the codestream. For example, the residual data may comprise residual samples for a primary component and/or residual samples for a secondary component. By way of example, the second indication may comprise an index, a flag or the like. In addition, the codestream may further comprise an indication indicating a length of the second indication. In some alternative embodiments, the codestream may comprise a list of indices, wherein each entry of the list may indicate the regions that are absent from the codestream, or each entry of the list may indicate the regions that are present in the codestream.

[0070] In some embodiments, substreams of different regions among the plurality of regions may be coded using an entropy coder in parallel. In some embodiments, the codestream may further comprise at least one of the following: an indication indicating the number of threads for coding residual codestream for a primary component, or an indication indicating the number of threads for coding residual codestream for a secondary component. As used herein, a thread may be a parallel processing unit.

[0071] In some embodiments, a correspondence between an index of a substream and a region

may be determined based on a raster scanning order of the plurality of regions. In some alternative embodiments, the codestream may further comprise an indication indicating a correspondence between an index of a substream and a region.

[0072] In some embodiments, the codestream may further comprise a third indication indicating a size of the substream. For example, the size of the substream may be represented by the number of bytes. Alternatively, the size of the substream may be represented by the number of bits. In one example embodiment, the third indication may be coded using variable length coding. Alternatively, the third indication may be coded using fixed length coding, or predictive coding or the like.

[0073] In some embodiments, the maximum number of the plurality of regions may be dependent on a profile to which the codestream conforms, a level to which the codestream conforms, a size of the visual data, and/or the like. Additionally or alternatively, a maximum size or a minimum size of one of the plurality of regions may be dependent on a profile to which the codestream conforms, a level to which the codestream conforms, a size of the visual data, the number of the plurality of regions, and/or the like.

[0074] In some embodiments, except for regions at a right boundary or a bottom boundary, the rest of the plurality of regions may be of a same size. For example, the right boundary and/or the bottom boundary may be a boundary of the visual data. The left subpicture of Fig. 19 illustrates an example of this case.

[0075] In some embodiments, the plurality of regions cover the entire visual data. In some further embodiments, a vertical size of one of the plurality of regions may be equal to a product of k_{ver} and N_{ver} , where k_{ver} may be an integer and N_{ver} may be a predetermined integer. Additionally or alternatively, a horizontal size of the region may be equal to a product of k_{hor} and N_{hor} , where k_{hor} may be an integer and N_{hor} may be a predetermined integer. In one example embodiment, k_{ver} may be determined based on the number of horizontal splits of the set of residual samples, k_{hor} may be determined based on the number of vertical splits of the set of residual samples, and/or each of N_{ver} and N_{hor} may be equal to 128.

[0076] In some embodiments, the codestream may further comprise at least one of the following: an indication indicating the number of regions for which a substream is present in the codestream, an indication indicating the number of regions for which a substream is absent from the codestream, an indication indicating whether substreams for all of the plurality of regions are included in the codestream, an indication indicating whether a substream of a region is discarded from the codestream, or an indication indicating whether

a substream of a region is extracted from the codestream.

[0077] In some embodiments, data corresponding to different regions among the plurality of regions may be allowed to be coded using an entropy coder in parallel. By way of example, the data may comprise at least one of the following: information for a primary component, information for a secondary component, residual information, hyper latent information, or quality map information. In addition, the codestream may further comprise a fourth indication indicating the number of parallel processing units. By way of example, the number of parallel processing units may be allowed to be different for different data to be processed in parallel.

[0078] In some embodiments, the number of parallel processing units and/or the maximum number of parallel processing units may be determined based on a profile to which the codestream conforms, a level to which the codestream conforms, a size of the visual data, the number of the plurality of regions, and/or the like.

[0079] In some embodiments, the plurality of regions may be centrally aligned or shifted.

This has been discussed in detail above with reference to Figs. 20A-20C. Moreover, the codestream may further comprise an indication indicating whether region coordinates are shifted.

[0080] In some embodiments, for processing a region, a tensor used as an input to a third processing module may be centrally aligned with a tensor used as an input to a fourth processing module. Alternatively, the tensor used as the input to the third processing module may be shifted with respect to the tensor used as the input to the fourth processing module. In one example embodiment, the processing of the region with a synthesis transform may be performed with centrally aligned tensors, and the processing of the region with a further processing module may be performed with non-aligned tensors.

[0081] In a further example embodiment, the processing of the region with a synthesis transform may be performed with centrally aligned tensors, and the processing of the region with a further processing module may be performed with centrally aligned tensors or non-aligned tensors. In addition, the codestream may further comprise an indication indicating whether the processing of the region with the further processing module is performed with the centrally aligned tensors or the non-aligned tensors. For example, the further processing module may comprise at least one of the following: a hyper decoding module, a hyper scale decoding module, an entropy decoder a sample prediction module, an MCM module, or a latent sample reconstruction module.

[0082] In some embodiments, the number of the plurality of regions may be no larger than a

predetermined integer. Additionally or alternatively, the number of rows of regions among the plurality of regions may be no later than a predetermined integer. Additionally or alternatively, the number of columns of regions among the plurality of regions may be no later than a predetermined integer. In some embodiments, an indication related to region-

5 based coding may be coded in a predictive way.

[0083] In some embodiments, whether a first coding mode is enabled is determined for the conversion between the visual data and the bitstream. In one example embodiment, the visual data may be at least a part of a picture of a video. Alternatively, the visual data may be at least a part of an image. In some embodiments, the bitstream may comprise an
10 indication indicating whether the first mode is enabled. In this case, at a decoder, information regarding whether the first mode is enabled may be parsed from the bitstream.

[0084] In the first coding mode, a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples, and a subset of residual samples among the plurality of subsets of residual samples is coded before a further subset of residual
15 samples among the plurality of subsets of residual samples. For example, a subset of residual samples at the first position (such as a top-left position) may be coded before the rest of the plurality of subsets of residual samples.

[0085] In some embodiments, a residual sample associated with the visual data may be a sample in a residual latent representation of the visual data. The residual latent
20 representation may indicate a difference between a latent representation of the visual data and a prediction of the latent representation. As used herein, the term “latent representation” may refer to an intermediate representation of the visual data during the conversion process. By way of example rather than limitation, the latent representation may comprise a latent tensor or latent for short. Correspondingly, the residual latent representation may comprise
25 a residual latent tensor or residual tensor for short. In this case, a subset of residual samples may be regarded as a sub-tensor of the residual tensor. It should be noted that the residual sample may also be a sample indicating difference in a pixel domain rather than in the above-mentioned latent domain.

[0086] By way of example rather than limitation, in an entropy decoding process where the
30 bitstream is converted to transformed coefficients (such as the residual samples), in aid of the first coding mode, the transformed coefficients of a region can be decoded before other regions.

[0087] In view of the above, in the first coding mode, a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples, and a subset

of residual samples among the plurality of subsets of residual samples is coded before a further subset of residual samples among the plurality of subsets of residual samples. Compared with the conventional raster-scan coding order, the proposed method can advantageously make it possible to code a set of residual samples without waiting for coding a further residual sample that is not comprised in the set of residual samples. Thereby, the proposed method can advantageously support coding different subsets of residual samples independently, and thus the coding efficiency can be improved.

[0088] In some embodiments, the first coding mode may be enabled for the conversion. In this case, whether a second coding mode is enabled for the conversion may be determined and the conversion is performed based on the determining of whether the second mode is enabled. In the second coding mode, a set of samples associated with the visual data may be partitioned into a plurality of subsets of samples, and a first subset of samples among the plurality of subsets of samples may be coded independently from the rest of the plurality of subsets of samples. For example, the set of samples may comprise one of the following: samples in the visual data, samples in a latent representation of the visual data, samples in a residual latent representation of the visual data, or the like. As used herein, a sample in the latent representation may also be referred to as a latent sample.

[0089] In one example embodiment, samples associated with the visual data and samples in the latent representation of the visual data may be partitioned in a same manner as the set of residual samples associated with the visual data. In this case, a subset of samples in the visual data may corresponds to a subset of samples in a latent representation of the visual data, and in turn, the subset of samples in a latent representation of the visual data may correspond to a subset of samples in a residual latent representation, i.e., a subset of residual samples. In other words, there may a correspondence between samples in a pixel domain and samples in a latent domain (aka. transformed domain), as shown in Fig. 2. It should be noted that they may also be partitioned in different manners. The scope of the present disclosure is not limited in this respect.

[0090] In some embodiments, the bitstream may comprise an indication indicating whether the second coding mode may be enabled. In this case, at a decoder, information regarding whether the second mode is enabled may be parsed from the bitstream.

[0091] By way of example rather than limitation, the decoding process may further comprise a sample reconstruction process in addition to the above-mentioned entropy decoding process. The sample reconstruction process may comprise a latent sample reconstruction process and a synthesis transform. In addition, the latent sample reconstruction process may

further comprise a latent sample prediction process and a latent sample compensation process. For example, latent samples are predicted in the latent sample prediction process, and then reconstructed latent samples are determined based on the predicted latent samples and the residual samples. Furthermore, a synthesis transform may be applied on the reconstructed latent samples to obtain reconstructed samples of the visual data which are in a pixel domain. In aid of the above-mentioned second coding mode, residual samples, latent samples and samples in the pixel domain of a region can be decoded independently from other regions. Thereby, the coding process is more flexible and more efficient.

[0092] In some embodiments, bits associated with the set of samples may be organized in the bitstream based on the partition of the plurality of subsets of samples. In one example embodiment, bits associated with a first component of the first subset of samples may be coded before bits associated with a first component of a second subset of samples among the plurality of subsets of samples. In addition, bits associated with a second component of the first subset of samples may be coded after bits associated with the first component of the second subset of samples. Alternatively, bits associated with all components of the first subset of samples may be coded before bits associated with all components of a second subset of samples among the plurality of subsets of samples. In addition, bits associated with a first component of the first subset of samples may be coded before bits associated with a second component of the first subset of samples.

[0093] In some embodiments, bits associated with all components of the first subset of samples may be group together in a substream of the bitstream. For example, bits for a region shall not be separated by a bit for a further region. By way of example rather than limitation a bitstream may be organized as follows: (1) bits for luma residual samples of a first region; (2) bits for chroma residual samples of the first region; (3) bits for luma residual samples of a second region; (4) bits for chroma residual samples of the second region, etc. In addition, bits for luma and chroma residual samples of the first region may be encapsulated in a substream, and bits for luma and chroma residual samples of the second region may be encapsulated in a further substream.

[0094] In some embodiments, at least one indication in the bitstream indicates whether bits associated with a first component or a second component of each of the plurality of subsets of samples are in a substream of the bitstream. By way of example rather than limitation, a syntax element in the bitstream may indicate whether the primary or secondary residual data for each region is in a substream.

[0095] For example, this at least one indication may be used as an indication of the second

coding mode. If the at least one indication indicates that bits associated with a first component or a second component of each subset of the plurality of subsets of samples are in a substream of the bitstream, the second coding mode is enabled. Otherwise, the second coding mode is disabled.

5 [0096] In some embodiments, the bitstream may comprise a plurality of substreams corresponding to the plurality of subsets of samples, each of the plurality of substreams may comprise bits associated with a corresponding subset of samples. In addition, there may be a marker at the beginning of at least one of the plurality of substreams. For example, a marker may be implemented as a code with one or more bytes, such as a one-byte code, a
10 two-byte code, or the like. It should be noted that a bitstream comprising one or more markers may also be referred to as a codestream.

[0097] In some alternative embodiments, the bitstream may comprise a plurality of substreams corresponding to the plurality of subsets of samples, each of the plurality of substreams may comprise bits associated with a corresponding subset of samples. In addition,
15 there may be a marker at the beginning of each of the plurality of substreams. In some alternative embodiments, there may be a marker at the beginning of bits associated with hyper tensor for a region.

[0098] In some embodiments, the bitstream may comprise a plurality of substreams corresponding to the plurality of subsets of samples. Each of the plurality of substreams may
20 comprise bits associated with a component of a corresponding subset of samples, and there may be a marker at the beginning of each of the plurality of substreams.

[0099] In some embodiments, a first component of the set of samples may be partitioned into a first number of subsets, a second component of the set of samples may be partitioned into a second number of subsets, and the first number may be equal to the second number. For
25 example, the first number and the second number may be indicated by at least one indication in the bitstream. In one example embodiments, the set of samples may be partitioned horizontally and vertically. In this case, the number of vertical splits of the set of samples may be indicated in the bitstream. By way of example rather than limitation, an indication in the bitstream may indicate the number of vertical splits of the set of samples minus 1.
30 Additionally or alternatively, the number of horizontal splits of the set of samples may be indicated in the bitstream. By way of example rather than limitation, an indication in the bitstream may indicate the number of horizontal splits of the set of samples minus 1. In this case, the first number and the second number may be equal to a product of the number of vertical splits and the number of horizontal splits. It should be noted that the first number

and the second number may also be different from each other.

[0100] In some embodiments, at least one of the following may be constrained depending on a profile and a level to which the bitstream conforms: the maximum value of the number of vertical splits of the set of samples, the minimum value of the number of vertical splits of the set of samples, the maximum value of the number of horizontal splits of the set of samples, or the minimum value of the number of horizontal splits of the set of samples.

[0101] In some embodiments, the plurality of subsets of samples corresponds to a plurality of regions, and each of the plurality of regions may comprise a corresponding subset of samples. Alternatively, each of the plurality of subsets of samples corresponds a subpicture, a tile, a slice, or the like. For ease of discussion, the case where each of the plurality of subsets of samples corresponds a region will be taken as an example and described in detail below. It should be noted that the concept described below may also be applied to a case where each of the plurality of subsets of samples corresponds a subpicture, a tile, a slice, or the like. The scope of the present disclosure is not limited in this respect.

[0102] In some embodiments, a position of each of the plurality of regions may be constrained. For example, a position of a top-left sample (e.g., a top-left luma sample) of a region shall be located at $(2^X, 2^Y)$ relative to a top-left sample of the set of samples, and each of X and Y may be a non-negative integer, such as 4, 5, or 6 the like.

[0103] In some additional embodiments, the minimal number of samples of a first component comprised in a region may be constrained. Additionally or alternatively, the minimal number of samples of a second component comprised in a region may be constrained. By way of example rather than limitation, the number of samples of a first or second component comprised in a region shall be no smaller than a predetermined number, such as 64, 128, 128^2 , or the like.

[0104] In some further embodiments, the maximal number of samples of a first component comprised in a region may be constrained. Additionally or alternatively, the maximal number of samples of a second component comprised in a region may be constrained.

[0105] In some embodiments, a region size may be a multiple of M samples, and M may be a non-negative integer. For example, M may be equal to 2^N , and N may be a non-negative integer, such as 4, 5, 6 or the like. By way of example rather than limitation, M may be equal to 128. It should be understood that the specific values recited herein are intended to be exemplary rather than limiting the scope of the present disclosure.

[0106] In some embodiments, all regions that are not located at a right boundary or a bottom boundary of the visual data have the above-mentioned region size. In other words, regions

that are located at a right boundary or a bottom boundary of the visual data are allowed to have a size different from the above-mentioned region size due to the size of the visual data and the partitioning scheme.

[0107] In some embodiments, a vertical coordinate of a region, a horizontal coordinate of the region, and/or a size of the region may be determined based on a size of the visual data and a depth parameter. For example, the size of the visual data may comprise at least one of a width or a height of the visual data. A set of equations for this purposed are listed in the above section 1.13.2, where the variable d represents the depth parameter.

[0108] By way of example rather than limitation, the NN-based model may perform 6 times upsampling operation during the decoding process, in each time, the size of the tensor that is being processed will be doubled. In order to make sure independent decoding is achieved, it is necessary to process tensors in each time independently, which means the coordinates of the regions at each upsampling step shall be determined. Hence, the region sizes/coordinates after each upsampling step may be determined. Then, these coordinates may be used to independently process each tensor. In this case, the depth parameter may be in a range from 0 to 5. A depth parameter being equal to 0 may indicate the first upsampling operation, i.e., the first upsampling step. A depth parameter being equal to 1 may indicate the second upsampling operation, i.e., the second upsampling step, and so on. It should be understood that the above illustrations are described merely for purpose of description. The scope of the present disclosure is not limited in this respect.

[0109] In some embodiments, a fifth indication of a regional access capability may be indicated in the bitstream. For example, the fifth indication may be comprised in a picture header syntax table in the bitstream, or any other suitable syntax table. By way of example rather than limitation, the regional access capability may comprise a capability of each of the plurality of regions to be correctly coded independently from other regions. Additionally or alternatively, the regional access capability may comprise a capability of each of the plurality of regions to be coded independently from other regions. In this case, there may be an “allowed deviation” from a correct reconstruction. In other words, a reconstruction with a deviation from correct reconstruction being smaller than a threshold may also be deemed as “conforming to the standard”.

[0110] For example, this fifth indication may be used as an indication of the second coding mode. If the fifth indication indicates that the regional access capability is enabled, the second coding mode is enabled. Otherwise, the second coding mode is disabled.

[0111] In some embodiments, if the fifth indication indicates the regional access capability is

enabled, each sample on a region boundary is also on a tile boundary, while a tile boundary may not be a region boundary. In other words, if the fifth indication indicates the regional access capability is enabled, a tile may be a part of a region.

[0112] In some embodiments, if the fifth indication indicates the regional access capability is enabled, a size of tiles for a first component of a region may be the same as a size of tiles for a second component of the region. For example, tiles for luma and chroma components may be aligned.

[0113] In some embodiments, if the fifth indication indicates the regional access capability is enabled, an amount of overlap for processing a region boundary is set equal to zero. For example, if the fifth indication indicates the regional access capability is enabled, a region may be processed without using a sample of other regions. Thereby, compared with a solution where a non-zero overlap is involved in processing a region boundary, the proposed solution can advantageously make it possible to code a region independently from neighboring regions.

[0114] In some alternative embodiments, if the fifth indication indicates the regional access capability is enabled, a padding operation may be applied to a region boundary. For example, a padding amount for the padding operation depends on a size of a region, a modulo of the size of the region, and/or the like. In some embodiments, the padding operation may comprise repetitive padding or padding with a constant value, such as 0, 1, 2 or the like. In some embodiments, if a size of a region is a multiple of a predetermined number (such as 64, 128 or the like), no padding operation may be performed for the region.

[0115] In some embodiments, at least one of the following may be performed on a region independently from other regions: a process of entropy coding, a process of sample prediction, a process of sample reconstruction, a synthesis transform, a multi-stage context model, a filtering process, or a hyper decoder.

[0116] In some embodiments, the NN-based model may comprise one or more NN-based modules for performing at least one of the following processes on a region independently from other regions: a process of sample prediction, or a process of sample reconstruction. For example, the one or more NN-based modules may comprise at least one of a hyper decoder or a multistage context modelling (MCM) module or a synthesis transform module. For example, an output of the hyper decoder may comprise an explicit prediction tensor. Additionally or alternatively, an output of the MCM module may comprise a reconstructed latent tensor. In addition, the one or more NN-based modules may be performed on each of the plurality of regions, e.g., for one time or multiple times.

[0117] In some embodiments, the NN-based model may comprise an entropy coder for obtaining residuals corresponding to a region independently from other regions. For example, the entropy coder may comprise an arithmetic coder, an asymmetric numeral system, or the like.

5 [0118] In some embodiments, an indicator may be signaled after all bits for all components of a region before coding a further region. Alternatively, an indicator may be signaled before all bits for all components of a region. In some further embodiments, an indicator may be signaled before all bits for one component of a region. Alternatively, an indicator may be signaled after all bits for one component of a region.

10 [0119] In some embodiments, in the first coding mode, a filtering process may be performed on a region independently from other regions. For example, the filtering process may comprise a convolution operation, a padding operation, and/or the like.

[0120] In some embodiments, a probability parameter of a region may be obtained independently of other regions. For example, probability parameters (e.g., variance
15 parameters, gaussian sigma parameters etc.) corresponding to a region may be used to decode the samples of only one region and not used to decode the samples of a second region.

[0121] In some embodiments, first information regarding at least one of the following may be indicated in the bitstream: whether to apply the method, or how to apply the method. For example, the first information may be indicated at a block level, a sequence level, a group of
20 pictures level, a picture level, a slice level, a tile group level, and/or the like.

[0122] In some embodiments, the first information may be indicated in one of the following: a coding structure of a coding tree unit (CTU), a coding structure of a coding unit (CU), a coding structure of a transform unit (TU), a coding structure of a prediction unit (PU), a coding structure of a coding tree block (CTB), a coding structure of a coding block (CB), a
25 coding structure of a transform block (TB), a coding structure of a prediction block (PB), 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.

[0123] In some embodiments, the first information may be dependent on coded information
30 of the visual data. By way of example rather than limitation, the coded information may comprise a block size, a color format, a single tree partitioning, a dual tree partitioning, a color component, a slice type, a picture type, and/or the like.

[0124] In some embodiments, any of the above-mentioned indication may be a syntax element. For example, the syntax element may be binarized as one of the following: a flag, a fixed

length code, an exponential Golomb (EG) code, a unary code, a truncated unary code, or a truncated binary code. In addition, the syntax element may be coded with at least one context model. Alternatively, the syntax element may be bypass coded. In some embodiments, the syntax element may be signaled based on a condition.

5 [0125] In some embodiments, the syntax element may be indicated at one of the following: a block level, a sequence level, a group of pictures level, a picture level, a slice level, or a tile group level. In some embodiments, the syntax element may be indicated in one of the following: a coding structure of a coding tree unit (CTU), a coding structure of a coding unit (CU), a coding structure of a transform unit (TU), a coding structure of a prediction unit
10 (PU), a coding structure of a coding tree block (CTB), a coding structure of a coding block (CB), a coding structure of a transform block (TB), a coding structure of a prediction block (PB), 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
15 group header.

[0126] In view of the above, the solutions in accordance with some embodiments of the present disclosure can advantageously support coding different subsets of residual samples independently, and thus the coding efficiency can be improved.

[0127] It should be noted that the above-described concept may also be applied to any other
20 image/video compression solutions with NN-based coding tools involved. The scope of the present disclosure is not limited in this respect.

[0128] 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 codestream of visual data which is generated by a method
25 performed by an apparatus for visual data processing. The method comprises: performing a conversion between the visual data and the codestream with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the
30 plurality of regions are in a substream of the codestream that corresponds to the region.

[0129] According to still further embodiments of the present disclosure, a method for storing codestream of visual data is provided. The method comprises: performing a conversion between the visual data and the codestream with a neural network (NN)-based model; and storing the codestream in a non-transitory computer-readable recording medium, wherein a

set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

5 [0130] 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.

[0131] Clause 1. A method for visual data processing, comprising: performing a conversion between visual data and a codestream of the visual data with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a
10 plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

[0132] Clause 2. The method of clause 1, wherein the set of residual samples is a residual tensor, and each of the plurality of subsets of residual samples is a residual sub-tensor.

15 [0133] Clause 3. The method of any of clauses 1-2, wherein if the first indication indicates that there is only one substream for primary residual samples for the plurality of regions and only one substream for secondary residual samples for the plurality of regions, for a processing with a first processing module in the NN-based model, the plurality of regions are allowed to overlap.

20 [0134] Clause 4. The method of clause 3, wherein the first processing module comprises at least one of the following: a hyper decoding module, a multistage context modelling (MCM) module, a synthesis transform module, a latent sample reconstruction module, a sample prediction module, an analysis transform module, or a hyper scale decoding module.

[0135] Clause 5. The method of any of clauses 3-4, wherein the plurality of regions comprise
25 a first region, and at least one padding amount for the first region is determined based on an overlap amount between the first region and one or more regions neighboring to the first region.

[0136] Clause 6. The method of clause 5, wherein the at least one padding amount comprises at least one of the following: a padding amount in a left direction, a padding amount in a
30 right direction, a padding amount in a bottom direction, or a padding amount in a top direction.

[0137] Clause 7. The method of any of clauses 5-6, wherein the number of samples to be discarded from an output of the first processing module is determined based on the at least one padding amount.

[0138] Clause 8. The method of any of clauses 3-7, wherein an overlap amount between two of the plurality of regions is dependent on the first processing module.

[0139] Clause 9. The method of clause 8, wherein the codestream further comprises at least one of the following: an indication indicating the overlap amount or a padding amount used in a hyper decoding module, or an indication indicating the overlap amount or a padding amount used in an MCM module.

[0140] Clause 10. The method of any of clauses 1-9, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, for a processing with a second processing module in the NN-based model, the plurality of regions do not overlap with each other.

[0141] Clause 11. The method of clause 10, wherein the second processing module comprises at least one of the following: a synthesis transform module, an analysis transform module, an MCM module, a sample prediction module, a latent sample reconstruction module, a hyper decoding module, or a hyper scale decoding module.

[0142] Clause 12. The method of any of clauses 10-11, wherein residual samples for each of the plurality of regions are coded independently from residual samples for the rest of the plurality of regions.

[0143] Clause 13. The method of any of clauses 1-12, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, for a processing with a synthesis transform module, at least one region among the plurality of regions is partitioned into a plurality of tiles, and the number of the plurality of tiles is an integer.

[0144] Clause 14. The method of clause 13, wherein a tile within a region is allowed to overlap with a further tile within the same region and is not allowed to overlap with a tile within a further region.

[0145] Clause 15. The method of any of clauses 1-14, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, the codestream comprises a plurality of substreams, and each of the plurality of substreams corresponds to one of the plurality of regions.

[0146] Clause 16. The method of clause 15, wherein a substream corresponding to one of the plurality of regions is allowed to be discarded.

[0147] Clause 17. The method of any of claim 15-16, wherein samples corresponding to one of the plurality of regions are reconstructed independently from substreams corresponding to the rest of the plurality of regions.

[0148] Clause 18. The method of any of clauses 15-17, wherein the codestream comprising a second indication of a region indicates that residual data for the region is present in the codestream.

[0149] Clause 19. The method of clause 18, wherein the second indication comprises an index.

5 [0150] Clause 20. The method of any of clauses 18-19, wherein the residual data comprises at least one of the following: residual samples for a primary component, or residual samples for a secondary component.

[0151] Clause 21. The method of any of clauses 10-20, wherein substreams of different regions among the plurality of regions are coded using an entropy coder in parallel.

10 [0152] Clause 22. The method of any of clauses 10-21, wherein the codestream further comprises at least one of the following: an indication indicating the number of threads for coding residual codestream for a primary component, or an indication indicating the number of threads for coding residual codestream for a secondary component.

15 [0153] Clause 23. The method of any of clauses 10-22, wherein a correspondence between an index of a substream and a region is determined based on a raster scanning order of the plurality of regions.

[0154] Clause 24. The method of any of clauses 3-23, wherein the codestream further comprises a third indication indicating a size of the substream.

20 [0155] Clause 25. The method of clause 24, wherein the size of the substream is represented by the number of bytes.

[0156] Clause 26. The method of any of clauses 24-25, wherein the third indication is coded using variable length coding.

25 [0157] Clause 27. The method of any of clauses 1-26, wherein the maximum number of the plurality of regions is dependent on at least one of the following: a profile to which the codestream conforms, a level to which the codestream conforms, or a size of the visual data.

[0158] Clause 28. The method of any of clauses 1-27, wherein a maximum size or a minimum size of one of the plurality of regions is dependent on at least one of the following: a profile to which the codestream conforms, a level to which the codestream conforms, a size of the visual data, or the number of the plurality of regions.

30 [0159] Clause 29. The method of any of clauses 1-28, wherein except for regions at a right boundary or a bottom boundary, the rest of the plurality of regions are of a same size.

[0160] Clause 30. The method of clause 29, wherein at least one of the right boundary or the bottom boundary is a boundary of the visual data.

[0161] Clause 31. The method of any of clauses 1-30, wherein the plurality of regions cover

the entire visual data.

[0162] Clause 32. The method of any of clauses 1-31, wherein a vertical size of one of the plurality of regions is equal to a product of k_{ver} and N_{ver} , k_{ver} is an integer and N_{ver} is a predetermined integer, or a horizontal size of the region is equal to a product of k_{hor} and N_{hor} , k_{hor} is an integer and N_{hor} is a predetermined integer.

[0163] Clause 33. The method of clause 32, wherein k_{ver} is determined based on the number of horizontal splits of the set of residual samples, k_{hor} is determined based on the number of vertical splits of the set of residual samples, or each of N_{ver} and N_{hor} is equal to 128.

[0164] Clause 34. The method of any of clauses 10-23, wherein data corresponding to one of the plurality of regions is allowed to be extracted from the codestream.

[0165] Clause 35. The method of clause 34, wherein at least one of the following is indicated in the codestream: whether to extract the data from the codestream, or how to extract the data from the codestream.

[0166] Clause 36. The method of any of clauses 34-35, wherein the data comprises at least one of the following: a substream, quality map information, residual data, luma residual data, chroma residual data, hyper latent data, or residual latent data.

[0167] Clause 37. The method of any of clauses 10-23, wherein the codestream further comprises at least one of the following: an indication indicating the number of regions for which a substream is present in the codestream, an indication indicating the number of regions for which a substream is absent from the codestream, an indication indicating whether substreams for all of the plurality of regions are included in the codestream, an indication indicating whether a substream of a region is discarded from the codestream, or an indication indicating whether a substream of a region is extracted from the codestream.

[0168] Clause 38. The method of any of clauses 18-20, wherein the codestream further comprises an indication indicating a length of the second indication.

[0169] Clause 39. The method of any of clauses 10-23, wherein data corresponding to different regions among the plurality of regions are allowed to be coded using an entropy coder in parallel.

[0170] Clause 40. The method of clause 39, wherein the data comprise at least one of the following: information for a primary component, information for a secondary component, residual information, hyper latent information, or quality map information.

[0171] Clause 41. The method of any of clauses 39-40, wherein the codestream further comprises a fourth indication indicating the number of parallel processing units.

[0172] Clause 42. The method of clause 41, wherein the number of parallel processing units

is allowed to be different for different data to be processed in parallel.

[0173] Clause 43. The method of any of clauses 39-42, wherein the number of parallel processing units or the maximum number of parallel processing units is determined based on at least one of the following: a profile to which the codestream conforms, a level to which the codestream conforms, a size of the visual data, or the number of the plurality of regions.

[0174] Clause 44. The method of any of clauses 1-28, wherein the plurality of regions are centrally aligned or shifted.

[0175] Clause 45. The method of clause 44, wherein the codestream further comprises an indication indicating whether region coordinates are shifted.

[0176] Clause 46. The method of any of clauses 1-45, wherein for processing a region, a tensor used as an input to a third processing module is centrally aligned with a tensor used as an input to a fourth processing module, or the tensor used as the input to the third processing module is shifted with respect to the tensor used as the input to the fourth processing module.

[0177] Clause 47. The method of clause 46, wherein the processing of the region with a synthesis transform is performed with centrally aligned tensors, and the processing of the region with a further processing module is performed with non-aligned tensors.

[0178] Clause 48. The method of clause 46, wherein the processing of the region with a synthesis transform is performed with centrally aligned tensors, and the processing of the region with a further processing module is performed with centrally aligned tensors or non-aligned tensors.

[0179] Clause 49. The method of clause 48, wherein the codestream further comprises an indication indicating whether the processing of the region with the further processing module is performed with the centrally aligned tensors or the non-aligned tensors.

[0180] Clause 50. The method of any of clauses 47-49, wherein the further processing module comprises at least one of the following: a hyper decoding module, a hyper scale decoding module, an entropy decoder a sample prediction module, an MCM module, or a latent sample reconstruction module.

[0181] Clause 51. The method of any of clauses 10-22, wherein the codestream further comprises an indication indicating a correspondence between an index of a substream and a region.

[0182] Clause 52. The method of any of clauses 1-51, wherein the number of the plurality of regions is no larger than a predetermined integer, or the number of rows of regions among the plurality of regions is no later than a predetermined integer, or the number of columns of

regions among the plurality of regions is no later than a predetermined integer.

[0183] Clause 53. The method of any of clauses 1-52, wherein an indication related to region-based coding is coded in a predictive way.

5 [0184] Clause 54. The method of any of clauses 1-53, wherein the visual data is at least a part of a picture of a video or an image.

[0185] Clause 55. The method of any of clauses 1-54, wherein the conversion includes encoding the visual data into the codestream.

[0186] Clause 56. The method of any of clauses 1-54, wherein the conversion includes decoding the visual data from the codestream.

10 [0187] Clause 57. An apparatus for visual data 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 1-56.

[0188] Clause 58. A non-transitory computer-readable storage medium storing instructions that cause a processor to perform a method in accordance with any of clauses 1-56.

15 [0189] Clause 59. A non-transitory computer-readable recording medium storing a codestream of visual data which is generated by a method performed by an apparatus for visual data processing, wherein the method comprises: performing a conversion between the visual data and the codestream with a neural network (NN)-based model, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of
20 residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

[0190] Clause 60. A method for storing a codestream of visual data, comprising: performing a conversion between the visual data and the codestream with a neural network (NN)-based
25 model; and storing the codestream in a non-transitory computer-readable recording medium, wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

30 **Example Device**

[0191] Fig. 29 illustrates a block diagram of a computing device 2900 in which various embodiments of the present disclosure can be implemented. The computing device 2900 may be implemented as or included in the source device 110 (or the visual data encoder 114) or the destination device 120 (or the visual data decoder 124).

[0192] It would be appreciated that the computing device 2900 shown in Fig. 29 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.

[0193] As shown in Fig. 29, the computing device 2900 includes a general-purpose computing device 2900. The computing device 2900 may at least comprise one or more processors or processing units 2910, a memory 2920, a storage unit 2930, one or more communication units 2940, one or more input devices 2950, and one or more output devices 2960.

[0194] In some embodiments, the computing device 2900 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 2900 can support any type of interface to a user (such as “wearable” circuitry and the like).

[0195] The processing unit 2910 may be a physical or virtual processor and can implement various processes based on programs stored in the memory 2920. 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 2900. The processing unit 2910 may also be referred to as a central processing unit (CPU), a microprocessor, a controller or a microcontroller.

[0196] The computing device 2900 typically includes various computer storage medium. Such medium can be any medium accessible by the computing device 2900, including, but not limited to, volatile and non-volatile medium, or detachable and non-detachable medium.

The memory 2920 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 2930 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 visual data and can be accessed in the computing device 2900.

[0197] The computing device 2900 may further include additional detachable/non-detachable, volatile/non-volatile memory medium. Although not shown in Fig. 29, 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 visual data medium interfaces.

[0198] The communication unit 2940 communicates with a further computing device via the communication medium. In addition, the functions of the components in the computing device 2900 can be implemented by a single computing cluster or multiple computing machines that can communicate via communication connections. Therefore, the computing device 2900 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.

[0199] The input device 2950 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 2960 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 2940, the computing device 2900 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 2900, or any devices (such as a network card, a modem and the like) enabling the computing device 2900 to communicate with one or more other computing devices, if required. Such communication can be performed via input/output (I/O) interfaces (not shown).

[0200] In some embodiments, instead of being integrated in a single device, some or all components of the computing device 2900 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, visual 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 visual 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 visual data center. Cloud computing infrastructures may provide the services through a shared visual 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.

[0201] The computing device 2900 may be used to implement visual data encoding/decoding in embodiments of the present disclosure. The memory 2920 may include one or more visual data coding modules 2925 having one or more program instructions. These modules are accessible and executable by the processing unit 2910 to perform the functionalities of the various embodiments described herein.

[0202] In the example embodiments of performing visual data encoding, the input device 2950 may receive visual data as an input 2970 to be encoded. The visual data may be processed, for example, by the visual data coding module 2925, to generate an encoded bitstream. The encoded bitstream may be provided via the output device 2960 as an output 2980.

[0203] In the example embodiments of performing visual data decoding, the input device 2950 may receive an encoded bitstream as the input 2970. The encoded bitstream may be processed, for example, by the visual data coding module 2925, to generate decoded visual data. The decoded visual data may be provided via the output device 2960 as the output 2980.

[0204] 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 visual data processing, comprising:
performing a conversion between visual data and a codestream of the visual data with a
5 neural network (NN)-based model,

wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

2. The method of claim 1, wherein the set of residual samples is a residual tensor, and each of the plurality of subsets of residual samples is a residual sub-tensor.

3. The method of any of claims 1-2, wherein if the first indication indicates that there is
15 only one substream for primary residual samples for the plurality of regions and only one substream for secondary residual samples for the plurality of regions, for a processing with a first processing module in the NN-based model, the plurality of regions are allowed to overlap.

4. The method of claim 3, wherein the first processing module comprises at least one of
20 the following:

a hyper decoding module,
a multistage context modelling (MCM) module,
a synthesis transform module,
a latent sample reconstruction module,
25 a sample prediction module,
an analysis transform module, or
a hyper scale decoding module.

5. The method of any of claims 3-4, wherein the plurality of regions comprise a first region,
30 and at least one padding amount for the first region is determined based on an overlap amount between the first region and one or more regions neighboring to the first region.

6. The method of claim 5, wherein the at least one padding amount comprises at least one of the following:

a padding amount in a left direction,
a padding amount in a right direction,
a padding amount in a bottom direction, or
a padding amount in a top direction.

5

7. The method of any of claims 5-6, wherein the number of samples to be discarded from an output of the first processing module is determined based on the at least one padding amount.

10

8. The method of any of claims 3-7, wherein an overlap amount between two of the plurality of regions is dependent on the first processing module.

9. The method of claim 8, wherein the codestream further comprises at least one of the following:

15

an indication indicating the overlap amount or a padding amount used in a hyper decoding module, or

an indication indicating the overlap amount or a padding amount used in an MCM module.

20

10. The method of any of claims 1-9, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, for a processing with a second processing module in the NN-based model, the plurality of regions do not overlap with each other.

11. The method of claim 10, wherein the second processing module comprises at least one of the following:

25

a synthesis transform module,

an analysis transform module,

an MCM module,

a sample prediction module,

a latent sample reconstruction module,

30

a hyper decoding module, or

a hyper scale decoding module.

12. The method of any of claims 10-11, wherein residual samples for each of the plurality of regions are coded independently from residual samples for the rest of the plurality of regions.

13. The method of any of claims 1-12, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, for a processing with a synthesis transform module, at least one region among the plurality of regions is partitioned into a plurality of tiles, and the number of the plurality of tiles is an integer.

14. The method of claim 13, wherein a tile within a region is allowed to overlap with a further tile within the same region and is not allowed to overlap with a tile within a further region.

15. The method of any of claims 1-14, wherein if the first indication indicates that residual samples for each of the plurality of regions is in a substream of the codestream, the codestream comprises a plurality of substreams, and each of the plurality of substreams corresponds to one of the plurality of regions.

16. The method of claim 15, wherein a substream corresponding to one of the plurality of regions is allowed to be discarded.

17. The method of any of claim 15-16, wherein samples corresponding to one of the plurality of regions are reconstructed independently from substreams corresponding to the rest of the plurality of regions.

18. The method of any of claims 15-17, wherein the codestream comprising a second indication of a region indicates that residual data for the region is present in the codestream.

19. The method of claim 18, wherein the second indication comprises an index.

20. The method of any of claims 18-19, wherein the residual data comprises at least one of the following:

residual samples for a primary component, or
residual samples for a secondary component.

21. The method of any of claims 10-20, wherein substreams of different regions among the plurality of regions are coded using an entropy coder in parallel.

22. The method of any of claims 10-21, wherein the codestream further comprises at least one of the following:

an indication indicating the number of threads for coding residual codestream for a primary component, or

5 an indication indicating the number of threads for coding residual codestream for a secondary component.

23. The method of any of claims 10-22, wherein a correspondence between an index of a substream and a region is determined based on a raster scanning order of the plurality of regions.

10

24. The method of any of claims 3-23, wherein the codestream further comprises a third indication indicating a size of the substream.

25. The method of claim 24, wherein the size of the substream is represented by the number of bytes.

15

26. The method of any of claims 24-25, wherein the third indication is coded using variable length coding.

27. The method of any of claims 1-26, wherein the maximum number of the plurality of regions is dependent on at least one of the following:

20

a profile to which the codestream conforms,
a level to which the codestream conforms, or
a size of the visual data.

25

28. The method of any of claims 1-27, wherein a maximum size or a minimum size of one of the plurality of regions is dependent on at least one of the following:

a profile to which the codestream conforms,
a level to which the codestream conforms,
a size of the visual data, or
the number of the plurality of regions.

30

29. The method of any of claims 1-28, wherein except for regions at a right boundary or a bottom boundary, the rest of the plurality of regions are of a same size.

30. The method of claim 29, wherein at least one of the right boundary or the bottom boundary is a boundary of the visual data.

5 31. The method of any of claims 1-30, wherein the plurality of regions cover the entire visual data.

32. The method of any of claims 1-31, wherein a vertical size of one of the plurality of regions is equal to a product of k_{ver} and N_{ver} , k_{ver} is an integer and N_{ver} is a predetermined integer, or

10 a horizontal size of the region is equal to a product of k_{hor} and N_{hor} , k_{hor} is an integer and N_{hor} is a predetermined integer.

33. The method of claim 32, wherein k_{ver} is determined based on the number of horizontal splits of the set of residual samples, k_{hor} is determined based on the number of vertical splits of the set of residual samples, or each of N_{ver} and N_{hor} is equal to 128.

34. The method of any of claims 10-23, wherein data corresponding to one of the plurality of regions is allowed to be extracted from the codestream.

20 35. The method of claim 34, wherein at least one of the following is indicated in the codestream:

whether to extract the data from the codestream, or
how to extract the data from the codestream.

25 36. The method of any of claims 34-35, wherein the data comprises at least one of the following:

a substream,
quality map information,
30 residual data,
luma residual data,
chroma residual data,
hyper latent data, or
residual latent data.

37. The method of any of claims 10-23, wherein the codestream further comprises at least one of the following:

an indication indicating the number of regions for which a substream is present in the
5 codestream,

an indication indicating the number of regions for which a substream is absent from the codestream,

an indication indicating whether substreams for all of the plurality of regions are included in the codestream,

10 an indication indicating whether a substream of a region is discarded from the codestream,

or

an indication indicating whether a substream of a region is extracted from the codestream.

38. The method of any of claims 18-20, wherein the codestream further comprises an
15 indication indicating a length of the second indication.

39. The method of any of claims 10-23, wherein data corresponding to different regions among the plurality of regions are allowed to be coded using an entropy coder in parallel.

20 40. The method of claim 39, wherein the data comprise at least one of the following:

information for a primary component,

information for a secondary component,

residual information,

hyper latent information, or

25 quality map information.

41. The method of any of claims 39-40, wherein the codestream further comprises a fourth indication indicating the number of parallel processing units.

30 42. The method of claim 41, wherein the number of parallel processing units is allowed to be different for different data to be processed in parallel.

43. The method of any of claims 39-42, wherein the number of parallel processing units or the maximum number of parallel processing units is determined based on at least one of the

following:

- a profile to which the codestream conforms,
- a level to which the codestream conforms,
- a size of the visual data, or
- 5 the number of the plurality of regions.

44. The method of any of claims 1-28, wherein the plurality of regions are centrally aligned or shifted.

10 45. The method of claim 44, wherein the codestream further comprises an indication indicating whether region coordinates are shifted.

15 46. The method of any of claims 1-45, wherein for processing a region, a tensor used as an input to a third processing module is centrally aligned with a tensor used as an input to a fourth processing module, or the tensor used as the input to the third processing module is shifted with respect to the tensor used as the input to the fourth processing module.

20 47. The method of claim 46, wherein the processing of the region with a synthesis transform is performed with centrally aligned tensors, and the processing of the region with a further processing module is performed with non-aligned tensors.

25 48. The method of claim 46, wherein the processing of the region with a synthesis transform is performed with centrally aligned tensors, and the processing of the region with a further processing module is performed with centrally aligned tensors or non-aligned tensors.

49. The method of claim 48, wherein the codestream further comprises an indication indicating whether the processing of the region with the further processing module is performed with the centrally aligned tensors or the non-aligned tensors.

30 50. The method of any of claims 47-49, wherein the further processing module comprises at least one of the following:

- a hyper decoding module,
- a hyper scale decoding module,
- an entropy decoder

a sample prediction module,
an MCM module, or
a latent sample reconstruction module.

5 51. The method of any of claims 10-22, wherein the codestream further comprises an indication indicating a correspondence between an index of a substream and a region.

 52. The method of any of claims 1-51, wherein the number of the plurality of regions is no
larger than a predetermined integer, or
10 the number of rows of regions among the plurality of regions is no later than a
predetermined integer, or
 the number of columns of regions among the plurality of regions is no later than a
predetermined integer.

15 53. The method of any of claims 1-52, wherein an indication related to region-based coding is coded in a predictive way.

 54. The method of any of claims 1-53, wherein the visual data is at least a part of a picture
of a video or an image.

20 55. The method of any of claims 1-54, wherein the conversion includes encoding the visual data into the codestream.

 56. The method of any of claims 1-54, wherein the conversion includes decoding the visual
25 data from the codestream.

 57. An apparatus for visual data processing comprising a processor and a non-transitory
memory with instructions thereon, wherein the instructions upon execution by the processor, cause
the processor to perform a method in accordance with any of claims 1-56.

30 58. A non-transitory computer-readable storage medium storing instructions that cause a processor to perform a method in accordance with any of claims 1-56.

 59. A non-transitory computer-readable recording medium storing a codestream of visual

data which is generated by a method performed by an apparatus for visual data processing, wherein the method comprises:

performing a conversion between the visual data and the codestream with a neural network (NN)-based model,

5 wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

10 60. A method for storing a codestream of visual data, comprising:

performing a conversion between the visual data and the codestream with a neural network (NN)-based model; and

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

15 wherein a set of residual samples associated with the visual data is partitioned into a plurality of subsets of residual samples that correspond to a plurality of regions, and the codestream comprises a first indication indicating whether residual samples for each of the plurality of regions are in a substream of the codestream that corresponds to the region.

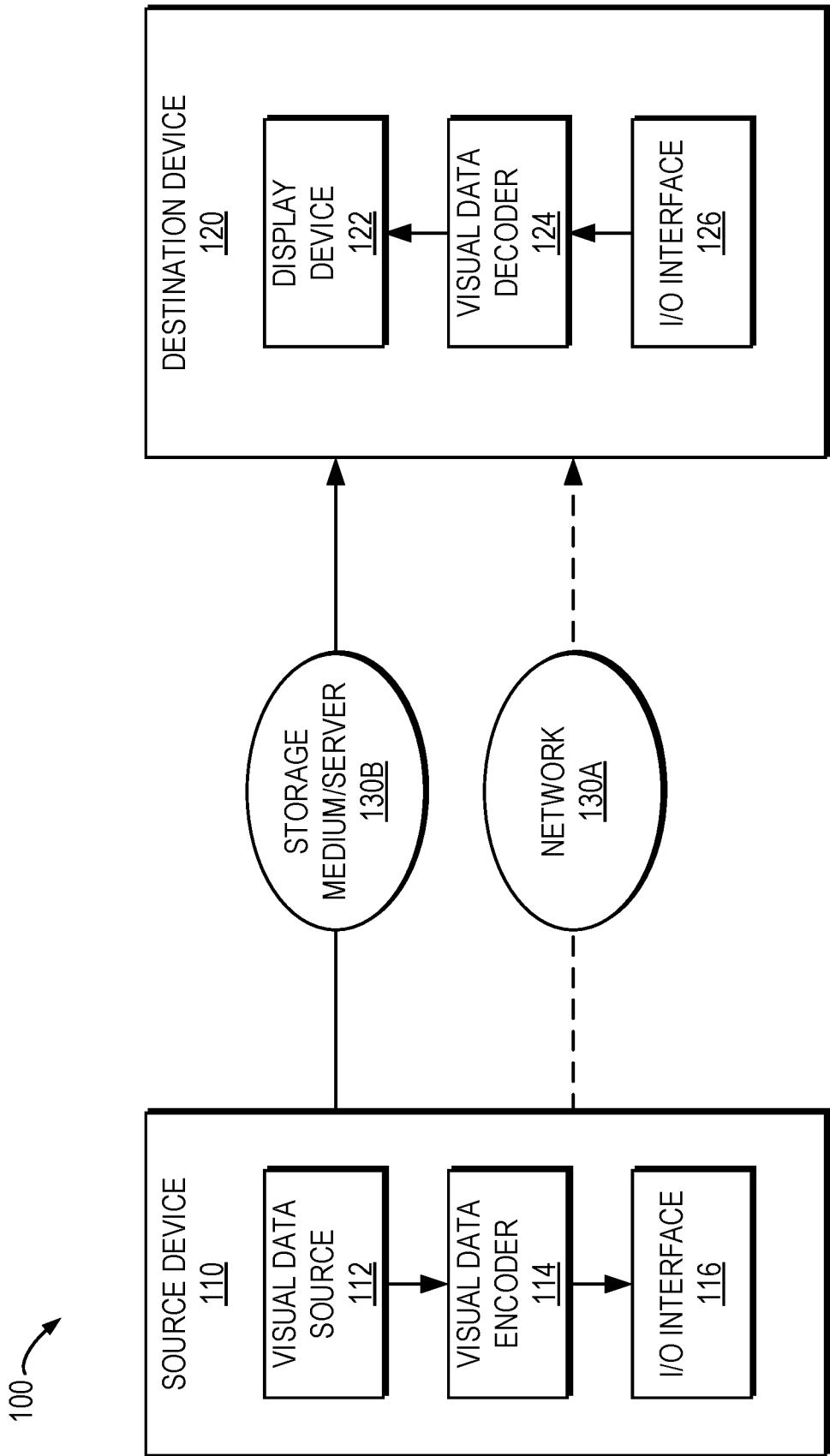


Fig. 1A

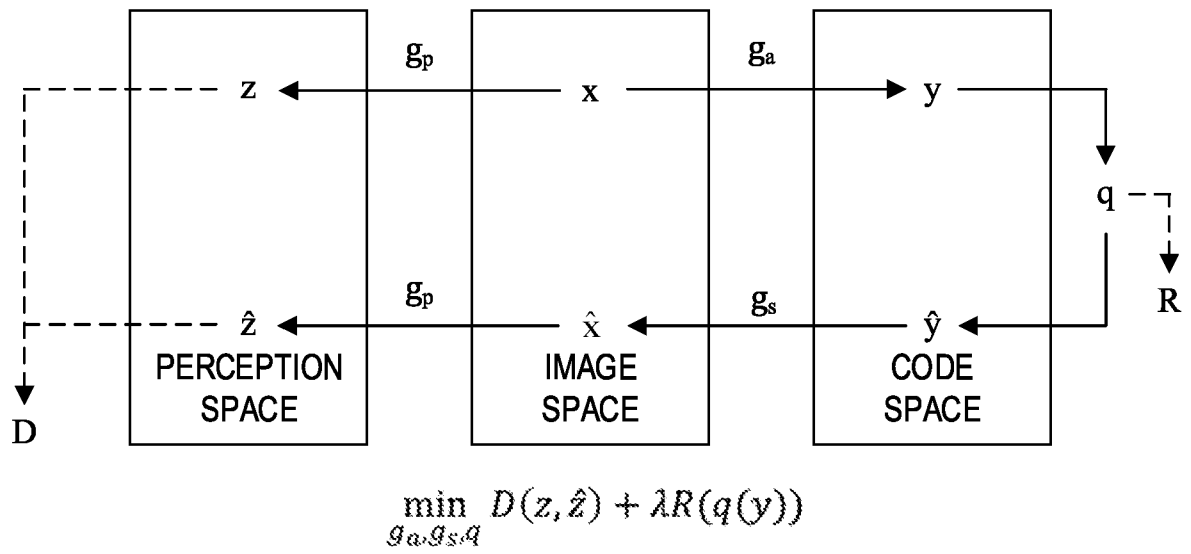


Fig. 1B

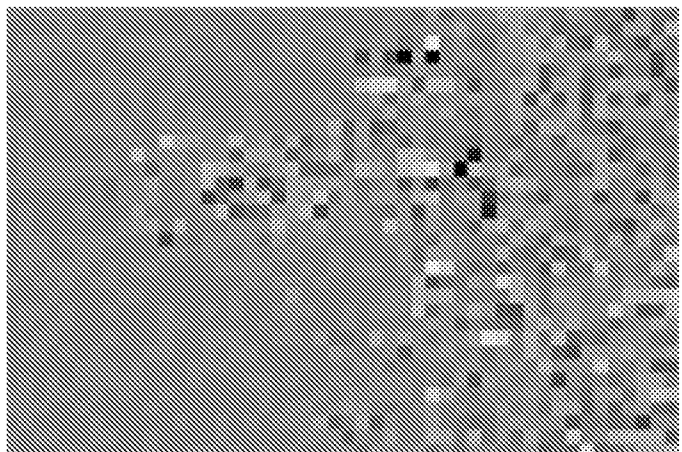
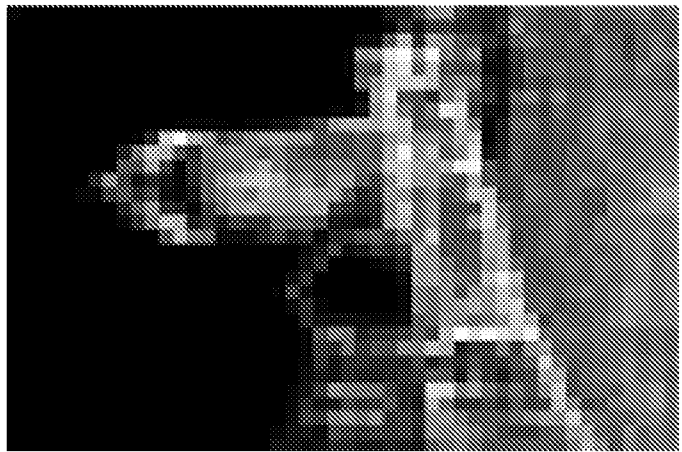
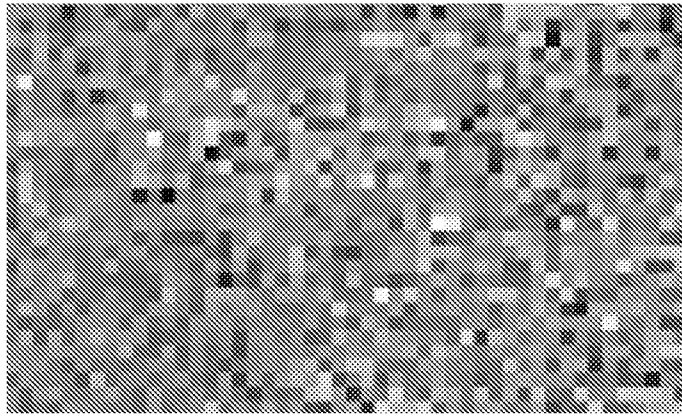


Fig. 2

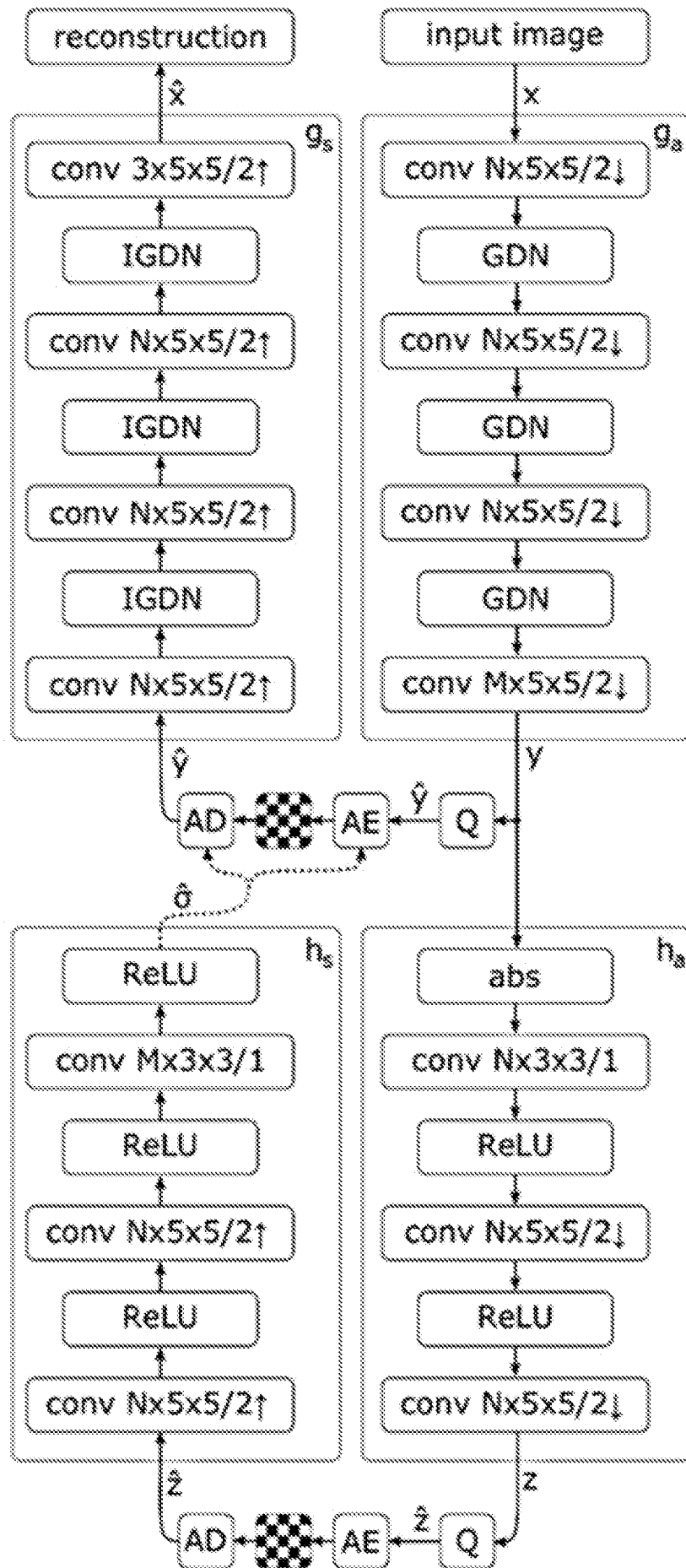


Fig. 3

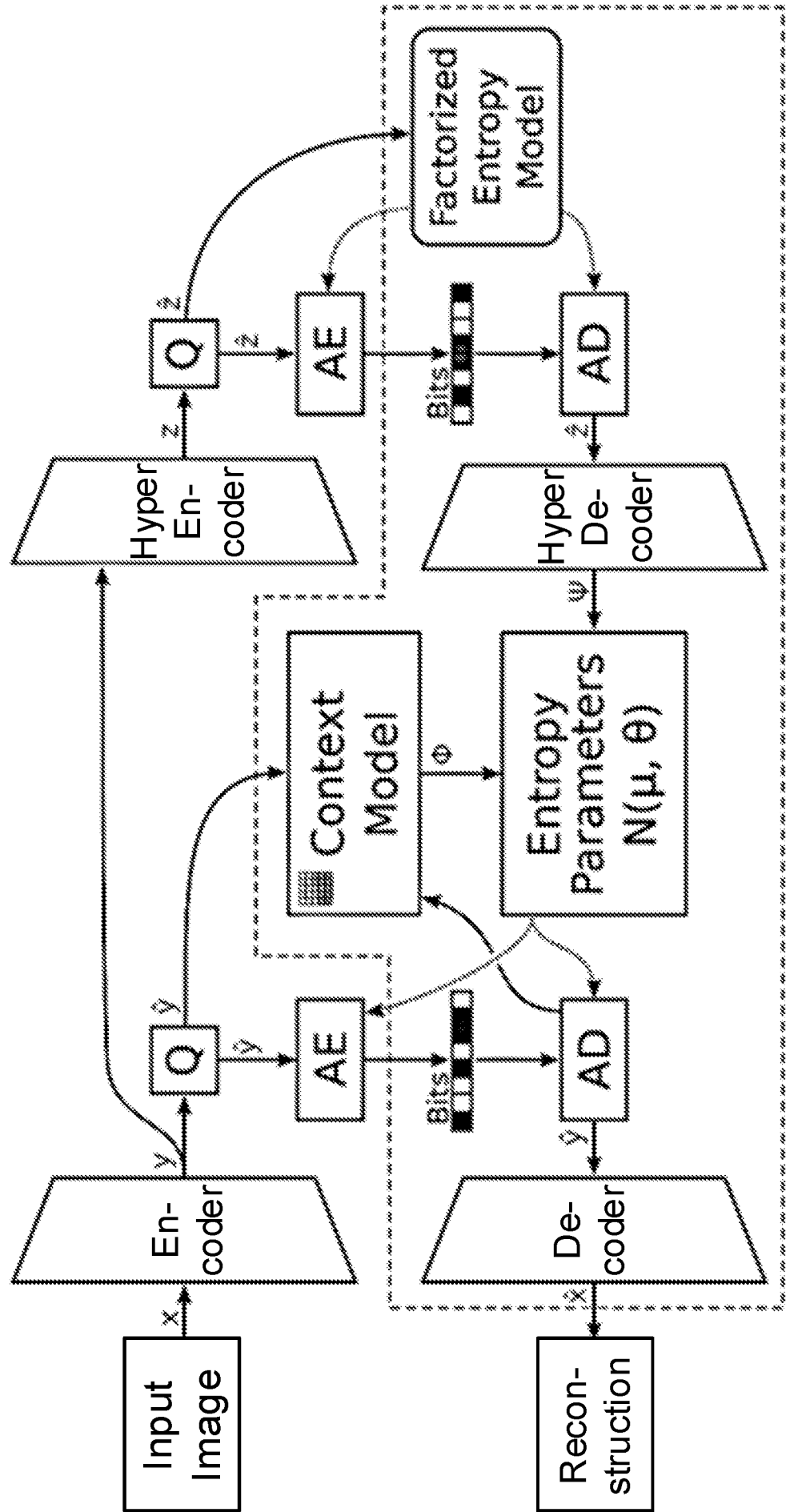


Fig. 4

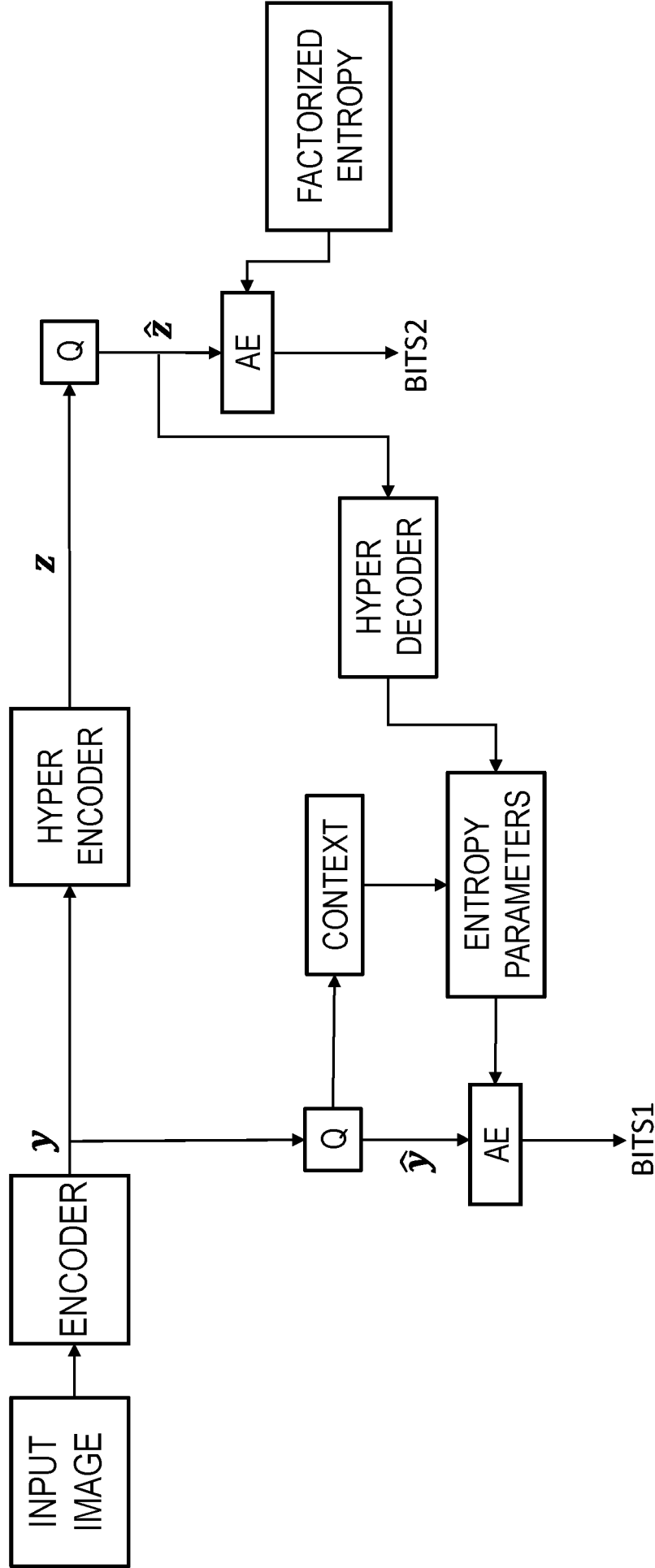


Fig. 5

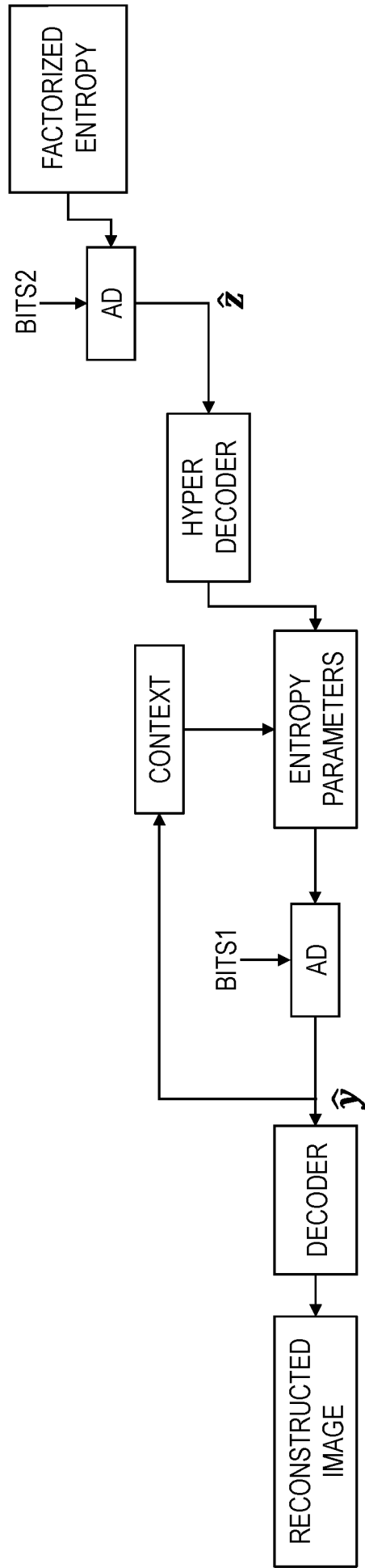


Fig. 6

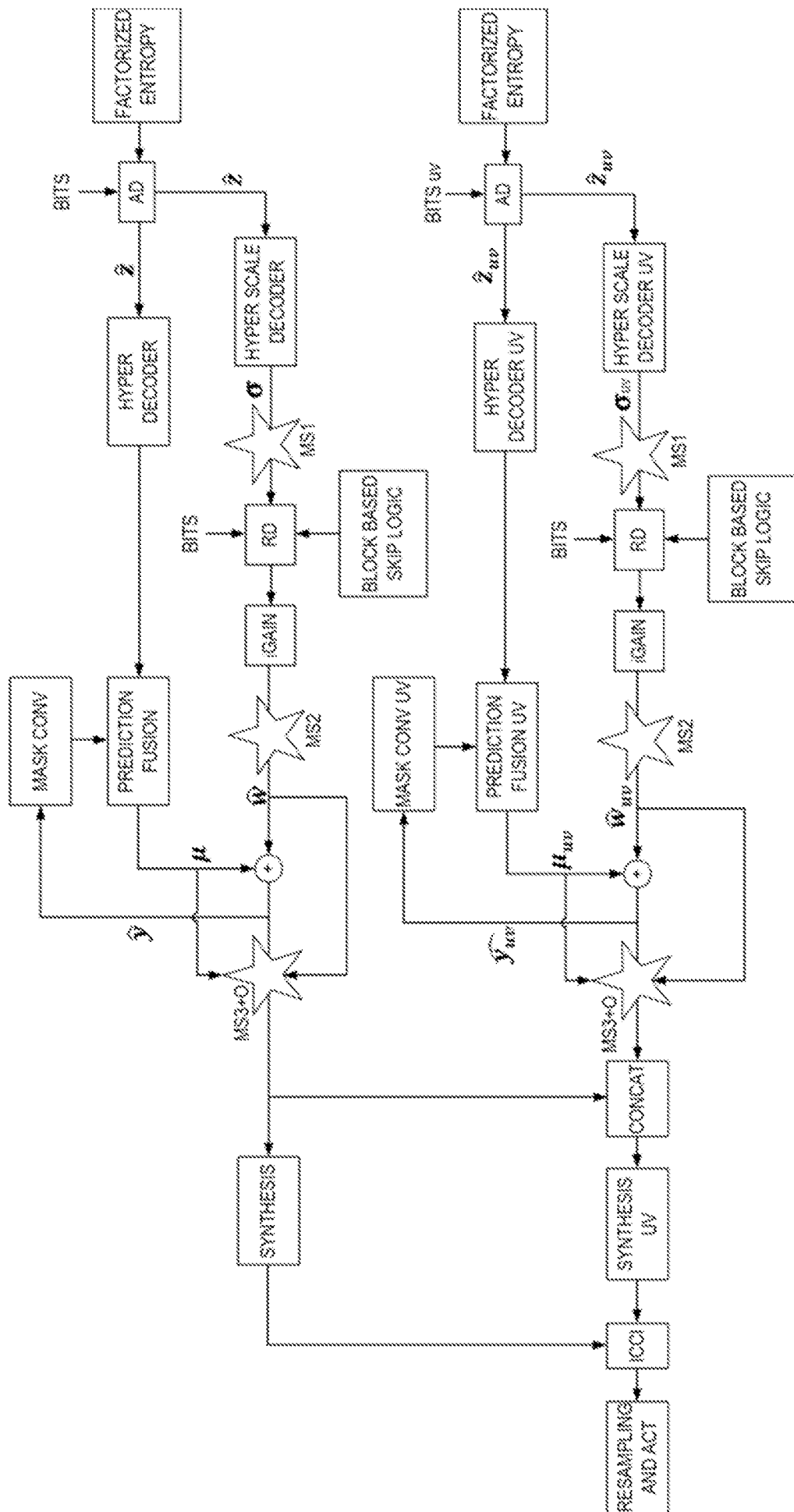


Fig. 7

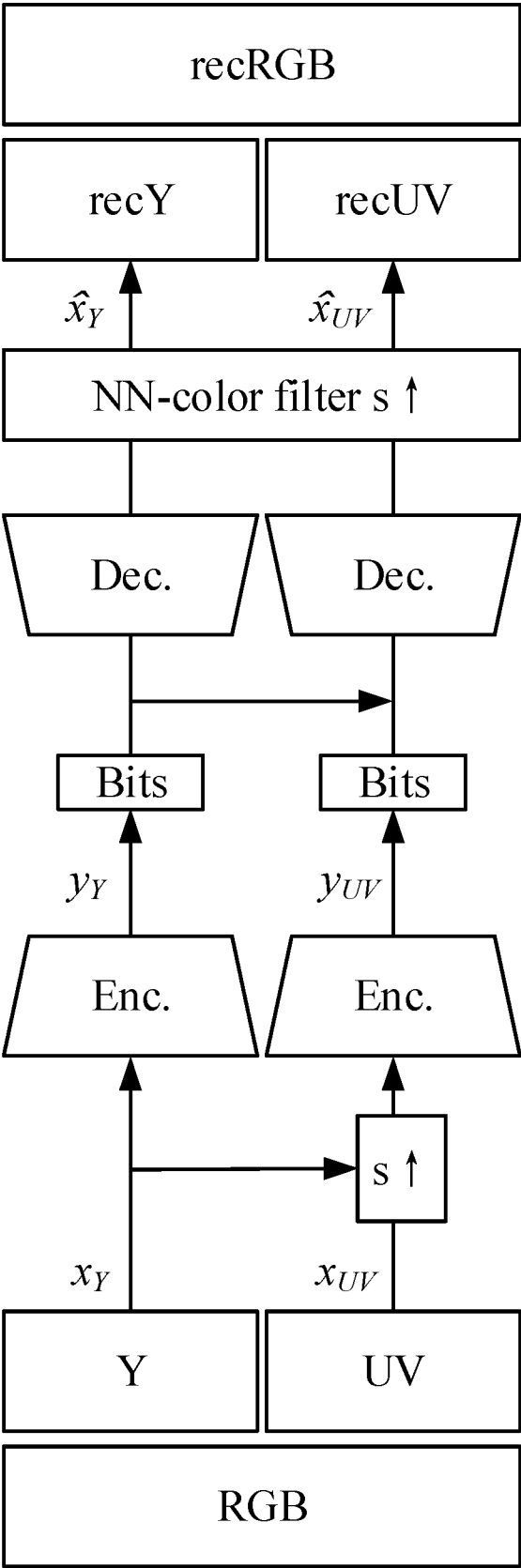


Fig. 8

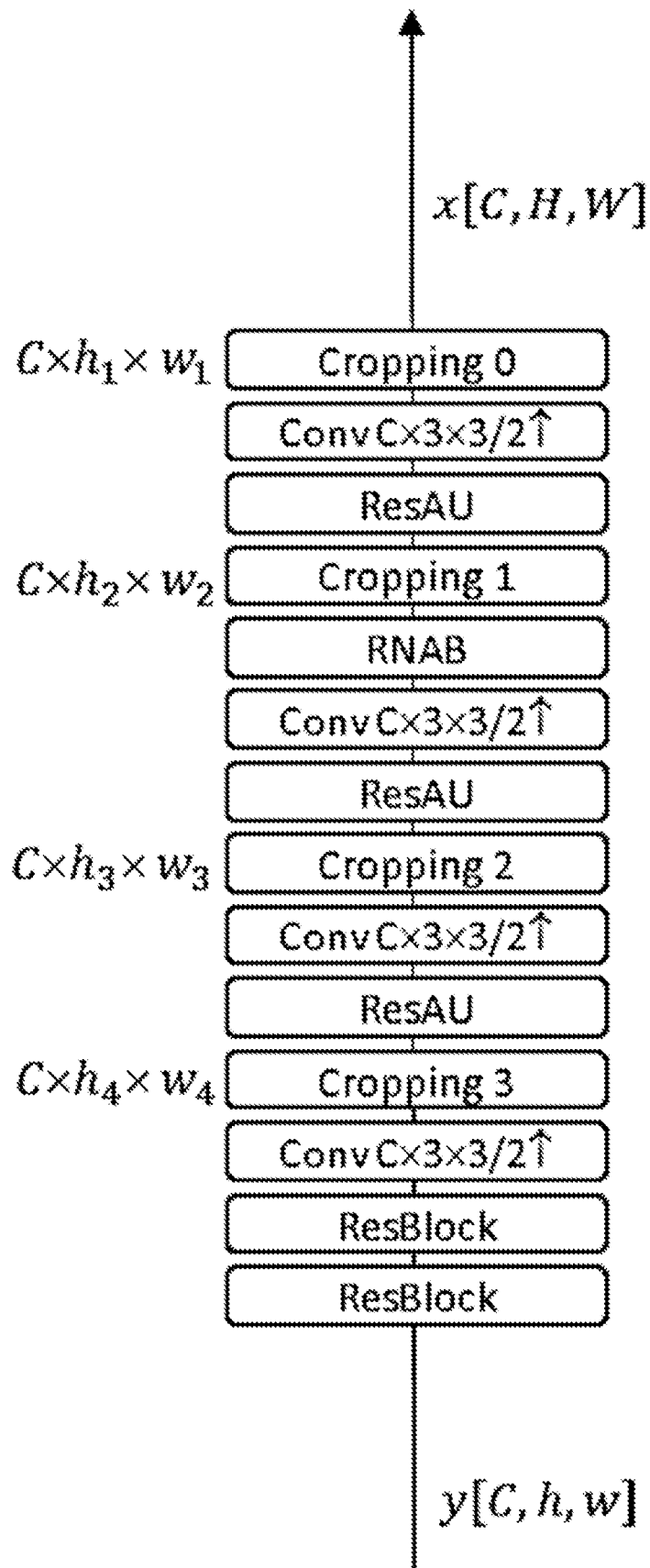


Fig. 9

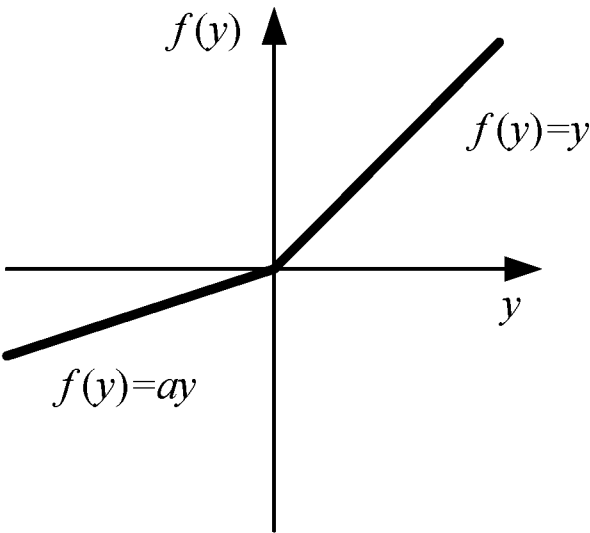


Fig. 10

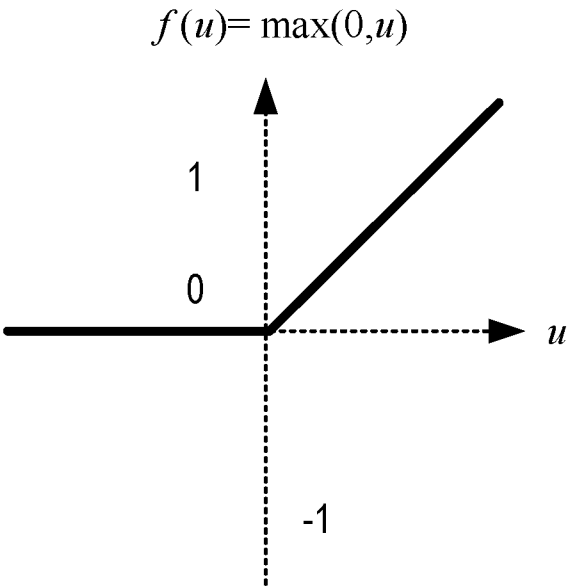


Fig. 11

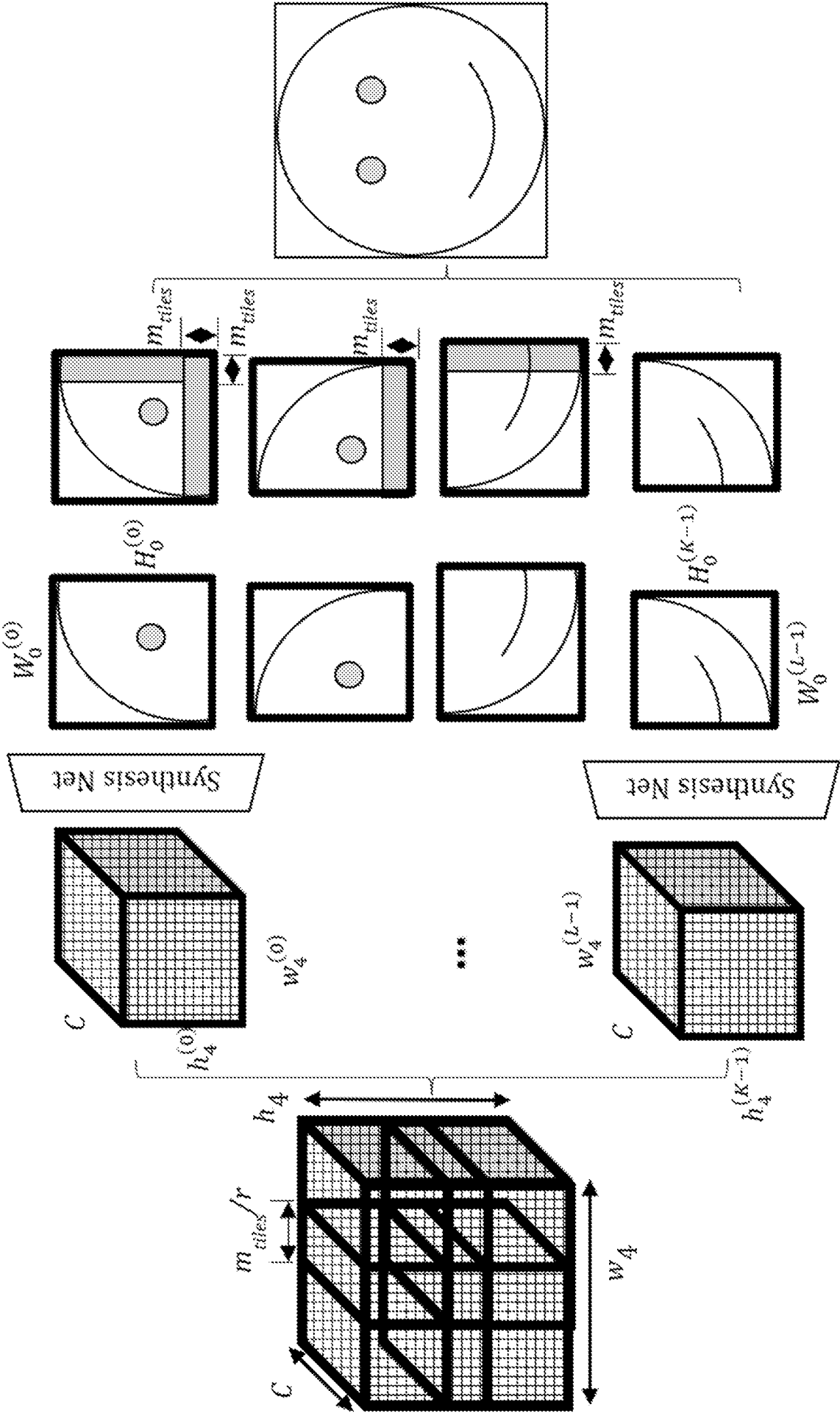


Fig. 12

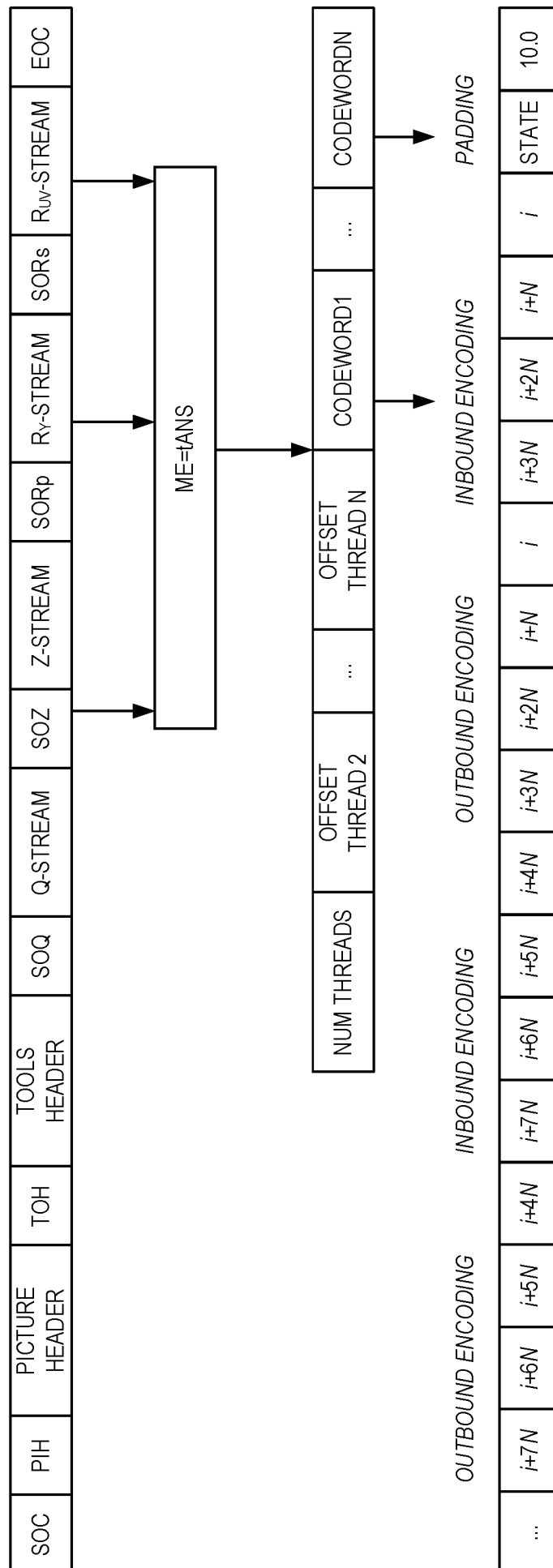


Fig. 13

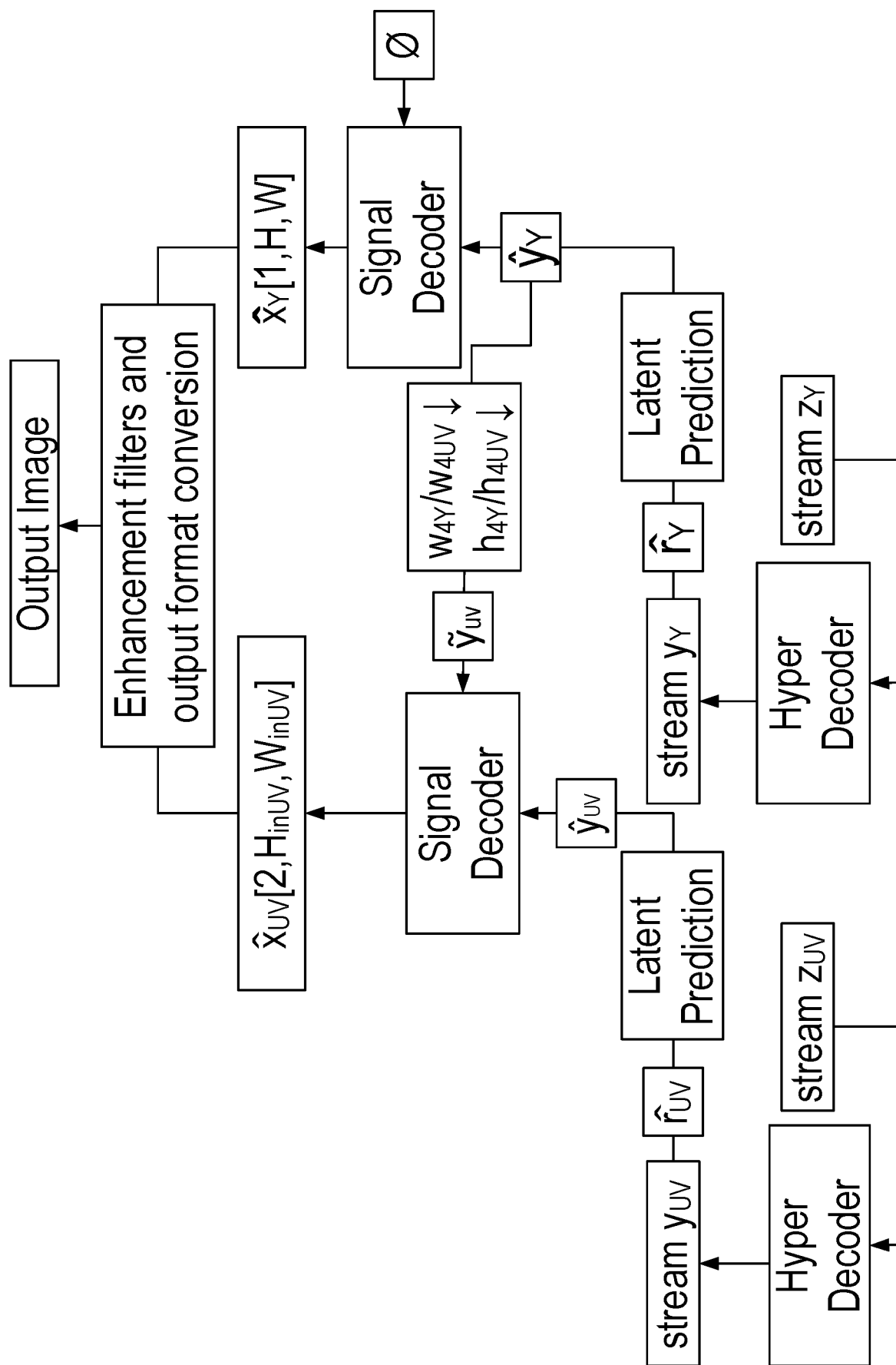


Fig. 14

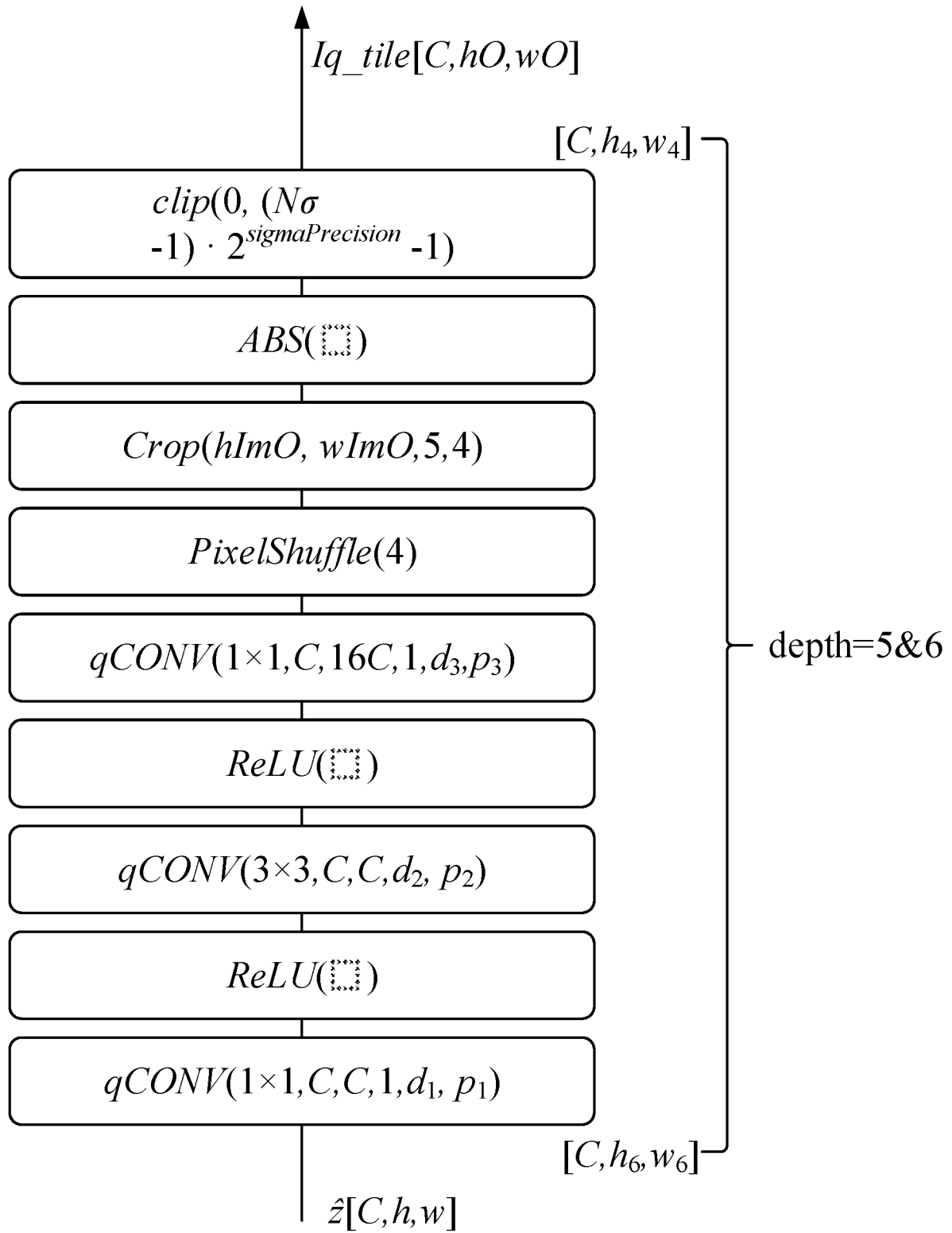


Fig. 15

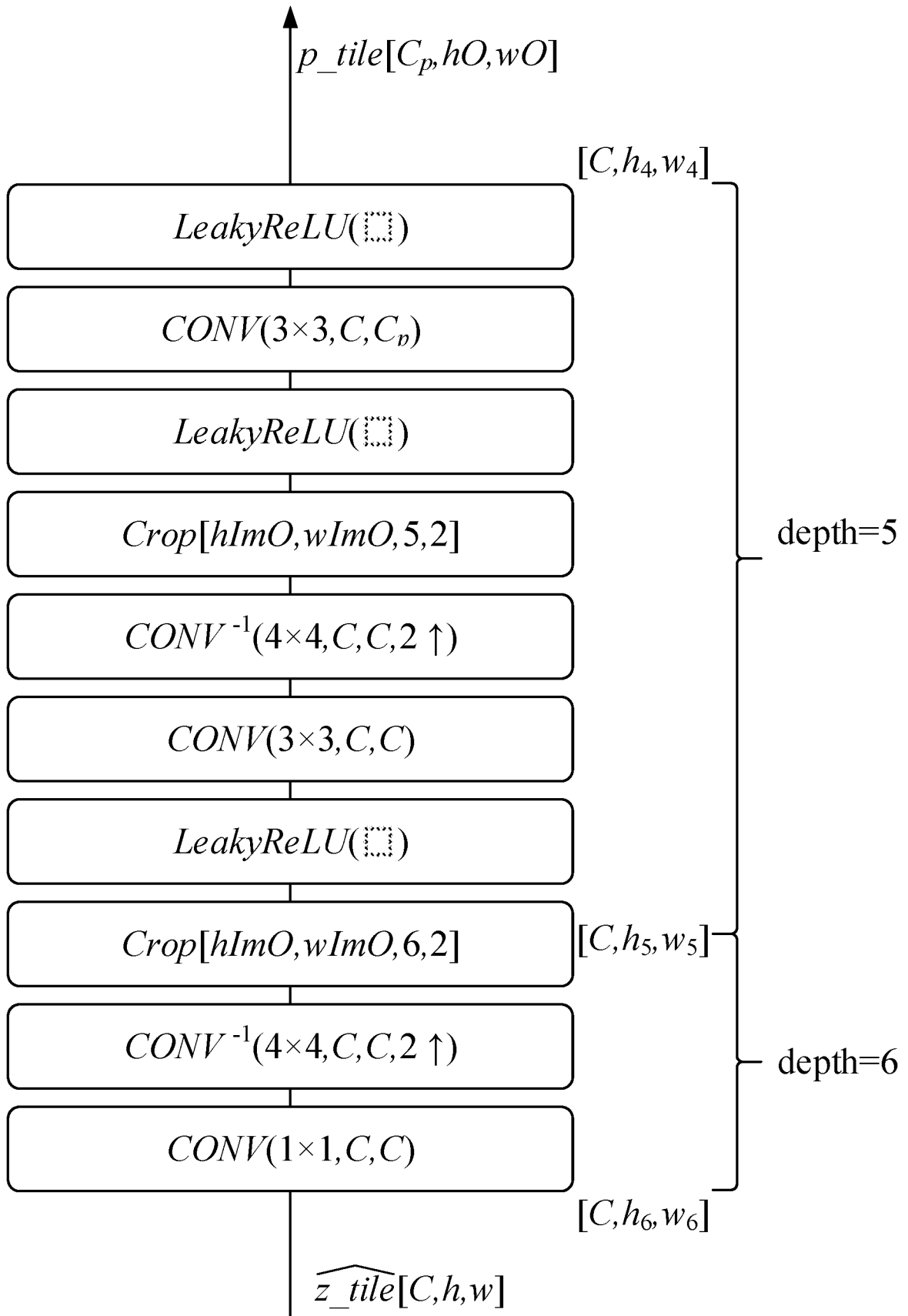


Fig. 16

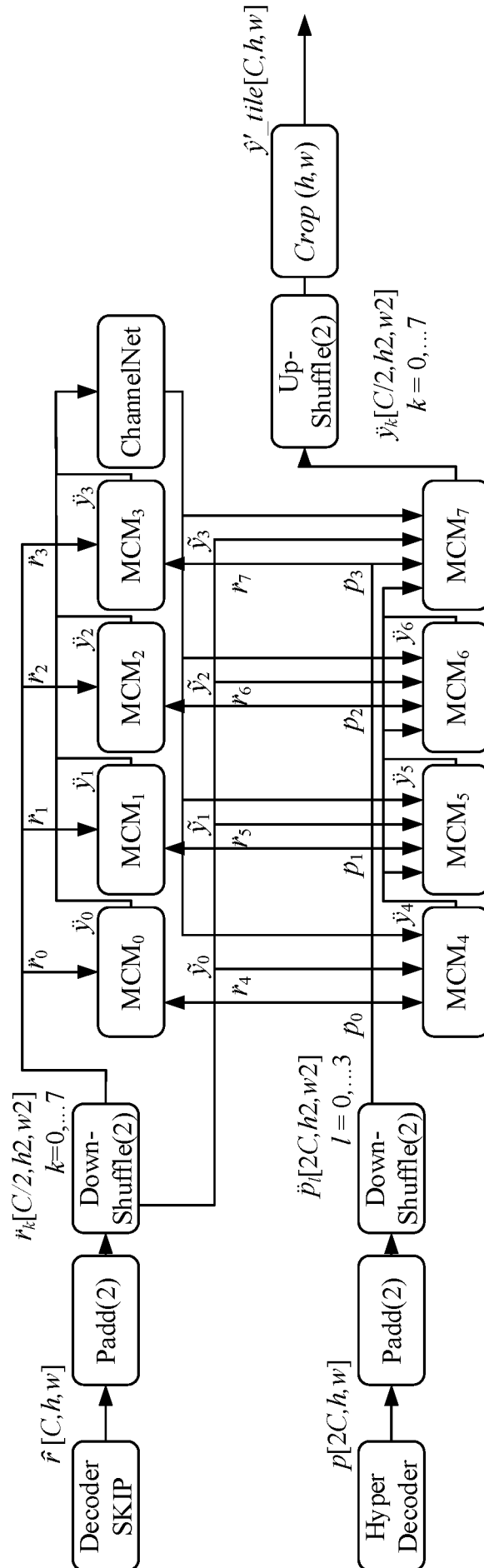


Fig. 17

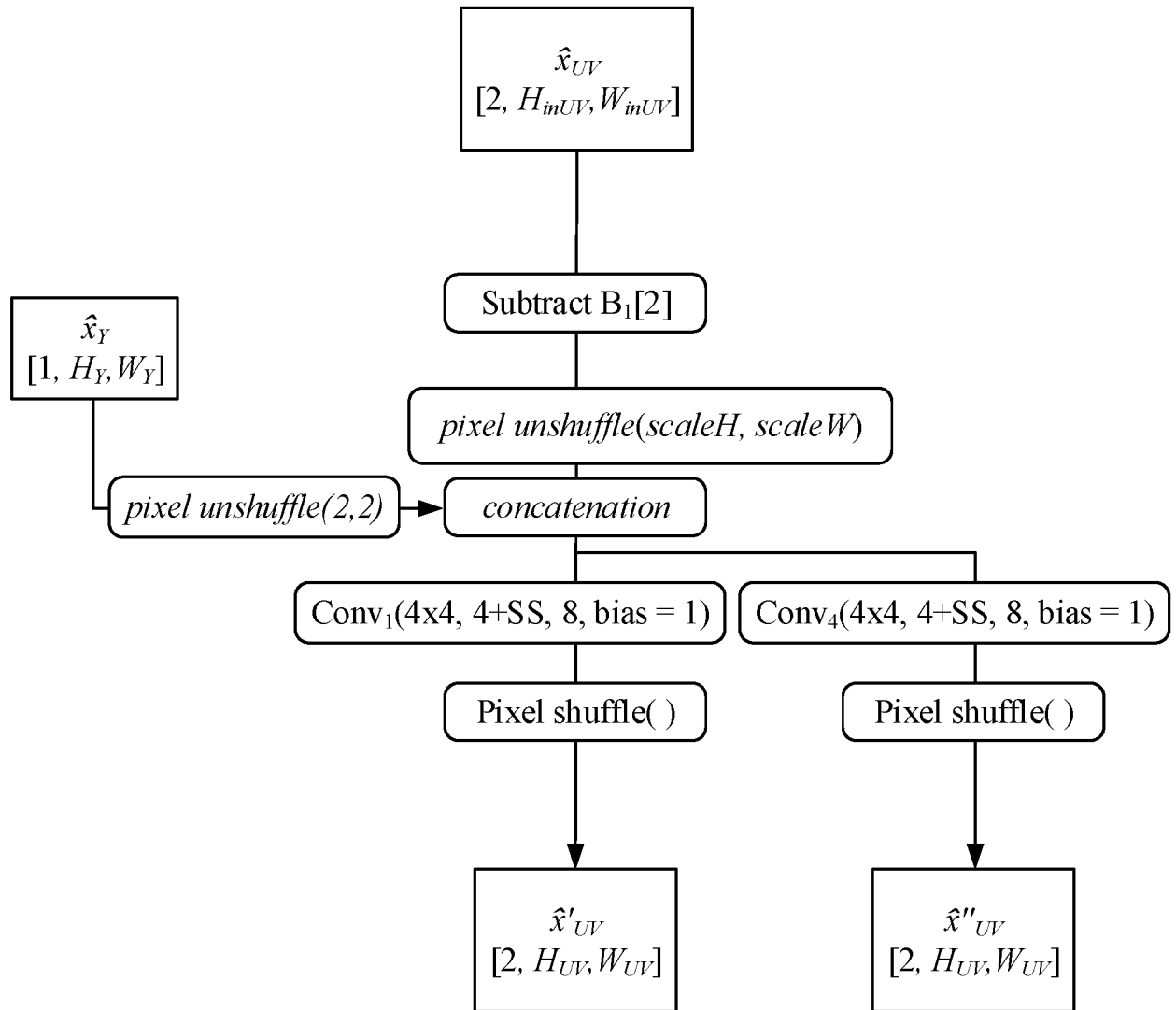


Fig. 18

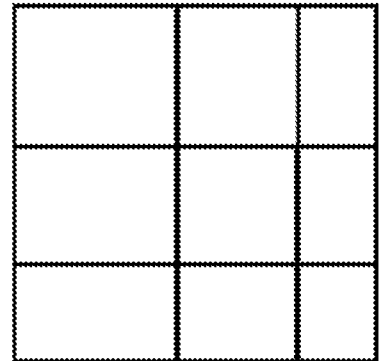
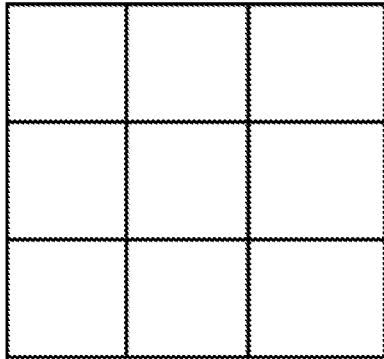


Fig. 19

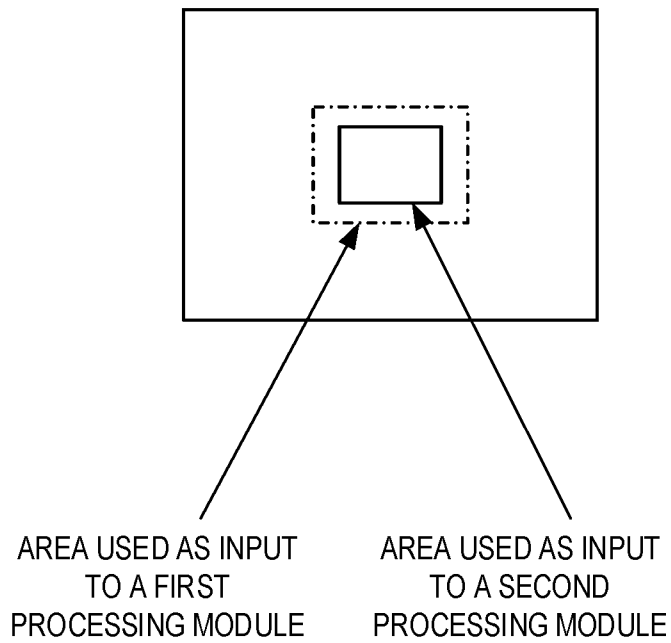
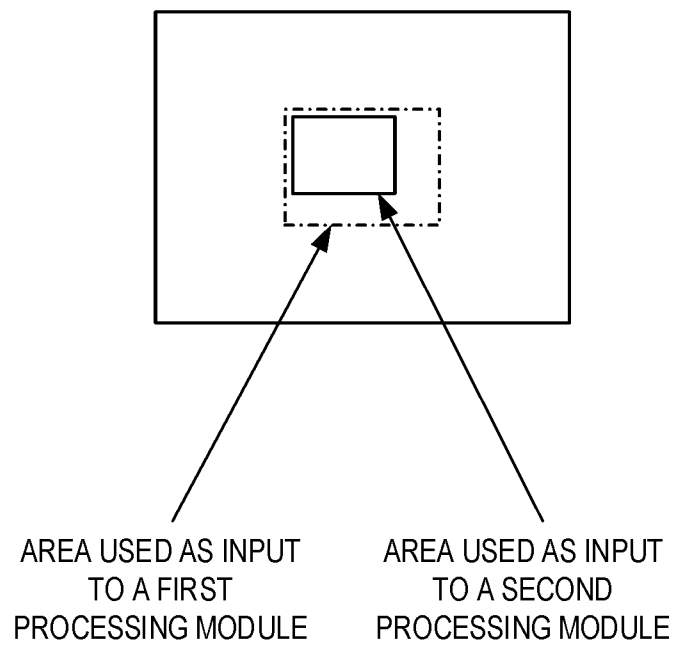
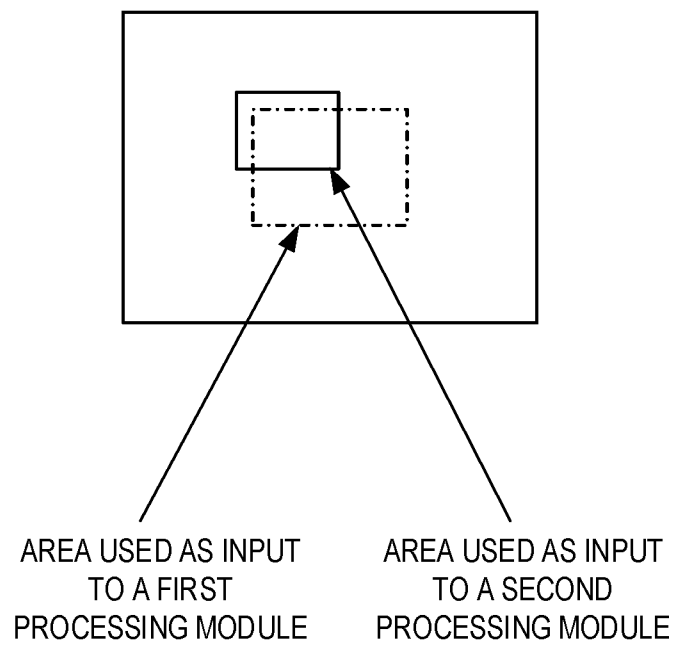


Fig. 20A

**Fig. 20B****Fig. 20C**

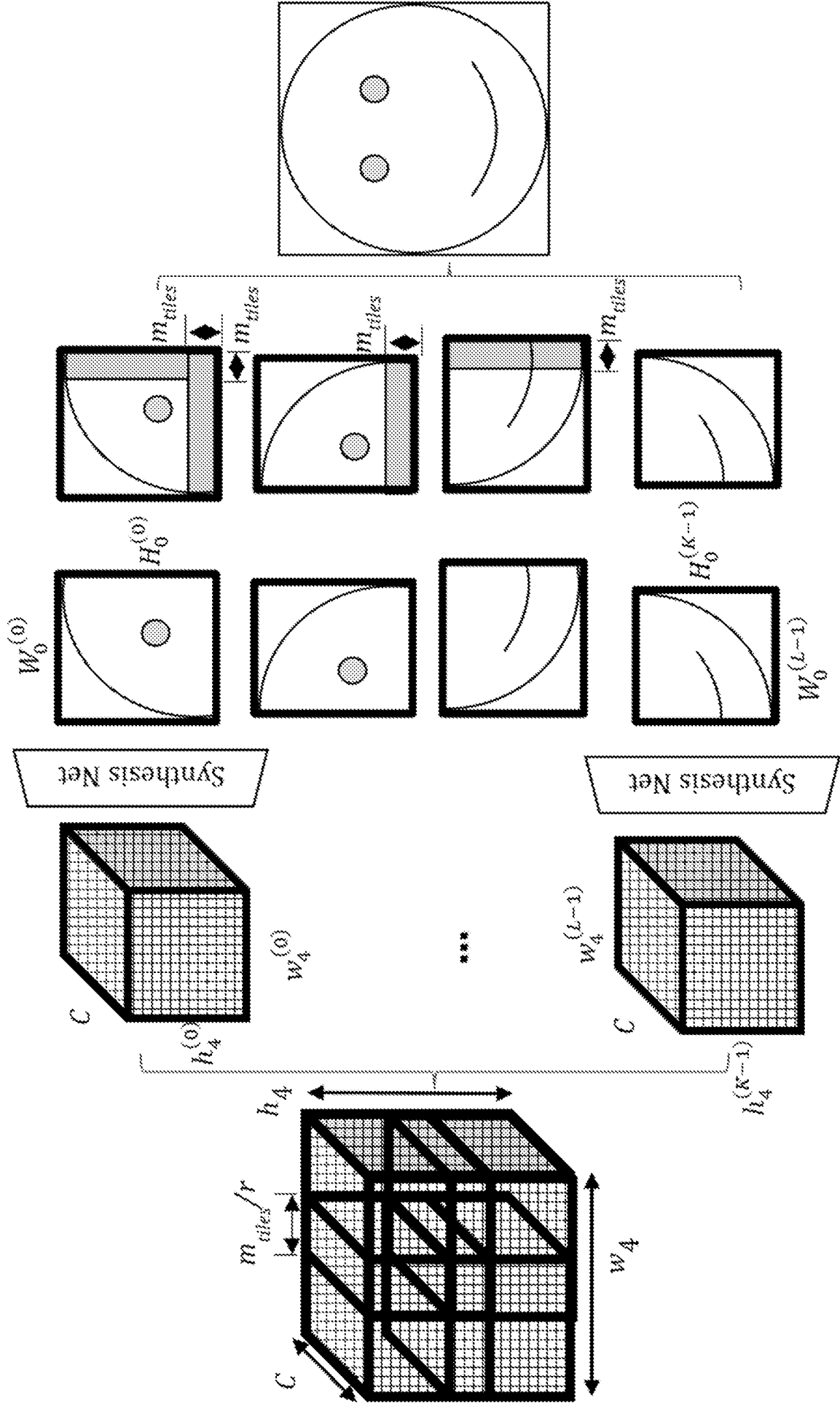
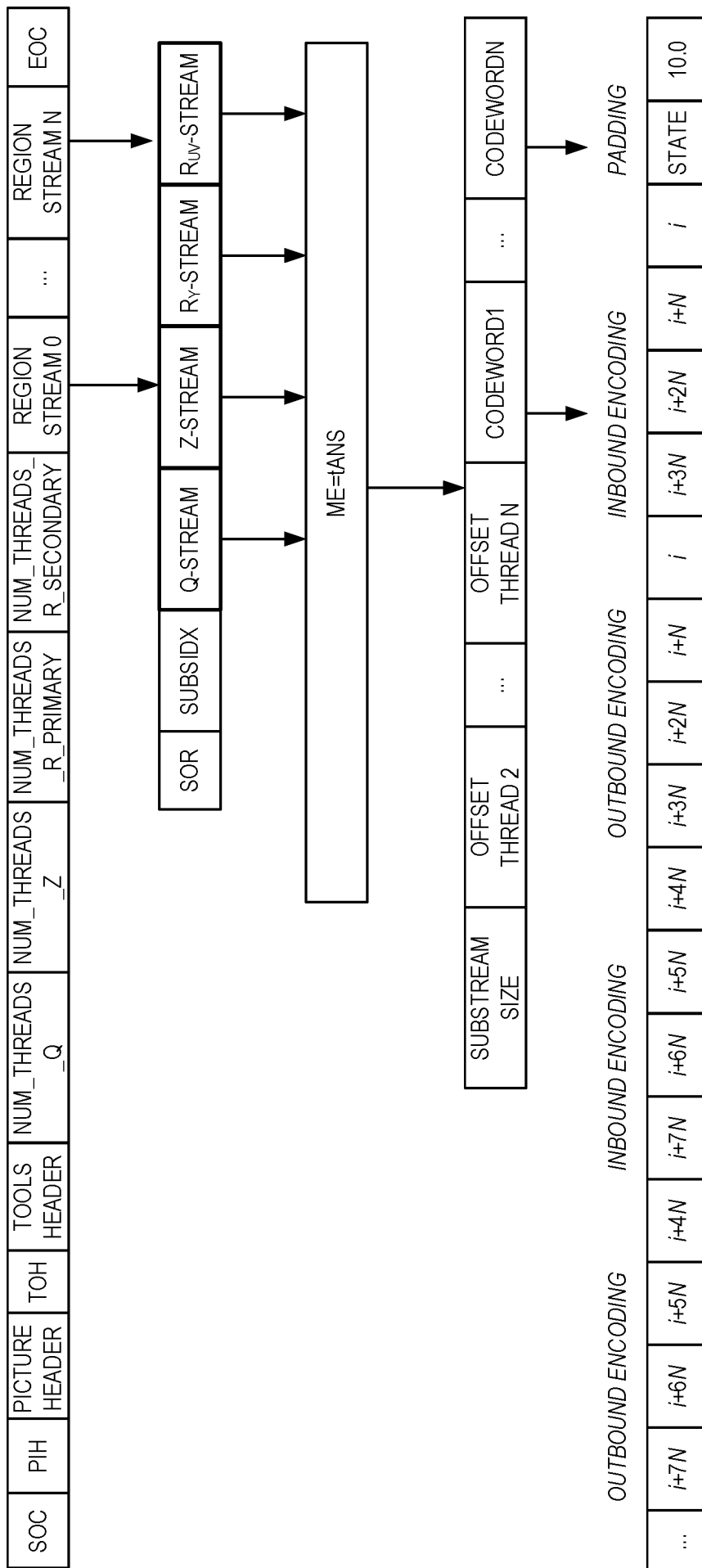


Fig. 21



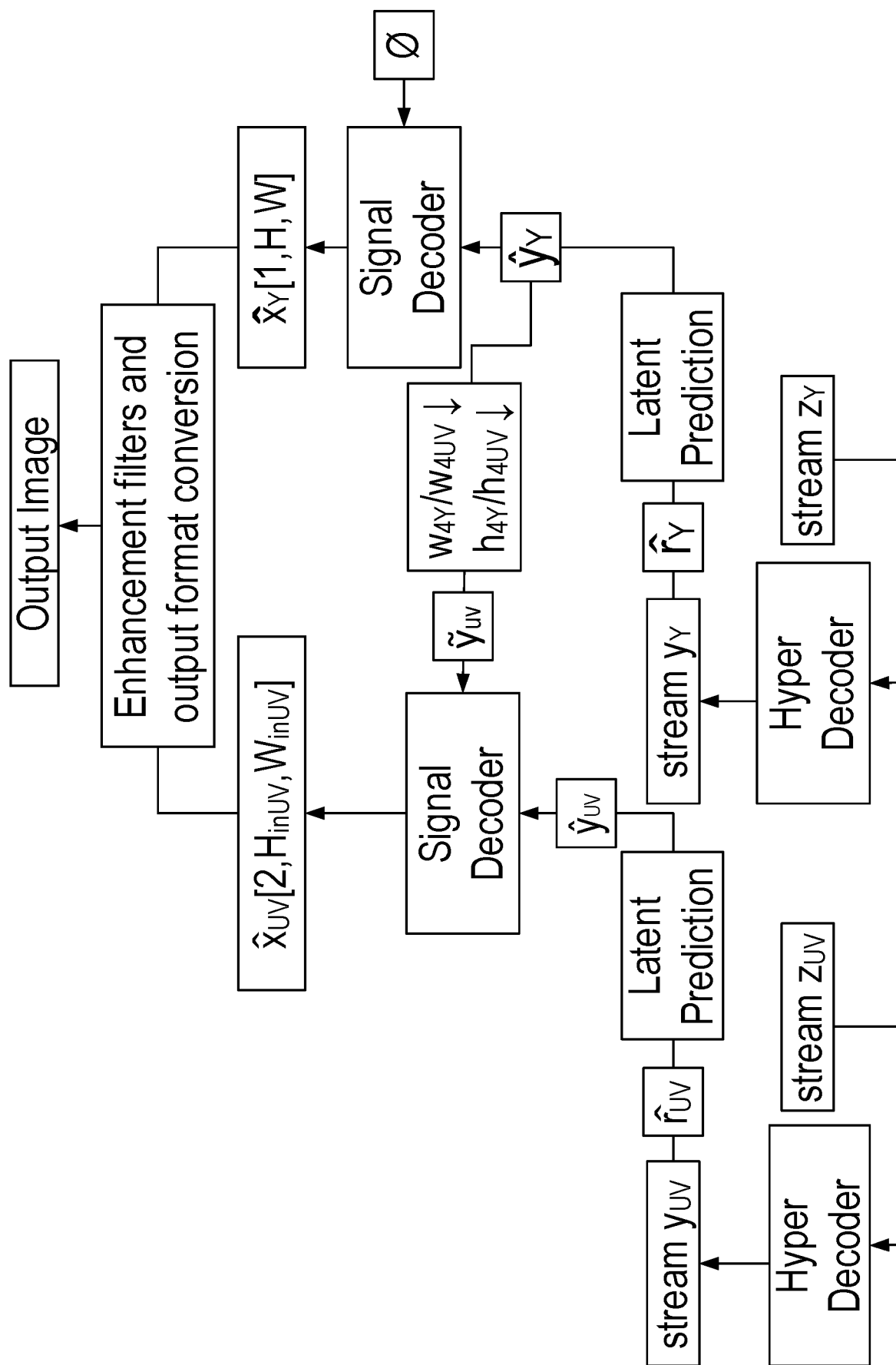


Fig. 23

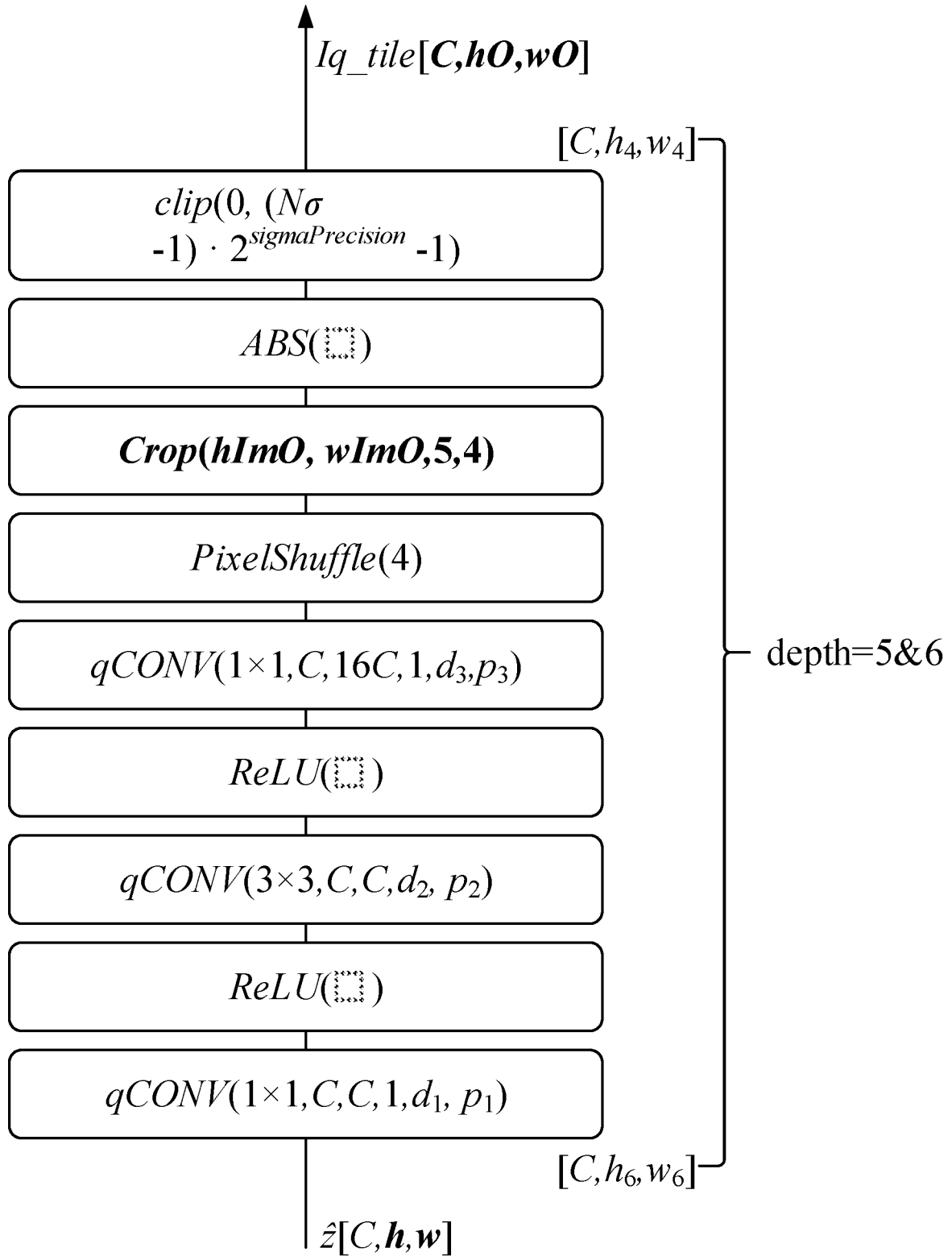


Fig. 24

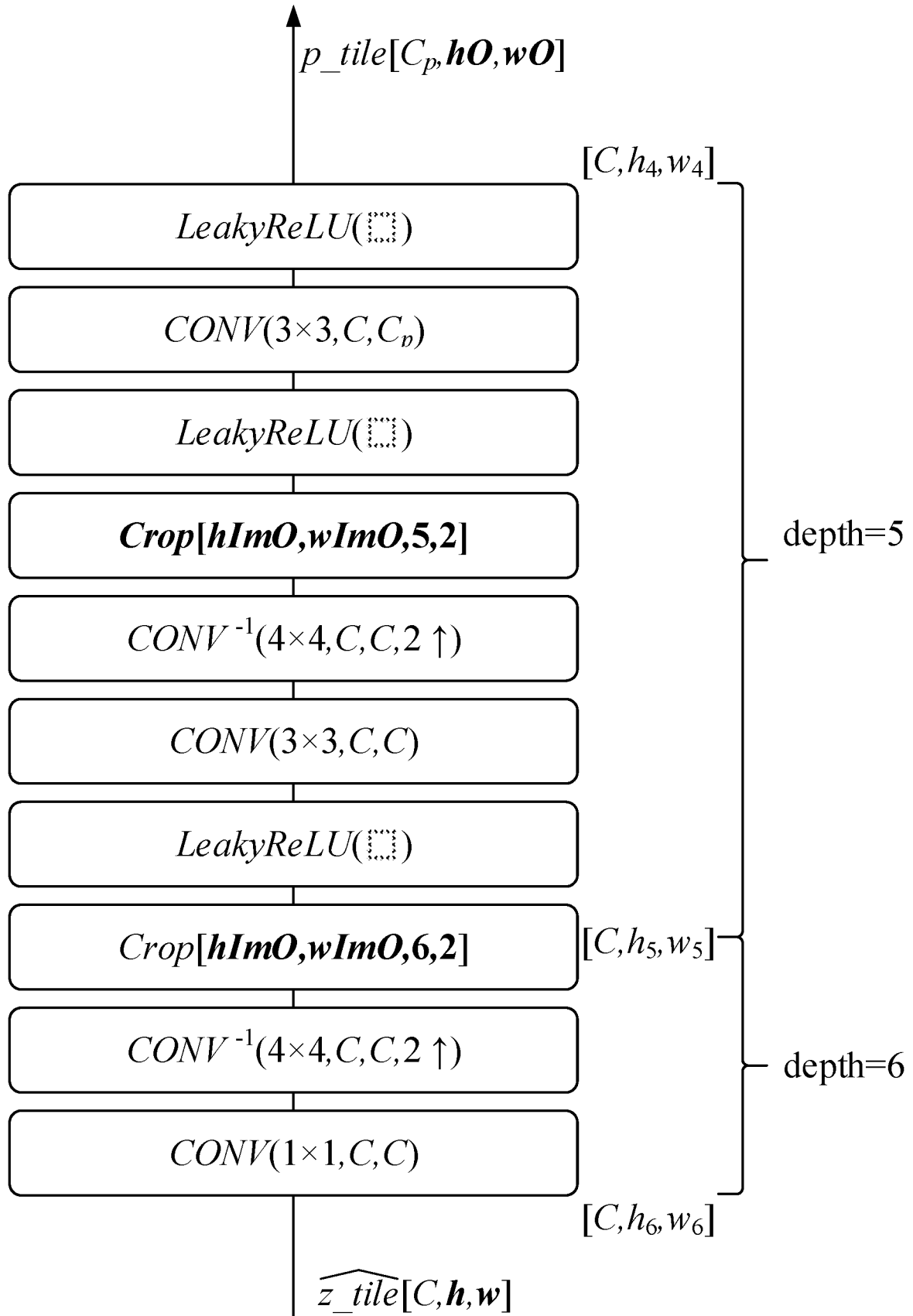


Fig. 25

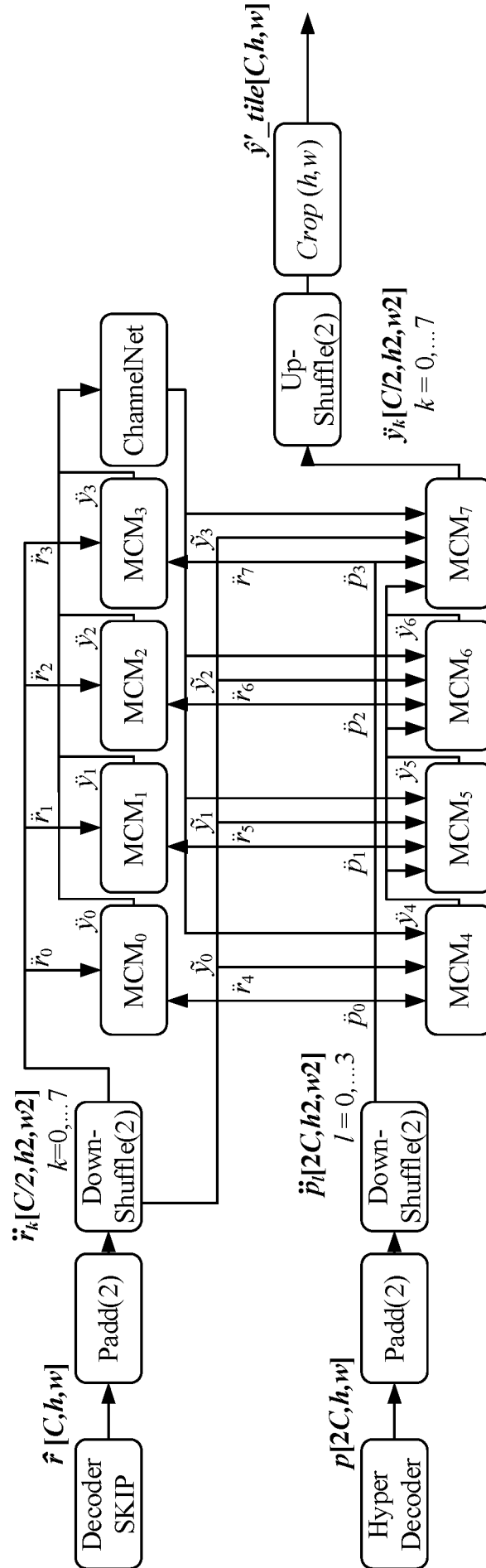


Fig. 26

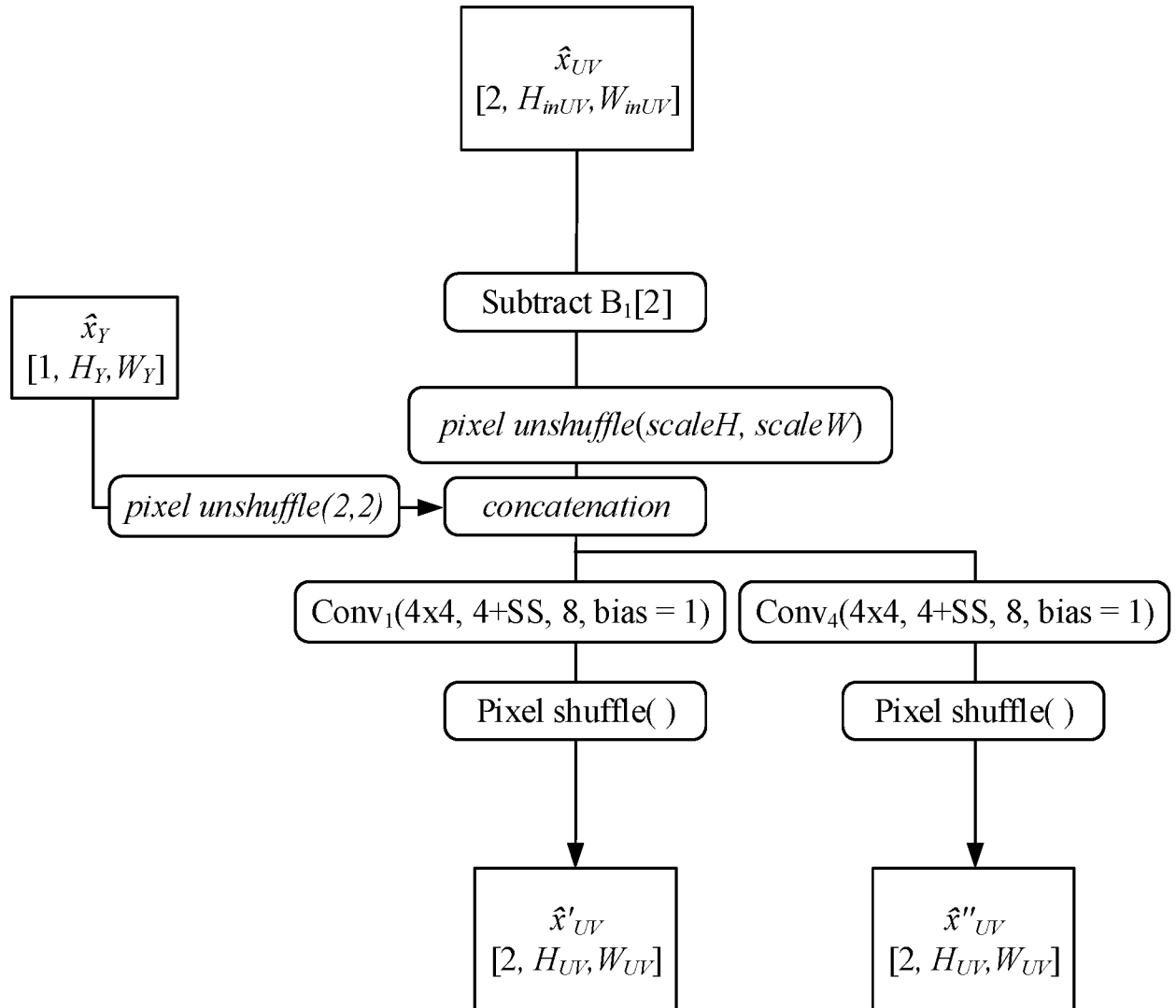
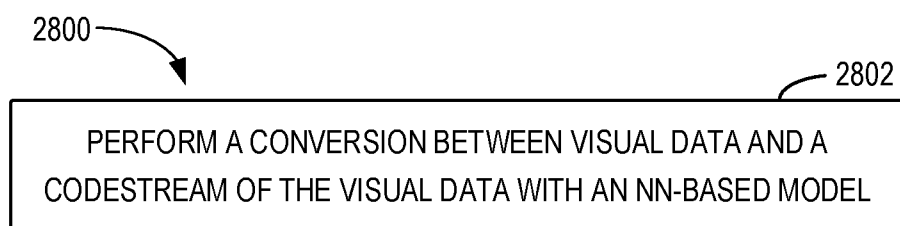
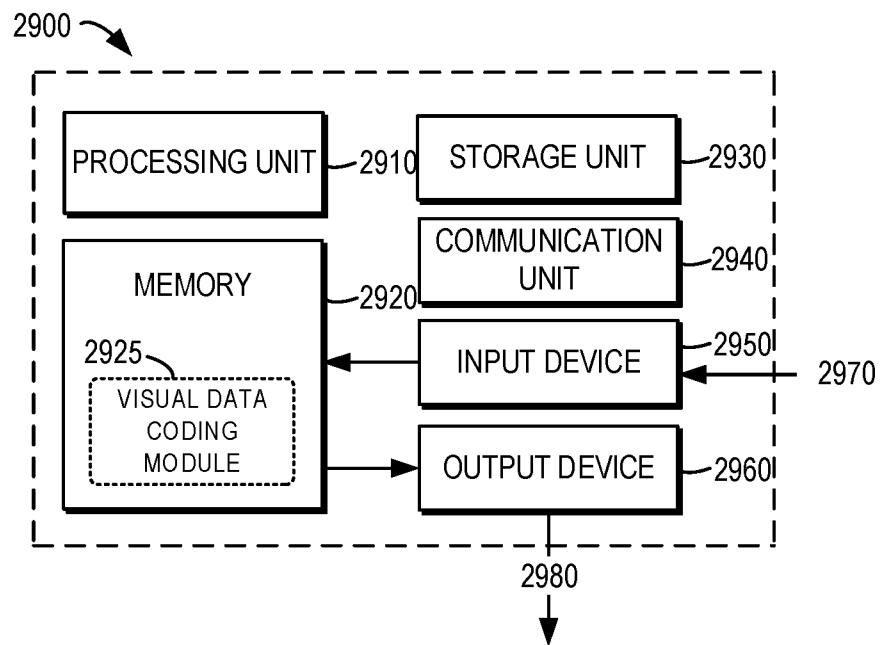


Fig. 27

**Fig. 28**

**Fig. 29**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2024/060832**Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)**

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☒ Claims Nos.: **5-58**
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2024/060832

A. CLASSIFICATION OF SUBJECT MATTERIPC: **H04N 19/102** (2025.01); **H04N 19/55** (2025.01); **H04N 19/59** (2025.01); **H04N 19/80** (2025.01); **H04N 19/117** (2025.01); **H04N 19/85** (2025.01); **G06N 3/02** (2025.01); **H04N 19/42** (2025.01); **G06N 3/08** (2025.01)CPC: **H04N 19/102**; **H04N 19/55**; **H04N 19/59**; **H04N 19/80**; **H04N 19/117**; **H04N 19/85**; **G06N 3/02**; **H04N 19/42**; **G06N 3/08**

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)

See Search History Document

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

See Search History Document

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

See Search History Document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2022/0295116 A1 (Intel Corporation) 15 September 2022 (15.09.2022) Figures 1-5; para: [0030]-[0036], [0059], [0062]	1-4, 59-60
A	US 2022/0385896 A1 (Tencent America LLC) 01 December 2022 (01.12.2022) Entire document	1-4, 59-60
A	US 2023/0336759 A1 (Huawei Technologies Co., LTD.) 19 October 2023 (19.10.2023) Entire document	1-4, 59-60
A	US 2022/0286695 A1 (Lemon Inc) 08 September 2022 (08.09.2022) entire document	1-4, 59-60

☐ 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

23 January 2025 (23.01.2025)

Date of mailing of the international search report

13 February 2025 (13.02.2025)

Name and mailing address of the ISA/US

COMMISSIONER FOR PATENTS
MAIL STOP PCT, ATTN: ISA/US
P.O. Box 1450
Alexandria, VA 22313-1450
UNITED STATES OF AMERICA

Facsimile No. 571-273-8300

Authorized officer

KARI RODRIQUEZ

Telephone No. PCT Help Desk: 571-272-4300