Method and apparatus for substream signalling in a bitstream

TECHNICAL FIELD

Embodiments of the present disclosure generally relate to the field of encoding and decoding databased on a neural network architecture. In particular, some embodiments relate to methods and apparatuses for such encoding and decoding images and/or videos from a bitstream using a plurality of processing layers.

BACKGROUND

Hybrid image and video codecs have been used for decades to compress image and video data. In such codecs, signal is typically encoded block-wisely by predicting a block and by further coding only the difference between the original bock and its prediction. In particular, such coding may include transformation, quantization and generating the bitstream, usually including some entropy coding. Typically, the three components of hybrid coding methods – transformation, quantization, and entropy coding – are separately optimized. Modern video compression standards like High-Efficiency Video Coding (HEVC), Versatile Video Coding (VVC) and Essential Video Coding (EVC) also use transformed representation to code residual signal after prediction.

Recently, neural network architectures have been applied to image and/or video coding. In general, these neural network (NN) based approaches can be applied in various different ways to the image and video coding. For example, some end-to-end optimized image or video coding frameworks have been discussed. Moreover, deep learning has been used to determine or optimize some parts of the end-to-end coding framework such as selection or compression of prediction parameters or the like. Besides, some neural network based approached have also been discussed for usage in hybrid image and video coding frameworks, e.g. for implementation as a trained deep learning model for intra or inter prediction in image or video coding.

The end-to-end optimized image or video coding applications discussed above have in common that they produce some feature map data, which is to be conveyed between encoder and decoder.

Neural networks are machine learning models that employ one or more layers of nonlinear units based on which they can predict an output for a received input. Some neural networks include one or more hidden layers in addition to an output layer. A corresponding feature map may be provided as an output of each hidden layer. Such corresponding feature map of each hidden layer may be used as an input to a subsequent layer in the network, i.e., a subsequent hidden layer or the output layer. Each layer of the network generates an output from a received input in accordance with current values of a respective set of parameters. In a neural network that is split between devices, e.g. between encoder and decoder, a device and a cloud or between different devices, a feature map at the output of the place of splitting (e.g. a first device) is compressed and transmitted to the remaining layers of the neural network (e.g. to a second device).

Further improvement of encoding and decoding using trained network architectures may be desirable.

The foregoing and other objects are achieved by the subject matter of the independent claims. Further implementation forms are apparent from the dependent claims, the description and the figures.

Particular embodiments are outlined in the attached independent claims, with other embodiments in the dependent claims.

SUMMARY

Particular embodiments are outlined in the attached independent claims, with other embodiments in the dependent claims.

According to a first aspect, the present disclosure relates to a decoding method, implemented by a decoder, the method comprising: receiving a bitstream including encoded data of an input signal, wherein the bitstream is formed of one or more substreams each comprising a size parameter defining a size of the substream; parsing at least one substream, the parsing comprising: determining a bit-length of the size parameter in dependence on a subset of bits included within the size parameter; parsing the size parameter to thereby determine a size of the substream. Advantageously, the decoder does not need to know, prior to decoding, how many bits the size parameter occupies; the decoder can advantageously determine the bit-length of the size parameter based only on the subset of bits within the size parameter. The bit-length of the size parameter is therefore defined by the size parameter itself.

The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination.

In a possible implementation each of the one or more substreams comprises an ID marker indicating a data type of the substream. This can help in interpreting the substreams. There may be a size parameter. It may define a size of the substream as a whole, or may define the size of the substream not including the portion of the substream containing the size parameter and/or the marker ID. This can further help to interpret the substream.

In a possible implementation a bit-length of the ID marker is predetermined. This can provide an efficient way of encoding the ID marker. The bit-length of the ID marker can be known to the decoder.

In a possible implementation parsing the at least one substream further comprises determining, in dependence on the ID marker, that the data type of the substream is unknown. This can allow appropriate action then to be taken.

In a possible implementation parsing the at least one substream further comprises, subsequent to determining the size of the substream, discarding data encoded within the substream in dependence on the determined size of the substream. This can allow for efficient decoding. Discarding the substream may be done in dependence on also determining that the data type of the substream is unknown based on the ID marker.

In a possible implementation the method further comprises, prior to determining the bit-length of the size parameter: determining that the size parameter of the substream is encoded in a size-defining format, wherein the subset of bits are configured, in the size-defining format, to define the bit-length of the size parameter. This can allow other forms of encoding to be used.

In a possible implementation parsing the size parameter further comprises: determining, starting from the left-most bit of the size parameter, a number of contiguous bits that are equal to zero; in dependence on determining the number of contiguous bits that are equal to zero, determining a code number encoded by the size parameter, wherein the code number defines the size of the substream. This can support an efficient way to convey the size of the substream. The code number may define a number of bytes. The number of bytes may be 1 fewer byte than the size of the substream in bytes.

In a possible implementation the number of contiguous bits that are equal to zero defines a bit-length of the code number. The code number may be defined by bits appended to the contiguous bits that are equal to zero. This can support an efficient way to define the code number. The size parameter may then be formed of a number of contiguous bits that are equal to zero, followed immediately by the code number.

In a possible implementation the size-defining format is an Exponential-Golomb coding format. In a possible implementation an order k of the Exponential-Golomb coding format is defined with order k equal to 0. These can be efficient ways to convey the size-defining format.

In a possible implementation each of the one or more substreams comprises a byte alignment buffer defining a buffer size. This can allow buffers of a similar size to be used at the encoder and decoder.

In a possible implementation parsing the at least one substream further comprises: determining a buffer size defined by the byte alignment buffer; parsing a remainder of the substream by parsing data starting from a starting bit in the substream, the starting bit positioned after the bits defining the size parameter by a number of bits defined by the buffer size. This can provide an efficient way to divide up processing of the substream.

In a possible implementation each of the one or more substreams comprises an ID marker indicating a data type encoded within the substream. A total number of bits forming i) the ID marker, ii) the size parameter, and iii) the byte alignment buffer, may be equal to an integer number of bytes. This can allow whole bytes to be used efficiently in the coding.

According to a second aspect, the present disclosure relates to a device for decoding data for picture or video processing from a bitstream, the device comprising: a receiving unit configured to: receive a bitstream including encoded data of an input signal, wherein the bitstream is formed of one or more substreams each comprising a size parameter defining a size of the substream; a parsing unit configured to parse at least one substream, the parsing comprising: determining a bit-length of the size parameter in dependence on a subset of bits included within size parameter; parsing the size parameter to thereby determine a size of the substream.

According to a third aspect, the present disclosure relates to an encoding method, implemented by an encoder, the method comprising: receiving an input signal; encoding the input signal into a bitstream comprising one or more substreams each comprising a size parameter defining a size of the substream, wherein the encoding comprises: generating, for at least one substream of the one or more substreams, the size parameter for the substream wherein a bit-length of the size parameter is defined by a subset of bits included within the size parameter.

The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination.

In a possible implementation the method comprises for each substream of the one or more substreams: generating an ID marker indicating a data type encoded within the substream; inserting the ID marker within the substream. The ID marker can usefully be employed at the decoder.

In a possible implementation the size parameter of the substream is encoded in a size-defining format, wherein the subset of bits are configured, in the size-defining format, to define the bit-length of the size parameter. This can effectively convey the size parameter to a decoder.

In a possible implementation the size parameter comprises one or more contiguous bits that are equal to zero which define a bit-length of a code number, wherein the code number defines the size of the substream and is defined by bits in the size parameter appended to the contiguous bits that are equal to zero. The code number can define a number of bytes. The number of bytes may be 1 fewer byte than the size of the substream in bytes. The size parameter comprises one or more contiguous bits

that are equal to zero starting from the left-most bit of the size parameter. Any one or more of these approaches can allow for efficient encoding of the respective information.

In a possible implementation the size-defining format is an Exponential-Golomb coding format. An order k of the Exponential-Golomb coding format may be defined with order k equal to 0. These can be efficient ways of encoding the size-defining format.

In a possible implementation the method further comprises for at least one substream of the one or more substreams: generating and inserting a byte alignment buffer, defining a buffer size, within the substream. Such a parameter can assist in decoding the substream.

In a possible implementation the byte alignment buffer is generated and inserted in dependence on determining that a total number of bits forming i) the ID marker and ii) the size parameter does not equal to an integer number of bytes. The alignment parameter can help to align the data.

In a possible implementation the byte alignment buffer is inserted in the substream immediately following the size parameter. This can assist a decoder to identify the parameter.

In a possible implementation each of the one or more substreams comprises an ID marker indicating a data type encoded within the substream, and wherein a total number of bits forming i) the ID marker, ii) the size parameter, and iii) the byte alignment buffer, is equal to an integer number of bytes. This can allow that data to be conveyed efficiently in a whole number of bytes.

According to a fourth aspect, the present disclosure relates to.a device for encoding data for picture or video processing to obtain a bitstream, the device comprising: a receiving unit configured to receive an input signal; and an encoding unit configured to: encode the input signal into a bitstream comprising one or more substreams each comprising a size parameter defining a size of the substream; generate, for at least one substream of the one or more substreams, the size parameter for the substream wherein a bit-length of the size parameter is defined by a subset of bits included within the size parameter.

The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination.

In a possible implementation the decoding apparatus comprises processing circuitry configured to execute steps of a method as set out above.

In a possible implementation the decoding apparatus comprises processing circuitry configured to execute steps of a method as set out above.

According to a fifth aspect, the present disclosure relates to a decoder comprising one or more processors and a non-transitory computer-readable storage medium coupled to the one or more processors, wherein the storage medium stores programming for execution by the one or more processors, wherein the programming, when executed by the one or more processors, configures the decoder to carry out a method as set out above.

According to a sixth aspect, the present disclosure relates to a storage medium storing in non-transitory form a bitstream encoded by a method as set out above.

According to a seventh aspect, the present disclosure relates to a computer program stored in non-transitory form and including code instructions, which, when executed on one or more processor, causes the one or more processor to execute a method as set out above.

According to an eighth aspect, the present disclosure relates to a system for delivering a bitstream, the system including at least one storage medium configured to store at least one bitstream generated by an encoding method as set out above.

According to a ninth aspect, the present disclosure relates to system for delivering a bitstream, the system comprising: at least one storage medium configured to store at least one bitstream generated by a method as set out above; and a video streaming device configured to obtain the bitstream from the at least one storage medium and send the bitstream to a terminal device, wherein the video streaming device comprises a content server or a content delivery server.

In a possible implementation the system further comprises one or more processor configured to perform encryption processing on at least one bitstream to obtain at least one encrypted bitstream, the at least one storage medium configured to store the encrypted bitstream; or the one or more processor configured to convert the bitstream in a first format into a bitstream in a second format, the at least one storage medium configured to store the bitstream in the second format. The storage medium can then store a usefully encoded form of data.

In a possible implementation the system further comprises: a receiver configured to receive a first operation request; wherein the one or more processor is configured to determine a target bitstream in the at least one storage medium in response to the first operation request; a transmitter configured to send the target bitstream to a terminal-side apparatus. The target bitstream can then be processed by the terminal side apparatus.

In a possible implementation the one or more processor is further configured to encapsulate the bitstream to obtain a transport stream in a first format, wherein the transmitter is further configured to: send the transport stream in the first format to a terminal-side apparatus for display; or send the transport stream in the first format to storage space for storage. In this way the transport stream can be processed as appropriate.

BRIEF DESCRIPTION OF THE DRAWINGS

In the following embodiments of the present disclosure are described in more detail with reference to the attached figures and drawings, in which

Fig. 1 is a schematic drawing illustrating channels processed by layers of a neural network;

Fig. 2 is a schematic drawing illustrating an autoencoder type of a neural network;

Fig. 3A is a schematic drawing illustrating an exemplary network architecture for encoder and decoder side including a hyperprior model;

Fig. 3B is a schematic drawing illustrating a general network architecture for encoder side including a hyperprior model;

Fig. 3C is a schematic drawing illustrating a general network architecture for decoder side including a hyperprior model;

Fig. 4 is a schematic drawing illustrating an exemplary network architecture for encoder and decoder side including a hyperprior model;

Fig. 5 is a block diagram illustrating a structure of a cloud-based solution for machine based tasks such as machine vision tasks;

Fig. 6A is a block diagram illustrating end-to-end video compression framework based on a neural networks;

Fig. 6B is a block diagram illustrating some exemplary details of application of a neural network for motion field compression;

Fig. 6C is a block diagram illustrating some exemplary details of application of a neural network for motion compensation;

Fig. 7 is a block diagram illustrating an example of an encoding apparatus or a decoding apparatus;

Fig. 8 is a block diagram illustrating another example of an encoding apparatus or a decoding apparatus;

Fig. 9 shows an example bitstream structure including reserved sections;

Fig. 10a shows part of a substream structure without a byte alignment buffer;

Fig. 10b shows part of a substream structure comprising a byte alignment buffer;

FIG. 11 is a flow diagram illustrating an exemplary method for decoding data such as image or video data or a component thereof;

Fig.12 is a flow diagram illustrating an exemplary method for encoding data such as image or video data or a component thereof;

FIG. 13 shows a device for decoding for processing by a neural network based unit;

FIG. 14 shows a device for encoding for processing by a neural network based unit;

FIG. 15 shows a bitstream structure;

FIG. 16 shows a substream structure;

Fig. 17 shows an example of a multicore coder;

Fig. 18 is a block diagram showing an example of a video coding system configured to implement embodiments of the present disclosure;

Fig. 19 is a block diagram showing another example of a video coding system configured to implement embodiments of the present disclosure;

Fig. 20 is a block diagram illustrating an example of an encoding apparatus or a decoding apparatus;

Fig. 21 is a block diagram illustrating another example of an encoding apparatus or a decoding apparatus; and

Fig. 22 is a block diagram illustrating another example of an encoding apparatus or a decoding apparatus.

Like reference numbers and designations in different drawings may indicate similar elements.

DETAILED DESCRIPTION OF THE EMBODIMENTS

In the following description, reference is made to the accompanying figures, which form part of the disclosure, and which show, by way of illustration, specific aspects of embodiments of the present disclosure or specific aspects in which embodiments of the present disclosure may be used. It is understood that embodiments of the present disclosure may be used in other aspects and comprise structural or logical changes not depicted in the figures. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present disclosure is defined by the appended claims.

For instance, it is understood that a disclosure in connection with a described method may also hold true for a corresponding device or system configured to perform the method and vice versa. For example, if one or a plurality of specific method steps are described, a corresponding device may include one or a plurality of units, e.g. functional units, to perform the described one or plurality of method steps (e.g. one unit performing the one or plurality of steps, or a plurality of units each performing one or more of the plurality of steps), even if such one or more units are not explicitly described or illustrated in the figures. On the other hand, for example, if a specific apparatus is described based on one or a plurality of units, e.g. functional units, a corresponding method may include one step to perform the functionality of the one or plurality of units (e.g. one step performing the functionality of the one or plurality of units, or a plurality of steps each performing the functionality of one or more of the plurality of units), even if such one or plurality of steps are not explicitly described or illustrated in the figures. Further, it is understood that the features of the various exemplary embodiments and/or aspects described herein may be combined with each other, unless specifically noted otherwise.

In the following, an overview over some of the used technical terms and framework within which the embodiments of the present disclosure may be employed is provided.

Artificial neural networks

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. Over time, attention moved to performing specific tasks, leading to deviations from biology. ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games, medical diagnosis, and even in activities that have traditionally been considered as reserved to humans, like painting.

The name “convolutional neural network” (CNN) indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are neural networks that use convolution in place of a general matrix multiplication in at least one of their layers.

Fig. 1 schematically illustrates a general concept of processing by a neural network such as the CNN. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. Input layer is the layer to which the input (such as a portion 11 of an input image as shown in Fig. 1) is provided for processing. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The result of a layer is one or more feature maps (illustrated by empty solid-line rectangles), sometimes also referred to as channels. There may be a resampling (such as subsampling) involved in some or all of the layers. As a consequence, the feature maps may become smaller, as illustrated in Fig. 1. It is noted that a convolution with a stride may also reduce the size (resample) an input feature map. The activation function in a CNN is usually a ReLU (Rectified Linear Unit) layer or Leaky ReLU, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. Though the layers are colloquially referred to as convolutions, this is only by convention. Mathematically, it is technically a sliding dot product or cross-correlation. This has significance for the indices in the matrix, in that it affects how the weight is determined at a specific index point.

When programming a CNN for processing images, as shown in Fig. 1, the input is a tensor with shape (number of images) x (image width) x (image height) x (image depth). It should be known that the image depth can be constituted by channels of an image. After passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map width) x (feature map height) x (feature map channels). A convolutional layer within a neural network should have the following attributes. Convolutional kernels defined by a width and height (hyper-parameters). The number of input channels and output channels (hyper-parameter). The depth of the convolution filter (the input channels) should be equal to the number channels (depth) of the input feature map.

In the past, traditional multilayer perceptron (MLP) models have been used for image recognition. However, due to the full connectivity between nodes, they suffered from high dimensionality, and did not scale well with higher resolution images. A 1000×1000-pixel image with RGB color channels has 3 million weights, which is too high to feasibly process efficiently at scale with full connectivity. Also, such network architecture does not take into account the spatial structure of data, treating input pixels which are far apart in the same way as pixels that are close together. This ignores locality of reference in image data, both computationally and semantically. Thus, full connectivity of neurons is wasteful for purposes such as image recognition that are dominated by spatially local input patterns.

Convolutional neural networks are biologically inspired variants of multilayer perceptrons that are specifically designed to emulate the behavior of a visual cortex. These models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (the above-mentioned kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map. A feature map, or activation map, is the output activations for a given filter. Feature map and activation has same meaning. In some papers it is called an activation map because it is a mapping that corresponds to the activation of different parts of the image, and also a feature map because it is also a mapping of where a certain kind of feature is found in the image. A high activation means that a certain feature was found.

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains

unchanged. In addition to max pooling, pooling units can use other functions, such as average pooling or ℓ2-norm pooling. Average pooling was often used historically but has recently fallen out of favour compared to max pooling, which often performs better in practice. Due to the aggressive reduction in the size of the representation, there is a recent trend towards using smaller filters or discarding pooling layers altogether. "Region of Interest" pooling (also known as ROI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter. Pooling is an important component of convolutional neural networks for object detection based on Fast R-CNN architecture.

The above-mentioned ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function. It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent and the sigmoid function. ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

Leaky Rectified Linear Unit, or Leaky ReLU, is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope. The slope coefficient is determined before training, i.e. it is not learnt during training. This type of activation function is popular in tasks where it suffers from sparse gradients, for example training generative adversarial networks. Leaky ReLU applies the element-wise function:

LeakyReLU(x)=max(0,x)+negative_slope∗min(0,x), or

$$\text{LeakyReLU}(x) = \begin{cases} x, & if\ x \geq 0 \\ negative_slope * x, & otherwise \end{cases}$$

Among them, parameters:
negative_slope – Controls the angle of the negative slope. Default: 1e-2
inplace – can optionally do the operation in-place. Default: False.

After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

The "loss layer" (including calculating of a loss function) specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions appropriate for different tasks may be used. Softmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0, 1]. Euclidean loss is used for regressing to real-valued labels. In summary, Fig. 1 shows the data flow in a typical convolutional neural network. First, the input image is passed through convolutional layers and becomes abstracted to a feature map comprising several channels, corresponding to a number of filters in a set of learnable filters of this layer. Then, the feature map is subsampled using e.g. a pooling layer, which reduces the dimension of each channel in the feature map. Next, the data comes to another convolutional layer, which may have different numbers of output channels. As was mentioned above, the number of input channels and output channels are hyper-parameters of the layer. To establish connectivity of the network, those parameters need to be synchronized between two connected layers, such that the number of input channels for the current layers should be equal to the number of output channels of the previous layer. For the first layer which processes input data, e.g. an image, the number of input channels is normally equal to the number of channels of data representation, for instance 3 channels for RGB or YUV representation of images or video, or 1 channel

for grayscale image or video representation. The channels obtained by one or more convolutional layers (and possibly resampling layer(s)) may be passed to an output layer. Such output layer may be a convolutional or resampling in some implementations. In an exemplary and non-limiting implementation, the output layer is a fully connected layer.

Autoencoders and unsupervised learning

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. A schematic drawing thereof is shown in Fig. 2. The autoencoder includes an encoder side 210 with an input x inputted into an input layer of an encoder subnetwork 220 and a decoder side 250 with output x' outputted from a decoder subnetwork 260. The aim of an autoencoder is to learn a representation (encoding) 230 for a set of data x, typically for dimensionality reduction, by training the network 220, 260 to ignore signal "noise". Along with the reduction (encoder) side subnetwork 220, a reconstructing (decoder) side subnetwork 260 is learnt, where the autoencoder tries to generate from the reduced encoding 230 a representation x' as close as possible to its original input x, hence its name. In the simplest case, given one hidden layer, the encoder stage of an autoencoder takes the input x and maps it to h

$$h = \sigma(Wx + b).$$

This image h is usually referred to as code 230, latent variables, or latent representation. Here, σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. W is a weight matrix b is a bias vector. Weights and biases are usually initialized randomly, and then updated iteratively during training through Backpropagation. After that, the decoder stage of the autoencoder maps h to the reconstruction x' of the same shape as x:

$$x' = \sigma'(W'h' + b')$$

where σ', W' and b' for the decoder may be unrelated to the corresponding σ, W and b for the encoder.

Variational autoencoder models make strong assumptions concerning the distribution of latent variables. They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes (SGVB) estimator. It assumes that the data is generated by a directed graphical model $p_\theta(x|h)$ and that the encoder is learning an approximation $q_\phi(h|x)$ to the posterior distribution $p_\theta(h|x)$ where ϕ and θ denote the parameters of the encoder (recognition model) and decoder (generative model) respectively. The probability distribution of the latent vector of a VAE typically matches that of the training data much closer than a standard autoencoder. The objective of VAE has the following form:

$$\mathcal{L}(\phi, \theta, x) = D_{KL}(q_\phi(h|x)||p_\theta(h)) - E_{q_\phi(h|x)}(log\ p_\theta(x|h))$$

Here, D_{KL} stands for the Kullback–Leibler divergence. The prior over the latent variables is usually set to be the centered isotropic multivariate Gaussian $p_\theta(h) = \mathcal{N}(0, I)$. Commonly, the shape of the variational and the likelihood distributions are chosen such that they are factorized Gaussians:

$$q_\phi(h|x) = \mathcal{N}(\rho(x), \omega^2(x)I)$$
$$p_\phi(x|h) = \mathcal{N}(\mu(h), \sigma^2(h)I)$$

where $\rho(x)$ and $\omega^2(x)$ are the encoder output, while $\mu(h)$ and $\sigma^2(h)$ are the decoder outputs.

Recent progress in artificial neural networks area and especially in convolutional neural networks enables researchers' interest of applying neural networks based technologies to the task of image and video compression. For example, End-to-end Optimized Image Compression has been proposed, which uses a network based on a variational autoencoder.

Accordingly, data compression is considered as a fundamental and well-studied problem in engineering, and is commonly formulated with the goal of designing codes for a given discrete data ensemble with minimal entropy. The solution relies

heavily on knowledge of the probabilistic structure of the data, and thus the problem is closely related to probabilistic source modeling. However, since all practical codes must have finite entropy, continuous-valued data (such as vectors of image pixel intensities) must be quantized to a finite set of discrete values, which introduces an error.

In this context, known as the lossy compression problem, one must trade off two competing costs: the entropy of the discretized representation (rate) and the error arising from the quantization (distortion). Different compression applications, such as data storage or transmission over limited-capacity channels, demand different rate–distortion trade-offs.

Joint optimization of rate and distortion is difficult. Without further constraints, the general problem of optimal quantization in high-dimensional spaces is intractable. For this reason, most existing image compression methods operate by linearly transforming the data vector into a suitable continuous-valued representation, quantizing its elements independently, and then encoding the resulting discrete representation using a lossless entropy code. This scheme is called transform coding due to the central role of the transformation.

For example, JPEG uses a discrete cosine transform on blocks of pixels, and JPEG 2000 uses a multi-scale orthogonal wavelet decomposition. Typically, the three components of transform coding methods – transform, quantizer, and entropy code – are separately optimized (often through manual parameter adjustment). Modern video compression standards like HEVC, VVC and EVC also use transformed representation to code residual signal after prediction. The several transforms are used for that purpose such as discrete cosine and sine transforms (DCT, DST), as well as low frequency non-separable manually optimized transforms (LFNST).

Variational image compression

Variable Auto-Encoder (VAE) framework can be considered as a nonlinear transforming coding model. The transforming process can be mainly divided into four parts. This is exemplified in Fig. 3A showing a VAE framework.

The transforming process can be mainly divided into four parts: Fig. 3A exemplifies the VAE framework. In Fig. 3A, the encoder 101 maps an input image x into a latent representation (denoted by y) via the function y = f (x). This latent representation may also be referred to as a part of or a point within a "latent space" in the following. The function f() is a transformation function that converts the input signal x into a more compressible representation y. The quantizer 102 transforms the latent representation y into the quantized latent representation $\hat{y}$ with (discrete) values by $\hat{y} = Q(y)$, with Q representing the quantizer function. The entropy model, or the hyper encoder/decoder (also known as hyperprior) 103 estimates the distribution of the quantized latent representation $\hat{y}$ to get the minimum rate achievable with a lossless entropy source coding.

The latent space can be understood as a representation of compressed data in which similar data points are closer together in the latent space. Latent space is useful for learning data features and for finding simpler representations of data for analysis. The quantized latent representation T, $\hat{y}$ and the side information $\hat{z}$ of the hyperprior 3 are included into a bitstream 2 (are binarized) using arithmetic coding (AE). Furthermore, a decoder 104 is provided that transforms the quantized latent representation to the reconstructed image $\hat{x}$, $\hat{x} = g(\hat{y})$. The signal $\hat{x}$ is the estimation of the input image x. It is desirable that x is as close to $\hat{x}$ as possible, in other words the reconstruction quality is as high as possible. However, the higher the similarity between $\hat{x}$ and x, the higher the amount of side information necessary to be transmitted. The side information includes bitstream1 and bitstream2 shown in Fig. 3A, which are generated by the encoder and transmitted to the decoder. Normally, the higher the amount of side information, the higher the reconstruction quality. However, a high amount of side information means that the compression ratio is low. Therefore, one purpose of the system described in Fig. 3A is to balance the reconstruction quality and the amount of side information conveyed in the bitstream.

In Fig. 3A the component AE 105 is the Arithmetic Encoding module, which converts samples of the quantized latent representation $\hat{y}$ and the side information $\hat{z}$ into a binary representation bitstream 1. The samples of $\hat{y}$ and $\hat{z}$ might for example comprise integer or floating point numbers. One purpose of the arithmetic encoding module is to convert (via the process of binarization) the sample values into a string of binary digits (which is then included in the bitstream that may comprise further portions corresponding to the encoded image or further side information).

The arithmetic decoding (AD) 106 is the process of reverting the binarization process, where binary digits are converted back to sample values. The arithmetic decoding is provided by the arithmetic decoding module 106.

It is noted that the present disclosure is not limited to this particular framework. Moreover, the present disclosure is not restricted to image or video compression, and can be applied to object detection, image generation, and recognition systems as well.

In Fig. 3A there are two sub networks concatenated to each other. A subnetwork in this context is a logical division between the parts of the total network. For example, in Fig. 3A the modules 101, 102, 104, 105 and 106 are called the "Encoder/Decoder" subnetwork. The "Encoder/Decoder" subnetwork is responsible for encoding (generating) and decoding (parsing) of the first bitstream "bitstream1". The second network in Fig. 3A comprises modules 103, 108, 109, 110 and 107 and is called "hyper encoder/decoder" subnetwork. The second subnetwork is responsible for generating the second bitstream "bitstream2". The purposes of the two subnetworks are different.

The first subnetwork is responsible for:

- the transformation 101 of the input image x into its latent representation y (which is easier to compress that x),
- quantizing 102 the latent representation y into a quantized latent representation $\hat{y}$,
- compressing the quantized latent representation $\hat{y}$ using the AE by the arithmetic encoding module 105 to obtain bitstream "bitstream 1",".
- parsing the bitstream 1 via AD using the arithmetic decoding module 106, and
- reconstructing 104 the reconstructed image ($\hat{x}$) using the parsed data.

The purpose of the second subnetwork is to obtain statistical properties (e.g. mean value, variance and correlations between samples of bitstream 1) of the samples of "bitstream1", such that the compressing of bitstream 1 by first subnetwork is more efficient. The second subnetwork generates a second bitstream "bitstream2", which comprises the said information (e.g. mean value, variance and correlations between samples of bitstream1).

The second network includes an encoding part which comprises transforming 103 of the quantized latent representation $\hat{y}$ into side information z, quantizing the side information z into quantized side information $\hat{z}$, and encoding (e.g. binarizing) 109 the quantized side information $\hat{z}$ into bitstream2. In this example, the binarization is performed by an arithmetic encoding (AE). A decoding part of the second network includes arithmetic decoding (AD) 110, which transforms the input bitstream2 into decoded quantized side information $\hat{z}'$. The $\hat{z}'$ might be identical to $\hat{z}$, since the arithmetic encoding end decoding operations are lossless compression methods. The decoded quantized side information $\hat{z}'$ is then transformed 107 into decoded side information $\hat{y}'$. $\hat{y}'$ represents the statistical properties of $\hat{y}$ (e.g. mean value of samples of $\hat{y}$, or the variance of sample values or like). The decoded latent representation $\hat{y}'$ is then provided to the above-mentioned Arithmetic Encoder 105 and Arithmetic Decoder 106 to control the probability model of $\hat{y}$.

The Fig. 3A describes an example of VAE (variational auto encoder), details of which might be different in different implementations. For example in a specific implementation additional components might be present to more efficiently obtain

the statistical properties of the samples of bitstream 1. In one such implementation a context modeler might be present, which targets extracting cross-correlation information of the bitstream 1. The statistical information provided by the second subnetwork might be used by AE (arithmetic encoder) 105 and AD (arithmetic decoder) 106 components.

Fig. 3A depicts the encoder and decoder in a single figure. As is clear to those skilled in the art, the encoder and the decoder may be, and very often are, embedded in mutually different devices.

Fig. 3B depicts the encoder and Fig. 3C depicts the decoder components of the VAE framework in isolation. As input, the encoder receives, according to some embodiments, a picture. The input picture may include one or more channels, such as color channels or other kind of channels, e.g. depth channel or motion information channel, or the like. The output of the encoder (as shown in Fig. 3B) is a bitstream1 and a bitstream2. The bitstream1 is the output of the first sub-network of the encoder and the bitstream2 is the output of the second subnetwork of the encoder.

Similarly, in Fig. 3C, the two bitstreams, bitstream1 and bitstream2, are received as input and $\hat{z}$, which is the reconstructed (decoded) image, is generated at the output. As indicated above, the VAE can be split into different logical units that perform different actions. This is exemplified in Figs. 3B and 3C so that Fig. 3B depicts components that participate in the encoding of a signal, like a video and provided encoded information. This encoded information is then received by the decoder components depicted in Fig. 3C for encoding, for example. It is noted that the components of the encoder and decoder denoted with numerals 12x and 14x may correspond in their function to the components referred to above in Fig. 3A and denoted with numerals 10x.

Specifically, as is seen in Fig. 3B, the encoder comprises the encoder 121 that transforms an input x into a signal y which is then provided to the quantizer 322. The quantizer 122 provides information to the arithmetic encoding module 125 and the hyper encoder 123. The hyper encoder 123 provides the bitstream2 already discussed above to the hyper decoder 147 that in turn provides the information to the arithmetic encoding module 105 (125).

The output of the arithmetic encoding module is the bitstream1. The bitstream1 and bitstream2 are the output of the encoding of the signal, which are then provided (transmitted) to the decoding process. Although the unit 101 (121) is called “encoder”, it is also possible to call the complete subnetwork described in Fig. 3B as “encoder”. The process of encoding in general means the unit (module) that converts an input to an encoded (e.g. compressed) output. It can be seen from Fig. 3B, that the unit 121 can be actually considered as a core of the whole subnetwork, since it performs the conversion of the input x into y, which is the compressed version of the x. The compression in the encoder 121 may be achieved, e.g. by applying a neural network, or in general any processing network with one or more layers. In such network, the compression may be performed by cascaded processing including downsampling which reduces size and/or number of channels of the input. Thus, the encoder may be referred to, e.g. as a neural network (NN) based encoder, or the like.

The remaining parts in the figure (quantization unit, hyper encoder, hyper decoder, arithmetic encoder/decoder) are all parts that either improve the efficiency of the encoding process or are responsible for converting the compressed output y into a series of bits (bitstream). Quantization may be provided to further compress the output of the NN encoder 121 by a lossy compression. The AE 125 in combination with the hyper encoder 123 and hyper decoder 127 used to configure the AE 125 may perform the binarization which may further compress the quantized signal by a lossless compression. Therefore, it is also possible to call the whole subnetwork in Fig. 3B an “encoder”.

A majority of Deep Learning (DL) based image/video compression systems reduce dimensionality of the signal before converting the signal into binary digits (bits). In the VAE framework for example, the encoder, which is a non-linear transform, maps the input image x into y, where y has a smaller width and height than x. Since the y has a smaller width and height, hence

a smaller size, the (size of the) dimension of the signal is reduced, and, hence, it is easier to compress the signal y. It is noted that in general, the encoder does not necessarily need to reduce the size in both (or in general all) dimensions. Rather, some exemplary implementations may provide an encoder which reduces size only in one (or in general a subset of) dimension.

In J. Balle, L. Valero Laparra, and E. P. Simoncelli (2015). "Density Modeling of Images Using a Generalized Normalization Transformation", In: arXiv e-prints, Presented at the 4th Int. Conf. for Learning Representations, 2016 (referred to in the following as "Balle") the authors proposed a framework for end-to-end optimization of an image compression model based on nonlinear transforms. The authors optimize for Mean Squared Error (MSE), but use a more flexible transforms built from cascades of linear convolutions and nonlinearities. Specifically, authors use a generalized divisive normalization (GDN) joint nonlinearity that is inspired by models of neurons in biological visual systems, and has proven effective in Gaussianizing image densities. This cascaded transformation is followed by uniform scalar quantization (i.e., each element is rounded to the nearest integer), which effectively implements a parametric form of vector quantization on the original image space. The compressed image is reconstructed from these quantized values using an approximate parametric nonlinear inverse transform.

Such example of the VAE framework is shown in Fig. 4, and it utilizes 6 downsampling layers that are marked with 401 to 406. The network architecture includes a hyperprior model. The left side (g_a, g_s) shows an image autoencoder architecture, the right side (h_a, h_s) corresponds to the autoencoder implementing the hyperprior. The factorized-prior model uses the identical architecture for the analysis and synthesis transforms g_a and g_s. Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder, respectively. The encoder subjects the input image x to g_a, yielding the responses y (latent representation) with spatially varying standard deviations. The encoding g_a includes a plurality of convolution layers with subsampling and, as an activation function, generalized divisive normalization (GDN).

The responses are fed into h_a, summarizing the distribution of standard deviations in z. z is then quantized, compressed, and transmitted as side information. The encoder then uses the quantized vector $\hat{z}$ to estimate $\hat{\sigma}$, the spatial distribution of standard deviations which is used for obtaining probability values (or frequency values) for arithmetic coding (AE), and uses it to compress and transmit the quantized image representation $\hat{y}$ (or latent representation). The decoder first recovers $\hat{z}$ from the compressed signal. It then uses h_s to obtain $\hat{y}$, which provides it with the correct probability estimates to successfully recover $\hat{y}$ as well. It then feeds $\hat{y}$ into g_s to obtain the reconstructed image.

The layers that include downsampling is indicated with the downward arrow in the layer description. The layer description „Conv N,k1,2↓" means that the layer is a convolution layer, with N channels and the convolution kernel is k1xk1 in size. For example, k1 may be equal to 5 and k2 may be equal to 3. As stated, the 2↓means that a downsampling with a factor of 2 is performed in this layer. Downsampling by a factor of 2 results in one of the dimensions of the input signal being reduced by half at the output. In Fig. 4, the 2↓indicates that both width and height of the input image is reduced by a factor of 2. Since there are 6 downsampling layers, if the width and height of the input image 414 (also denoted with x) is given by w and h, the output signal z^413 is has width and height equal to w/64 and h/64 respectively. Modules denoted by AE and AD are arithmetic encoder and arithmetic decoder, which are explained with reference to Figs. 3A to 3C. The arithmetic encoder and decoder are specific implementations of entropy coding. AE and AD can be replaced by other means of entropy coding. In information theory, an entropy encoding is a lossless data compression scheme that is used to convert the values of a symbol into a binary representation which is a revertible process. Also, the "Q" in the figure corresponds to the quantization operation that was also referred to above in relation to Fig. 4 and is further explained above in the section "Quantization". Also, the quantization operation and a corresponding quantization unit as part of the component 413 or 415 is not necessarily present and/or can be replaced with another unit.

In Fig. 4, there is also shown the decoder comprising upsampling layers 407 to 412. A further layer 420 is provided between the upsampling layers 411 and 410 in the processing order of an input that is implemented as convolutional layer but does not provide an upsampling to the input received. A corresponding convolutional layer 430 is also shown for the decoder. Such layers can be provided in NNs for performing operations on the input that do not alter the size of the input but change specific characteristics. However, it is not necessary that such a layer is provided.

When seen in the processing order of bitstream2 through the decoder, the upsampling layers are run through in reverse order, i.e. from upsampling layer 412 to upsampling layer 407. Each upsampling layer is shown here to provide an upsampling with an upsampling ratio of 2, which is indicated by the ↑. It is, of course, not necessarily the case that all upsampling layers have the same upsampling ratio and also other upsampling ratios like 3, 4, 8 or the like may be used. The layers 407 to 412 are implemented as convolutional layers (conv). Specifically, as they may be intended to provide an operation on the input that is reverse to that of the encoder, the upsampling layers may apply a deconvolution operation to the input received so that its size is increased by a factor corresponding to the upsampling ratio. However, the present disclosure is not generally limited to deconvolution and the upsampling may be performed in any other manner such as by bilinear interpolation between two neighboring samples, or by nearest neighbor sample copying, or the like.

In the first subnetwork, some convolutional layers (401 to 403) are followed by generalized divisive normalization (GDN) at the encoder side and by the inverse GDN (IGDN) at the decoder side. In the second subnetwork, the activation function applied is ReLu. It is noted that the present disclosure is not limited to such implementation and in general, other activation functions may be used instead of GDN or ReLu.

Cloud solutions for machine tasks

The Video Coding for Machines (VCM) is another computer science direction being popular nowadays. The main idea behind this approach is to transmit a coded representation of image or video information targeted to further processing by computer vision (CV) algorithms, like object segmentation, detection and recognition. In contrast to traditional image and video coding targeted to human perception the quality characteristic is the performance of computer vision task, e.g. object detection accuracy, rather than reconstructed quality. This is illustrated in Fig. 5.

Video Coding for Machines is also referred to as collaborative intelligence and it is a relatively new paradigm for efficient deployment of deep neural networks across the mobile-cloud infrastructure. By dividing the network between the mobile side 510 and the cloud side 590 (e.g. a cloud server), it is possible to distribute the computational workload such that the overall energy and/or latency of the system is minimized. In general, the collaborative intelligence is a paradigm where processing of a neural network is distributed between two or more different computation nodes; for example devices, but in general, any functionally defined nodes. Here, the term “node” does not refer to the above-mentioned neural network nodes. Rather the (computation) nodes here refer to (physically or at least logically) separate devices/modules, which implement parts of the neural network. Such devices may be different servers, different end user devices, a mixture of servers and/or user devices and/or cloud and/or processor or the like. In other words, the computation nodes may be considered as nodes belonging to the same neural network and communicating with each other to convey coded data within/for the neural network. For example, in order to be able to perform complex computations, one or more layers may be executed on a first device (such as a device on mobile side 510) and one or more layers may be executed in another device (such as a cloud server on cloud side 590). However, the distribution may also be finer and a single layer may be executed on a plurality of devices. In this disclosure, the term “plurality” refers to two or more. In some existing solution, a part of a neural network functionality is executed in a device (user device or edge device or the like) or a plurality of such devices and then the output (feature map) is passed to a cloud. A cloud is a collection of processing or computing systems that are located outside the device, which is operating the part of the neural network. The notion of collaborative intelligence has been extended to model training as well. In this case, data flows

both ways: from the cloud to the mobile during back-propagation in training, and from the mobile to the cloud (illustrated in Fig. 5) during forward passes in training, as well as inference.

Some works presented semantic image compression by encoding deep features and then reconstructing the input image from them. The compression based on uniform quantization was shown, followed by context-based adaptive arithmetic coding (CABAC) from H.264. In some scenarios, it may be more efficient, to transmit from the mobile part 510 to the cloud 590 an output of a hidden layer (a deep feature map) 550, rather than sending compressed natural image data to the cloud and perform the object detection using reconstructed images. It may thus be advantageous to compress the data (features) generated by the mobile side 510, which may include a quantization layer 520 for this purpose. Correspondingly, the cloud side 590 may include an inverse quantization layer 560. The efficient compression of feature maps benefits the image and video compression and reconstruction both for human perception and for machine vision. Entropy coding methods, e.g. arithmetic coding is a popular approach to compression of deep features (i.e. feature maps).

Nowadays, video content contributes to more than 80% internet traffic, and the percentage is expected to increase even further. Therefore, it is critical to build an efficient video compression system and generate higher quality frames at given bandwidth budget. In addition, most video related computer vision tasks such as video object detection or video object tracking are sensitive to the quality of compressed videos, and efficient video compression may bring benefits for other computer vision tasks. Meanwhile, the techniques in video compression are also helpful for action recognition and model compression. However, in the past decades, video compression algorithms rely on hand-crafted modules, e.g., block based motion estimation and Discrete Cosine Transform (DCT), to reduce the redundancies in the video sequences, as mentioned above. Although each module is well designed, the whole compression system is not end-to-end optimized. It is desirable to further improve video compression performance by jointly optimizing the whole compression system.

End-to-end image or video compression

DNN based image compression methods can exploit large scale end-to-end training and highly non-linear transform, which are not used in the traditional approaches. However, it is non-trivial to directly apply these techniques to build an end-to-end learning system for video compression. First, it remains an open problem to learn how to generate and compress the motion information tailored for video compression. Video compression methods heavily rely on motion information to reduce temporal redundancy in video sequences.

A straightforward solution is to use the learning based optical flow to represent motion information. However, current learning based optical flow approaches aim at generating flow fields as accurate as possible. The precise optical flow is often not optimal for a particular video task. In addition, the data volume of optical flow increases significantly when compared with motion information in the traditional compression systems and directly applying the existing compression approaches to compress optical flow values will significantly increase the number of bits required for storing motion information. Second, it is unclear how to build a DNN based video compression system by minimizing the rate-distortion based objective for both residual and motion information. Rate-distortion optimization (RDO) aims at achieving higher quality of reconstructed frame (i.e., less distortion) when the number of bits (or bit rate) for compression is given. RDO is important for video compression performance. In order to exploit the power of end-to-end training for learning based compression system, the RDO strategy is required to optimize the whole system.

In *Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, Zhiyong Gao; „DVC: An End-to-end Deep Video Compression Framework". Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 11006-11015*, authors proposed the end-to-end deep video compression (DVC) model that jointly learns motion estimation, motion compression, and residual coding.

Such encoder is illustrated in Figure 6A. In particular, Figure 6A shows an overall structure of end-to-end trainable video compression framework. In order to compress motion information, a CNN was designated to transform the optical flow v_t to the corresponding representations m_t suitable for better compression. Specifically, an auto-encoder style network is used to compress the optical flow. The motion vectors (MV) compression network is shown in Figure 6B. The network architecture is somewhat similar to the ga/gs of Figure 4. In particular, the optical flow v_t is fed into a series of convolution operation and nonlinear transform including GDN and IGDN. The number of output channels c for convolution (deconvolution) is here exemplarily 128 except for the last deconvolution layer, which is equal to 2 in this example. The kernel size is k, e.g. k=3. Given optical flow with the size of M × N × 2, the MV encoder will generate the motion representation m_t with the size of M/16×N/16×128. Then motion representation is quantized (Q), entropy coded and sent to bitstream as $\hat{m}_t$. The MV decoder receives the quantized representation $\hat{m}_t$ and reconstruct motion information $\hat{v}_t$ using MV encoder. In general, the values for k and c may differ from the above mentioned examples as is known from the art.

Figure 6C shows a structure of the motion compensation part. Here, using previous reconstructed frame x_{t-1} and reconstructed motion information, the warping unit generates the warped frame (normally, with help of interpolation filter such as bi-linear interpolation filter). Then a separate CNN with three inputs generates the predicted picture. The architecture of the motion compensation CNN is also shown in Figure 6C.

The residual information between the original frame and the predicted frame is encoded by the residual encoder network. A highly non-linear neural network is used to transform the residuals to the corresponding latent representation. Compared with discrete cosine transform in the traditional video compression system, this approach can better exploit the power of non-linear transform and achieve higher compression efficiency.

From above overview it can be seen that CNN based architecture can be applied both for image and video compression, considering different parts of video framework including motion estimation, motion compensation and residual coding. Entropy coding is popular method used for data compression, which is widely adopted by the industry and is also applicable for feature map compression either for human perception or for computer vision tasks.

Video Coding for Machines

The Video Coding for Machines (VCM) is another computer science direction being popular nowadays. The main idea behind this approach is to transmit the coded representation of image or video information targeted to further processing by computer vision (CV) algorithms, like object segmentation, detection and recognition. In contrast to traditional image and video coding targeted to human perception the quality characteristic is the performance of computer vision task, e.g. object detection accuracy, rather than reconstructed quality.

A recent study proposed a new deployment paradigm called collaborative intelligence, whereby a deep model is split between the mobile and the cloud. Extensive experiments under various hardware configurations and wireless connectivity modes revealed that the optimal operating point in terms of energy consumption and/or computational latency involves splitting the model, usually at a point deep in the network. Today's common solutions, where the model sits fully in the cloud or fully at the mobile, were found to be rarely (if ever) optimal. The notion of collaborative intelligence has been extended to model training as well. In this case, data flows both ways: from the cloud to the mobile during back-propagation in training, and from the mobile to the cloud during forward passes in training, as well as inference.

Lossy compression of deep feature data has been studied based on HEVC intra coding, in the context of a recent deep model for object detection. It was noted the degradation of detection performance with increased compression levels and proposed

compression-augmented training to minimize this loss by producing a model that is more robust to quantization noise in feature values. However, this is still a sub-optimal solution, because the codec employed is highly complex and optimized for natural scene compression rather than deep feature compression.

The problem of deep feature compression for the collaborative intelligence has been addressed by an approach for object detection task using popular YOLOv2 network for the study of compression efficiency and recognition accuracy trade-off. Here the term deep feature has the same meaning as feature map. The word ‘deep‘ comes from the collaborative intelligence idea when the output feature map of some hidden (deep) layer is captured and transferred to the cloud to perform inference. That appears to be more efficient rather than sending compressed natural image data to the cloud and perform the object detection using reconstructed images.

The efficient compression of feature maps benefits the image and video compression and reconstruction both for human perception and for machine vision. Said about disadvantages of state-of-the art autoencoder based approach to compression are also valid for machine vision tasks.

Functional modules

Variable bitrate module

An encoder can output bitstreams at different bit rates. Therefore, in some methods, an output of an encoding network is scaled (for example, each channel is multiplied by a corresponding scaling factor that is also referred to as a target gain value), and an input of a decoding network is inversely scaled (for example, each channel is multiplied by a corresponding scaling factor reciprocal that is also referred to as a target inverse gain value), as shown in FIG. 7. The scaling factor may be preset. Different quality levels or quantization parameters correspond to different target gain values. If the output of the encoding network is scaled to a smaller value, a bitstream size may be decreased. Otherwise, the bitstream size may be increased.

Color format transform

RGB and YUV are common color spaces. Conversion between RGB and YUV may be performed according to an equation specified in standards such as CCIR 601 and BT.709.

Separate structure for luma and chroma

Some VAE-based codecs use the YUV color space as an input of an encoder and an output of a decoder, as shown in FIG. 8. A Y component indicates luma, and a UV component indicates chroma. Resolution of the UV component may be the same as or lower than that of the Y component. Typical formats include YUV4:4:4, YUV4:2:2, and YUV4:2:0. The Y component is converted into a feature map F_Y through a network, and an entropy encoding module generates a bitstream of the Y component based on the feature map F_Y. The UV component is converted into a feature map F_UV through another network, and the entropy encoding module generates a bitstream of the UV component based on the feature map F_UV. Under this structure, the feature map of the Y component and the feature map of the UV component may be independently quantized, so that bits are flexibly allocated for luma and chroma. For example, for a color-sensitive image, a feature map of a UV component may be less quantized, and a quantity of bitstream bits for a UV component may be increased, to improve reconstruction quality of the UV component and achieve better visual effect.

In some other methods, an encoder concatenates (concatenate) a Y component and a UV component and then sends to a UV component processing module (for converting image information into a feature map). In addition, a decoder concatenates a reconstructed feature map of the Y component and a reconstructed feature map of the UV component and then sends to a UV component processing module 2 (for converting a feature map into image information). In this method, a correlation between the Y component and the UV component may be used to reduce a bitstream of the UV component.

In the present specification, in the context of a neural network, a 'parameter' may be a value used in an operation process of each layer forming a neural network, and for example, may include a weight used when an input value is applied to a certain operation expression. Here, the parameter may be expressed in a matrix form. The parameter is a value set as a result of training, and may be updated through separate training data when necessary.

In the present specification, in the context of a bitstream, a 'parameter' may be a sequence of bits used to encode some property, flag, or value that defines or identifies an aspect or property of the bitstream. For example, each substream of a bitstream comprises a size parameter, which represents a sequence of bits that encode the size (e.g., the size in bytes) of the substream in which the size parameter is included.

Substream signalling

Exemplary methods and devices according to particular embodiments of the present disclosure will now be described in further detail. Embodiments of the preset disclosure may be used for an image coding scenario using. Generally, embodiments allow backwards compatibility of bitstreams encoded using subsequent standards with existing decoders and legacy decoders. Specifically, embodiments of the present invention allow decoders to recognise substreams as unknown, and to subsequently decode a size of the unknown substream and discard that substream without the need to recognise or know the type of data encoded in the substream. Thus, embodiments of the present disclosure allow for decoders, including legacy decoders, to handle bitstreams containing substreams whose data type the decoders do not recognise.

A bitstream generally comprises, or consists of, one or more substreams. For example, a bitstream may comprise a plurality of substreams. Each substream contains a payload having a particular data type. In order to signal to a decoder what the data type of a substream is, each substream comprises a marker ID. Preferably, the marker ID is formed of a sequence of bits that are leading bits in the substream. In some examples, the marker ID makes up the first 16 bits of a substream. Preferably, the marker ID in each substream uses the same, predetermined, number of bits. Using a predetermined number of bits, e.g., 16 bits, is helpful to the decoder. For example, if the marker ID is known to be n bits and is known to occupy the first n bits of the substream, this guarantees that the decoder will be able to read the marker ID by reading the first n bits of the substream.

Substreams also comprise a 'substream size', which in the specification is called the 'size parameter'. In examples, the marker ID and the substream size / size parameter make up the header of the substream. However, generally, the length of the size parameters can vary depending on the marker ID. For example, a 'picture header' usually had a 16 bit substream size parameter. In contrast, a 'tools header' has an 8 bit substream size parameter, and the 'z stream' has a 24 bit size parameter. For known data types, the decoder will recognise the data type based on the marker ID and will subsequently be able to determine (e.g., based on known syntax rules or based on a lookup table and the like) how many bits the substream size parameter occupies. For example, in the case of a picture header, the decoder would recognise the 'picture header' type from the marker ID, and would thus be able to determine that the subsequent 16 bits of the substream form the substream size ID.

However, bitstreams created using codecs according to current examples contain 'reserved' values denoting substreams that have not yet been defined.

Fig. 9 shows a bitstreams having such reserved sections. In fig. 9, each row of the table corresponds to a different substream. The reserved sections 900 in this example comprise seven substreams. In newer versions of codecs that produce a bitstream according to Fig. 9, a new set of substreams and corresponding markers may be inserted in place of the reserved sections 900. The problem therefore is that these new markers are not yet defined, and so marker IDs corresponding to the new substreams will be unknown to the decoder.

Generally, if a decoder receives a recognised substream that it does not wish to handle, or cannot handle, the decoder discard the substream. In other words, the decoder will determine the length of the substream from the substream size parameter, and will ignore the subsequent number of bits corresponding to the size identified by the size parameter. Thus, the decoder skips over the discarded substream to reach the next substream.

However, if the decoder does not know how to interpret the header code and marker ID then the decoder will not be able to handle the substream properly, i.e., it will not even be able to properly discard them. In other words, if the decoder cannot recognise the marker ID, it will not know the data type, will not know where to find the size of the substream size parameter, and thus will not be able to determine which bits the substream size parameter occupies. The problem is that, although the marker ID size is fixed, the substream size parameter generally varies. Thus, if the marker ID is unknown, the decoder will not be able to determine which bits correspond to the substream size parameter because it will not know what length the substream size parameter is.

Therefore, for newly defined or unknown marker IDs, the decoder will not be able to determine the size of the substream in order to discard it. So, the decoder will not be able to discard that substream, and so will not be able to decode any of the subsequent substreams because it is unable to determine where they start. So, rather than simply discarding the substream, the decoder will not be able to work on entirety of the bitstream. This presents a backwards compatibility problem. The present disclosure aims to solve this problem, i.e., to provide a decoder that is able to successfully discard substreams of unknown type even when the decoder cannot recognise the marker ID of the substream.

The solution of the present disclosure uses a format for the size parameter that defines its own bit-size using a subset of bits contained within the size parameter. In other words, the size parameter is able to define its own size at the same time as defining a size of the substream. Generally, a subset of bits of the size parameter are dedicated to defining a size (in bits) of the size parameter itself, and the remaining bits of the size parameter define a size of the substream. The size of the substream may be defined in some other units, e.g., bytes. This enables a decoder to determine a bit-length of the size parameter (of a substream) in dependence on a subset of bits included within the size parameter.

Specifically, embodiments of the present disclosure may use exponential Golomb binarization for coding the size parameter of the substream. In that case, all decoders (including legacy and new decoders) will be able to read both the marker ID and the size parameter of the substream, even if the marker ID is unknown to the decoder. In a case of unknown marker, the decoder will be able to discard the whole substream and process substreams with known types as it should. This is because the decoder will be able to determine, from the format of the size parameter that defines its own bit-length, how many bits the size parameter occupies and therefore the size of the substream.

Exponential Golomb binarization example

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v). These denote that the size parameter are defined according to the Exponential Golomb binarization format.

Inputs to this process are bits from the raw codestream/substream sequence. Outputs of this process are syntax element values. The following process describes how the size parameter may be parsed to determine the bit-length of the size parameter, and the size of the substream defined by the size parameter.

Syntax elements coded as ue(v) are Exp-Golomb-coded with order k equal to 0. The parsing process for these syntax elements (i.e., the parsing process of the size parameter) begins with reading the bits starting at the current location (which is the start of

the size parameter) in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows, define using pseudo code:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$codeNum = (2^{leadingZeroBits} - 1) * 2^{k} + read_bits(leadingZeroBits + k)$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

The following table illustrates the structure of the 0-th order Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits.

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 x_1 x_0	3..6
0 0 0 1 x_2 x_1 x_0	7..14
0 0 0 0 1 x_3 x_2 x_1 x_0	15..30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31..62
...	...

The "prefix" bits are those bits that are parsed for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i , with i in the range of 0 to leadingZeroBits − 1, inclusive. Each xi is equal to either 0 or 1.

The table below illustrates explicitly the assignment of bit strings (i.e., the bits of the size parameter in exp Golomb format) to codeNum values (e.g., the actual size that is intended to be encoded by the size parameter).

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
...	...

It can therefore be seen that the number of contiguous zeroes in the leading position of the size parameter define the bit-length of the code number that defines the size of the substream. For example, the bit string ‘0 0 0 1 0 0 0’ has three leading contiguous zeroes, which denotes that the 4 (3 + 1) following bits define the size value encoded by the size parameter.

Substream structure

A bitstream is comprised, as shown in figure 9. Each of them starts with marker ID, size of the substream and payloads. The size of the substream signaled by Exp-Golomb supports discarding of new substreams marked by new marker in future standards/codecs in a case of decoding by legacy decoders. The decoders can still read the marker ID, and if the decoder doesn't support the marker, the decoder will still be able to discard the following bytes that defined by the size field.

marker_id is marker ID of the substream. This is preferbal fixed.

substream_size is the rest size of the substream in bytes. This is variable in size, but in the Exp-Golomb format (and other suitable formats that would occur to the skilled person) the size (bit-length) of the substream size parameter is defined/identified by parameter itself.

alignment_zero_bit shall be equal to 0.

An additional problem is that the the Exp-Golomb format does not have to code the number of bits in a number of bits that is a multiple of 8, (i.e., an integer number of bytes). This can cause problems because decoders and encoders read and write data in units/chunks of bytes. Read/write operations cannot be performed generally at the bit level resolution.

Fig. 10a shows a substream that would be formed by an encoder for a substream of 12 bits (i.e., not a multiple or of 8, and thus not an integer number of bytes). The entire payload data would have to be shifted to the left, and indicated by 1000a. This wastes the encoder's time, and would therefore be computationally very inefficient, and a poor use of the memory resources of a decoder. It would therefore be beneficial if it could be enforced that the payload data begins at an integer number of bytes along the substream.

Fig. 10b illustrates a substream that includes a byte alignment buffer following the substream size. The bits in the byte alignment buffer are irrelevant, and zeros may be used. In this case 1000b, the size of the substream size parameter is 12 bits. Therefore, a buffer of 4 bits is appended after the substream size parameter. This means that the data can be written directly by the encoder (or read by the decoder) immediately after the end of the substream size parameter without needing to shift the position of the payload data. This is because the total number of bits forming the i) the ID marker, ii) the size parameter, and iii) the byte alignment buffer, is equal to an integer number of bytes.

FIG. 11 is a flow diagram illustrating an exemplary method for decoding an image based on a neural network architecture. This method 1100 describes the decoding method in general terms, wherein, as a first step, the decoder receives a bitstream including input signal and one or more substreams. The decoder then determines a bit-length of a size parameter defining a substream. The decoder then parse the size parameter to thereby determine a size of the substream. Advantageously, the parsing of the at least one substream, by the decoder comprises: determining a bit-length of the size parameter in dependence on a subset of bits included within the size parameter; parsing the size parameter to thereby determine a size of the substream. Because the bit-length of the size parameter is defined by the size parameter itself, the decoder need not know the data type of the substream.

Fig. 12 is a flow diagram illustrating an exemplary method for encoding. The embodiment according to FIG. 12 may be configured to provide output readily decoded by the decoding method described with reference to FIG. 11. The method 1200 of FIG. 12 will be described as being performed by a neural network system of one or more computers located in one or more locations. For example, a system configured to perform image compression, e.g., the neural network of FIG. 1 can perform the

method 1200.Fig. 17 shows an example of a multicore coder. The arithmetic decoding may be performed in parallel, for example by a multi-core decoder. In addition, only parts of the arithmetic decoding may be performed in parallel. The method of arithmetic decoding may be realized as a range coding.

The arithmetic coding of the present disclosure may be readily applied to encoding of feature maps of a neural network or in classic picture (still or video) encoding and decoding. The neural networks may be used for any purpose, in particular for encoding and decoding or pictures (still or moving), or encoding and decoding of picture-related data such as motion flow or motion vectors or other parameters. The neural network may also be used for computer vision applications such as classification of images, depth detection, segmentation map determination, object recognition of identification or the like.

The entropy decoding may be performed in parallel, for example by a multi-core decoder. In addition, only parts of the entropy decoding may be performed in parallel. Fig. 19 shows an exemplary scheme of a parallel (e.g. a multi-core) encoder 620. Each of the input data channels 610 may be encoded into an individual substream including coded bits 630-633 and trailing bits 640-643. The lengths of the substreams 650 are signaled. In parallel processing implementations, the bitstream consists of several substreams, which are concatenated in a final step. Each of the substreams needs to be finalized. This because the substreams are encoded independently of each other, so that the encoding (and thus also decoding) of one substream does not require previous encoding (or decoding) of another one or more substreams.

The input data channels may refer to channels obtained by processing some data by a neural network. For example, the input data may be feature channels such as output channels or latent representation channels of a neural network. In an exemplary implementation, the neural network is a deep neural network and/or a convolutional neural network or the like. The neural network may be trained to process pictures (still or moving). The processing may be for picture encoding and reconstruction or for computer vision such as object recognition, classification, segmentation, or the like. In general, the present disclosure is not limited to any particular kind of tasks or neural networks. Rather, the present disclosure is applicable for encoding any kind of data coming from a plurality of channels, which are to be generally understood as any sources of data. Moreover, the channels may be provided by a pre-processing of source data.

Some exemplary implementations in hardware and software

The corresponding system which may deploy the above-mentioned encoder-decoder processing chain is illustrated in Fig. 18. Fig. 18. is a schematic block diagram illustrating an example coding system, e.g. a video, image, audio, and/or other coding system (or short coding system) that may utilize techniques of this present application. Video encoder 20 (or short encoder 20) and video decoder 30 (or short decoder 30) of video coding system 10 represent examples of devices that may be configured to perform techniques in accordance with various examples described in the present application. For example, the video coding and decoding may employ neural network such which may be distributed and which may apply the above-mentioned bitstream parsing and/or bitstream generation to convey feature maps between the distributed computation nodes (two or more).

As shown in Fig. 18., the coding system 10 comprises a source device 12 configured to provide encoded picture data 21 e.g. to a destination device 14 for decoding the encoded picture data 13.

The source device 12 comprises an encoder 20, and may additionally, i.e. optionally, comprise a picture source 16, a pre-processor (or pre-processing unit) 18, e.g. a picture pre-processor 18, and a communication interface or communication unit 22.

The picture source 16 may comprise or be any kind of picture capturing device, for example a camera for capturing a real-world picture, and/or any kind of a picture generating device, for example a computer-graphics processor for generating a

computer animated picture, or any kind of other device for obtaining and/or providing a real-world picture, a computer generated picture (e.g. a screen content, a virtual reality (VR) picture) and/or any combination thereof (e.g. an augmented reality (AR) picture). The picture source may be any kind of memory or storage storing any of the aforementioned pictures.

In distinction to the pre-processor 18 and the processing performed by the pre-processing unit 18, the picture or picture data 17 may also be referred to as raw picture or raw picture data 17.

Pre-processor 18 is configured to receive the (raw) picture data 17 and to perform pre-processing on the picture data 17 to obtain a pre-processed picture 19 or pre-processed picture data 19. Pre-processing performed by the pre-processor 18 may, e.g., comprise trimming, color format conversion (e.g. from RGB to YCbCr), color correction, or de-noising. It can be understood that the pre-processing unit 18 may be optional component. It is noted that the pre-processing may also employ a neural network (such as in any of Figs. 1 to 7) which uses the presence indicator signaling.

The video encoder 20 is configured to receive the pre-processed picture data 19 and provide encoded picture data 21.

Communication interface 22 of the source device 12 may be configured to receive the encoded picture data 21 and to transmit the encoded picture data 21 (or any further processed version thereof) over communication channel 13 to another device, e.g. the destination device 14 or any other device, for storage or direct reconstruction.

The destination device 14 comprises a decoder 30 (e.g. a video decoder 30), and may additionally, i.e. optionally, comprise a communication interface or communication unit 28, a post-processor 32 (or post-processing unit 32) and a display device 34.

The communication interface 28 of the destination device 14 is configured receive the encoded picture data 21 (or any further processed version thereof), e.g. directly from the source device 12 or from any other source, e.g. a storage device, e.g. an encoded picture data storage device, and provide the encoded picture data 21 to the decoder 30.

The communication interface 22 and the communication interface 28 may be configured to transmit or receive the encoded picture data 21 or encoded data 13 via a direct communication link between the source device 12 and the destination device 14, e.g. a direct wired or wireless connection, or via any kind of network, e.g. a wired or wireless network or any combination thereof, or any kind of private and public network, or any kind of combination thereof.

The communication interface 22 may be, e.g., configured to package the encoded picture data 21 into an appropriate format, e.g. packets, and/or process the encoded picture data using any kind of transmission encoding or processing for transmission over a communication link or communication network.

The communication interface 28, forming the counterpart of the communication interface 22, may be, e.g., configured to receive the transmitted data and process the transmission data using any kind of corresponding transmission decoding or processing and/or de-packaging to obtain the encoded picture data 21.

Both, communication interface 22 and communication interface 28 may be configured as unidirectional communication interfaces as indicated by the arrow for the communication channel 13 in Fig. M-4 pointing from the source device 12 to the destination device 14, or bi-directional communication interfaces, and may be configured, e.g. to send and receive messages, e.g. to set up a connection, to acknowledge and exchange any other information related to the communication link and/or data transmission, e.g. encoded picture data transmission. The decoder 30 is configured to receive the encoded picture data 21 and provide decoded picture data 31 or a decoded picture 31.

The post-processor 32 of destination device 14 is configured to post-process the decoded picture data 31 (also called reconstructed picture data), e.g. the decoded picture 31, to obtain post-processed picture data 33, e.g. a post-processed picture 33. The post-processing performed by the post-processing unit 32 may comprise, e.g. color format conversion (e.g. from YCbCr to RGB), color correction, trimming, or re-sampling, or any other processing, e.g. for preparing the decoded picture data 31 for display, e.g. by display device 34.

The display device 34 of the destination device 14 is configured to receive the post-processed picture data 33 for displaying the picture, e.g. to a user or viewer. The display device 34 may be or comprise any kind of display for representing the reconstructed picture, e.g. an integrated or external display or monitor. The displays may, e.g. comprise liquid crystal displays (LCD), organic light emitting diodes (OLED) displays, plasma displays, projectors , micro LED displays, liquid crystal on silicon (LCoS), digital light processor (DLP) or any kind of other display.

Although Fig. 18 depicts the source device 12 and the destination device 14 as separate devices, embodiments of devices may also comprise both or both functionalities, the source device 12 or corresponding functionality and the destination device 14 or corresponding functionality. In such embodiments the source device 12 or corresponding functionality and the destination device 14 or corresponding functionality may be implemented using the same hardware and/or software or by separate hardware and/or software or any combination thereof.

As will be apparent for the skilled person based on the description, the existence and (exact) split of functionalities of the different units or functionalities within the source device 12 and/or destination device 14 as shown in Fig. 18 may vary depending on the actual device and application.

The encoder 20 (e.g. a video encoder 20) or the decoder 30 (e.g. a video decoder 30) or both encoder 20 and decoder 30 may be implemented via processing circuitry, such as one or more microprocessors, digital signal processors (DSPs), application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), discrete logic, hardware, video coding dedicated or any combinations thereof. The encoder 20 may be implemented via processing circuitry 46 to embody the various modules including the neural network or its parts. The decoder 30 may be implemented via processing circuitry 46 to embody any coding system or subsystem described herein. The processing circuitry may be configured to perform the various operations as discussed later. If the techniques are implemented partially in software, a device may store instructions for the software in a suitable, non-transitory computer-readable storage medium and may execute the instructions in hardware using one or more processors to perform the techniques of this disclosure. Either of video encoder 20 and video decoder 30 may be integrated as part of a combined encoder/decoder (CODEC) in a single device, for example, as shown in Fig. 19.

Source device 12 and destination device 14 may comprise any of a wide range of devices, including any kind of handheld or stationary devices, e.g. notebook or laptop computers, mobile phones, smart phones, tablets or tablet computers, cameras, desktop computers, set-top boxes, televisions, display devices, digital media players, video gaming consoles, video streaming devices(such as content services servers or content delivery servers), broadcast receiver device, broadcast transmitter device, or the like and may use no or any kind of operating system. In some cases, the source device 12 and the destination device 14 may be equipped for wireless communication. Thus, the source device 12 and the destination device 14 may be wireless communication devices.

In some cases, video coding system 10 illustrated in Fig. 18 is merely an example and the techniques of the present application may apply to video coding settings (e.g., video encoding or video decoding) that do not necessarily include any data communication between the encoding and decoding devices. In other examples, data is retrieved from a local memory, streamed

over a network, or the like. A video encoding device may encode and store data to memory, and/or a video decoding device may retrieve and decode data from memory. In some examples, the encoding and decoding is performed by devices that do not communicate with one another, but simply encode data to memory and/or retrieve and decode data from memory.

Fig. 20 is a schematic diagram of a video coding device 8000 according to an embodiment of the disclosure. The video coding device 8000 is suitable for implementing the disclosed embodiments as described herein. In an embodiment, the video coding device 8000 may be a decoder such as video decoder 30 of Fig. 18 or an encoder such as video encoder 20 of Fig. 18.

The video coding device 8000 comprises ingress ports 8010 (or input ports 8010) and receiver units (Rx) 8020 for receiving data; a processor, logic unit, or central processing unit (CPU) 8030 to process the data; transmitter units (Tx) 8040 and egress ports 8050 (or output ports 8050) for transmitting the data; and a memory 8060 for storing the data. The video coding device 8000 may also comprise optical-to-electrical (OE) components and electrical-to-optical (EO) components coupled to the ingress ports 8010, the receiver units 8020, the transmitter units 8040, and the egress ports 8050 for egress or ingress of optical or electrical signals.

The processor 8030 is implemented by hardware and software. The processor 8030 may be implemented as one or more CPU chips, cores (e.g., as a multi-core processor), FPGAs, ASICs, and DSPs. The processor 8030 is in communication with the ingress ports 8010, receiver units 8020, transmitter units 8040, egress ports 8050, and memory 8060. The processor 8030 comprises a neural network based codec 8070. The neural network based codec 8070 implements the disclosed embodiments described above. For instance, the neural network based codec 8070 implements, processes, prepares, or provides the various coding operations. The inclusion of the neural network based codec 8070 therefore provides a substantial improvement to the functionality of the video coding device 8000 and effects a transformation of the video coding device 8000 to a different state. Alternatively, the neural network based codec 8070 is implemented as instructions stored in the memory 8060 and executed by the processor 8030.

The memory 8060 may comprise one or more disks, tape drives, and solid-state drives and may be used as an over-flow data storage device, to store programs when such programs are selected for execution, and to store instructions and data that are read during program execution. The memory 8060 may be, for example, volatile and/or non-volatile and may be a read-only memory (ROM), random access memory (RAM), ternary content-addressable memory (TCAM), and/or static random-access memory (SRAM).

Fig. 21 is a simplified block diagram of an apparatus that may be used as either or both of the source device 12 and the destination device 14 from Fig. 18 according to an exemplary embodiment.

A processor 9002 in the apparatus 9000 can be a central processing unit. Alternatively, the processor 9002 can be any other type of device, or multiple devices, capable of manipulating or processing information now-existing or hereafter developed. Although the disclosed implementations can be practiced with a single processor as shown, e.g., the processor 9002, advantages in speed and efficiency can be achieved using more than one processor.

A memory 9004 in the apparatus 9000 can be a read only memory (ROM) device or a random access memory (RAM) device in an implementation. Any other suitable type of storage device can be used as the memory 9004. The memory 9004 can include code and data 9006 that is accessed by the processor 9002 using a bus 9012. The memory 9004 can further include an operating system 9008 and application programs 9010, the application programs 9010 including at least one program that permits the processor 9002 to perform the methods described here. For example, the application programs 9010 can include applications 1 through N, which further include a video coding application that performs the methods described here.

The apparatus 9000 can also include one or more output devices, such as a display 9018. The display 9018 may be, in one example, a touch sensitive display that combines a display with a touch sensitive element that is operable to sense touch inputs. The display 9018 can be coupled to the processor 9002 via the bus 9012.

Although depicted here as a single bus, the bus 9012 of the apparatus 9000 can be composed of multiple buses. Further, a secondary storage can be directly coupled to the other components of the apparatus 9000 or can be accessed via a network and can comprise a single integrated unit such as a memory card or multiple units such as multiple memory cards. The apparatus 9000 can thus be implemented in a wide variety of configurations.

Fig. 22 is a block diagram of a video coding system 10000 according to an embodiment of the disclosure.

A platform 10002 in the system 10000 can be could sever or local sever. Alternatively, the platform 10002 can be any other type of device, or multiple devices, capable of calculation, storing, transcoding, encryption, rendering, decoding or encoding. Although the disclosed implementations can be practiced with a single platform as shown, e.g., the platform 10002, advantages in speed and efficiency can be achieved using more than one platform.

A content delivery network (CDN) 10004 in the system 10000 can be a group of geographically distributed servers. Alternatively, the CDN 10004 can be any other type of device, or multiple devices, capable of data buffering, scheduling, dissemination or speed up the delivery of web content by bringing it closer to where users are. Although the disclosed implementations can be practiced with a single CDN as shown, e.g., the CDN 10004, advantages in speed and efficiency can be achieved using more than one CDN.

A terminal 10006 in the apparatus 10000 can be a mobile phone, computer, television, laptop, camera. Alternatively, the terminal 10006 can be any other type of device, or multiple devices, capable of displaying video or image.

CLAIMS

1. A decoding method, implemented by a decoder, the method comprising:

receiving a bitstream including encoded data of an input signal, wherein the bitstream is formed of one or more substreams each comprising a size parameter defining a size of the substream;

parsing at least one substream, the parsing comprising:

determining a bit-length of the size parameter in dependence on a subset of bits included within the size parameter;

parsing the size parameter to thereby determine a size of the substream.

2. The decoding method of claim 1, wherein each of the one or more substreams comprises an ID marker indicating a data type of the substream.

3. The decoding method of claim 2, wherein a bit-length of the ID marker is predetermined.

4. The decoding method of claim 2 or 3, wherein parsing the at least one substream further comprises determining, in dependence on the ID marker, that the data type of the substream is reserved.

5. The decoding method of any preceding claim, wherein parsing the at least one substream further comprises, subsequent to determining the size of the substream, discarding data encoded within the substream in dependence on the determined size of the substream.

6. The decoding method of any preceding claim, further comprising, prior to determining the bit-length of the size parameter:

determining that the size parameter of the substream is encoded in a size-defining format, wherein the subset of bits are configured, in the size-defining format, to define the bit-length of the size parameter.

7. The decoding method of claim 6, wherein parsing the size parameter further comprises:

determining, starting from the left-most bit of the size parameter, a number of contiguous bits that are equal to zero;

in dependence on determining the number of contiguous bits that are equal to zero, determining a code number encoded by the size parameter, wherein the code number defines the size of the substream.

8. The decoding method of claim 7, wherein the number of contiguous bits that are equal to zero defines a bit-length of the code number, and wherein the code number is defined by bits appended to the contiguous bits that are equal to zero.

9. The decoding method of claim 8, wherein the size-defining format is an Exponential-Golomb coding format.

10. The method of claim 9, wherein an order k of the Exponential-Golomb coding format is defined with order k equal to 0.

11. The decoding method of any preceding claim, wherein each of the one or more substreams comprises a byte alignment buffer defining a buffer size.

12. The decoding method of claim 11, wherein parsing the at least one substream further comprises:

determining a buffer size defined by the byte alignment buffer;

parsing a remainder of the substream by parsing data starting from a starting bit in the substream, the starting bit positioned after the bits defining the size parameter by a number of bits defined by the buffer size.

13. The decoding method of claim 12, wherein each of the one or more substreams comprises an ID marker indicating a data type encoded within the substream, and wherein a total number of bits forming i) the ID marker, ii) the size parameter, and iii) the byte alignment buffer, is equal to an integer number of bytes.

14. A device for decoding data for picture or video processing from a bitstream, the device comprising:

a receiving unit configured to:

receive a bitstream including encoded data of an input signal, wherein the bitstream is formed of one or more substreams each comprising a size parameter defining a size of the substream;

a parsing unit configured to parse at least one substream, the parsing comprising:

determining a bit-length of the size parameter in dependence on a subset of bits included within size parameter;

parsing the size parameter to thereby determine a size of the substream.

15. An encoding method, implemented by an encoder, the method comprising:

receiving an input signal;

encoding the input signal into a bitstream comprising one or more substreams each comprising a size parameter defining a size of the substream, wherein the encoding comprises:

generating, for at least one substream of the one or more substreams, the size parameter for the substream wherein a bit-length of the size parameter is defined by a subset of bits included within the size parameter.

16. The encoding method of claim 15, further comprising for each substream of the one or more substreams:

generating an ID marker indicating a data type encoded within the substream;

inserting the ID marker within the substream.

17. The encoding method of claim 15 or 16, wherein the size parameter of the substream is encoded in a size-defining format, wherein the subset of bits are configured, in the size-defining format, to define the bit-length of the size parameter.

18. The encoding method of claim 17, wherein the size parameter comprises one or more contiguous bits that are equal to zero which define a bit-length of a code number, wherein the code number defines the size of the substream and is defined by bits in the size parameter appended to the contiguous bits that are equal to zero.

19. The encoding method of claim 18, wherein the size-defining format is an Exponential-Golomb coding format.

20. The encoding method of any of claims 15 to 19, further comprising for at least one substream of the one or more substreams:

generating and inserting a byte alignment buffer, defining a buffer size, within the substream.

21. The encoding method of claim 20, wherein the byte alignment buffer is generated and inserted in dependence on determining that a total number of bits forming i) the ID marker and ii) the size parameter does not equal to an integer number of bytes.

22. The encoding method of claim 20 or 21, wherein the byte alignment buffer is inserted in the substream immediately following the size parameter.

23. The encoding method of any of claims 20 to 21, wherein each of the one or more substreams comprises an ID marker indicating a data type encoded within the substream, and wherein a total number of bits forming i) the ID marker, ii) the size parameter, and iii) the byte alignment buffer, is equal to an integer number of bytes.

24. A device for encoding data for picture or video processing to obtain a bitstream, the device comprising:
a receiving unit configured to receive an input signal; and
an encoding unit configured to:
encode the input signal into a bitstream comprising one or more substreams each comprising a size parameter defining a size of the substream;
generate, for at least one substream of the one or more substreams, the size parameter for the substream wherein a bit-length of the size parameter is defined by a subset of bits included within the size parameter.

25. A decoding apparatus comprising processing circuitry configured to execute steps of the method according to any claims 1 to 13.

26. An encoding apparatus comprising processing circuitry configured to execute steps of the method according to any claims 15 to 23.

27. A decoder comprising one or more processors and a non-transitory computer-readable storage medium coupled to the one or more processors, wherein the storage medium stores programming for execution by the one or more processors, wherein the programming, when executed by the one or more processors, configures the decoder to carry out the method according to any one of claims 1 to 13.

28. A non-transitory storage medium comprising a bitstream encoded by the method of any of claims 15 to 23.

29. A computer program stored on a non-transitory medium and including code instructions, which, when executed on one or more processor, causes the one or more processor to execute the method according to any of claims 1 to 13, or 15 to 23.

30. A system for delivering a bitstream, the system including at least one storage medium configured to store at least one bitstream generated by the encoding method described in any of claims 15 to 23.

31. A system for delivering a bitstream, the system comprising:
at least one storage medium configured to store at least one bitstream generated by the method of any one of the claims 15 to 23; and
a video streaming device configured to obtain the bitstream from one of the at least one storage medium and send the bitstream to a terminal device, wherein the video streaming device comprises a content server or a content delivery server.

32. The system according to claim 31, further comprising one or more processor configured to perform encryption processing on at least one bitstream to obtain at least one encrypted bitstream, the at least one storage medium configured to store the encrypted bitstream; or
the one or more processor configured to convert the bitstream in a first format into a bitstream in a second format, the at least one storage medium configured to store the bitstream in the second format.

33. The system according to claim 31 or 32, further comprising:

a receiver configured to receive a first operation request; wherein the one or more processor is configured to determine a target bitstream in the at least one storage medium in response to the first operation request; and

a transmitter configured to send the target bitstream to a terminal-side apparatus.

34. The system according to claim 33, wherein the one or more processor is further configured to encapsulate the bitstream to obtain a transport stream in a first format, wherein the transmitter is further configured to:

send the transport stream in the first format to a terminal-side apparatus for display; or

send the transport stream in the first format to storage space for storage.