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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
03 October 2019 (03.10.2019)

WIPO | PCT

(10) International Publication Number
WO 2019/185987 A1

(51) International Patent Classification:
G06N 3/08 (2006.01) *H04N 19/50* (2014.01)
G06T 9/00 (2006.01) *H04N 19/124* (2014.01)
H04N 19/48 (2014.01) *H04N 19/154* (2014.01)
H04L 29/08 (2006.01) *H03M 7/30* (2006.01)

(21) International Application Number:
PCT/FI2019/050238

(22) International Filing Date:
21 March 2019 (21.03.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/649,699 29 March 2018 (29.03.2018) US

(71) Applicant: **NOKIA TECHNOLOGIES OY** [FI/FI]; Karakaari 7, 02610 Espoo (FI).

(72) Inventors: **AYTEKIN, Caglar**; Riihuhdankatu 13 B 6, 33580 Tampere (FI). **CRICRI, Francesco**; Massunkuja 1 A 14, 33100 Tampere (FI).

(74) Agent: **NOKIA TECHNOLOGIES OY** et al.; Ari Aarnio, IPR Department, Karakaari 7, 02610 Espoo (FI).

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

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

(54) Title: ENTROPY-FRIENDLY NEURAL NETWORK REPRESENTATIONS AND THEIR USE IN TRAINING AND USING NEURAL NETWORKS SUCH AS AUTOENCODERS

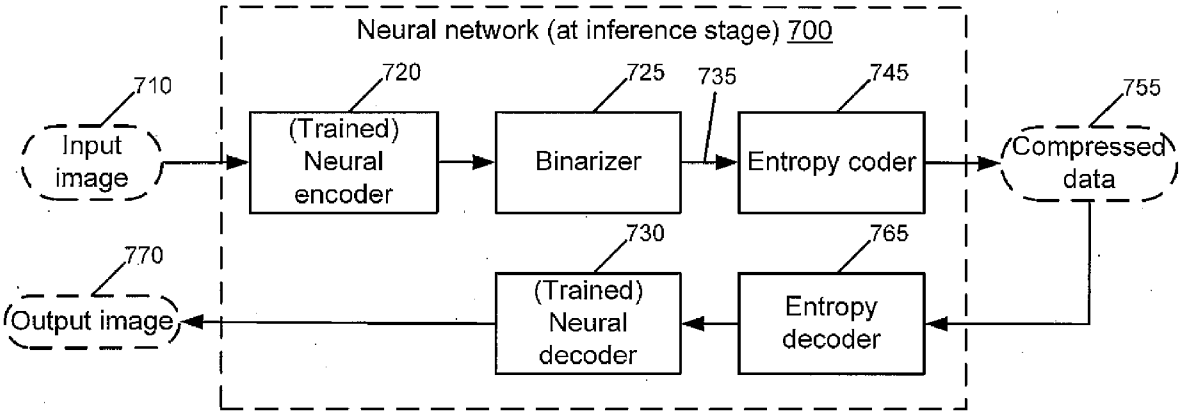

FIG. 7

(57) Abstract: This invention relates generally to neural networks and their use and, more specifically, relates to training and use of neural networks such as auto encoders for compressing and de-compressing data. An autoencoder may comprise a neural encoder part and a neural decoder part. The representations output by the encoder comprise a compressed version of the original data. The output of the decoder is a reconstruction of the input data from the representations output by the encoder.

WO 2019/185987 A1

[Continued on next page]

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

Published:

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

Entropy-Friendly Neural Network Representations and their use in Training and Using Neural Networks such as Autoencoders

TECHNICAL FIELD

[0001] This invention relates generally to neural networks and their use and, more specifically, relates to training and use of neural networks such as autoencoders.

BACKGROUND

[0002] This section is intended to provide a background or context to the invention disclosed below. The description herein may include concepts that could be pursued, but are not necessarily ones that have been previously conceived, implemented or described. Therefore, unless otherwise explicitly indicated herein, what is described in this section is not prior art to the description in this application and is not admitted to be prior art by inclusion in this section. Abbreviations that may be found in the specification and/or the drawing figures are defined below, after the main part of the detailed description section.

[0003] Neural networks are being utilized in an ever-increasing number of applications for many types of devices, such as mobile phones. Examples include image and video analysis and processing, social media data analysis, device usage data analysis, and the like.

[0004] Recently, there is an increasing trend in using artificial neural networks for compressing and de-compressing data. Examples of data types are image, video, text, audio, and the like. In particular, for image compression, one category of neural networks which are being explored is deep convolutional autoencoders. Such autoencoders contain a neural encoder part and a neural decoder part. The representations output by the encoder are the compressed version of the original data. The output of the decoder is a reconstruction of the input images from the representations output by the encoder.

[0005] In order to obtain a saving in terms of bitrate or file size, the encoded representation may be quantized or binarized. Here, we consider the binarization case, which comprises quantizing each values of the output tensor of the encoder into only two values: 0 (zero) or 1 (one). However, typically neural networks are able to output floating-point values. Therefore, a binarization process may be used for converting the encoder's output representations to a proper bit-stream.

[0006] Some methods impose a direct enforcing of the encoded representations to be a bitstream. The last layer of the neural encoder may be a sigmoid layer, so that the intermediate encoder's representations which are passed through the sigmoid layer are mapped to values within the [0, 1] range first. In order to obtain a binary value from a value in the [0, 1] range, one straightforward way is to threshold the value, for example at 0.5. However, using binary values (zeros or ones) during the training process may not be possible if the training is based on the gradient of the loss with respect of the neural network's parameters, because the binarization process may not be differentiable or the gradient of the binarization function with respect to its input is mostly zeros. This is a problem, as most current neural networks are trained based on the gradient of the loss with respect to the neural network trainable parameters (such as the connections' weights). The thresholding operation would make all gradients become zero, thus making it difficult to train the neural encoder.

[0007] Binarization can therefore be approximated via some tricks such as simulating the binarization by adding a noise to representations, probabilistic derivatives, and the like. Note that the challenge is only at training time, whereas in inference a real binarization can be employed. The binarization would not allow for training the neural network components which are before the binarization operation (in the order determined by a forward pass from input to encoder to binarization to decoder), whereas it would still be possible to train the decoder. One approach is to train both encoder and decoder using the binarization approximation, optionally followed by a final training of only the decoder while using a real binarization. Another approach is to train in two alternating phases, where during the first phase both encoder and decoder are trained using the binarization approximation, and where during the second phase the real binarization is used and only the decoder is trained.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] In the attached Drawing Figures:

[0009] FIG. 1 is a block diagram providing an overview, including a neural network and a training method, of an exemplary proposed approach for training the neural network using entropy-friendly neural network representations;

[0010] FIG. 2 is a block diagram illustrating computation of entropy-friendly loss, in an exemplary embodiment;

[0011] FIG. 3 is a table presenting memory requirements (in megabytes) for codes for a validation set of images that were not seen by the network during training, and compares exemplary embodiments with conventional techniques;

[0012] FIG. 4 is a smooth concatenated vector in an exemplary embodiment;

[0013] FIG. 5A is a block diagram of an example of a system suitable for performing exemplary embodiments;

[0014] FIG 5B is a block diagram of an example of a system suitable for performing exemplary embodiments, using multiple neural networks;

[0015] FIGS. 6A and 6B are block diagrams of examples of another system and corresponding methods suitable for performing exemplary embodiments;

[0016] FIG. 7 is a block diagram of an exemplary neural network at an inference stage, in accordance with an exemplary embodiment;

[0017] FIG. 8A is an example of an entropy coder from FIG. 7, with an option of plain coding; and

[0018] FIG. 8B is an example of an entropy coder from FIG. 7, with an option of differential coding.

DETAILED DESCRIPTION OF THE DRAWINGS

[0019] The word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments. All of the embodiments described in this Detailed Description are exemplary embodiments provided to enable persons skilled in the art to make or use the invention and not to limit the scope of the invention which is defined by the claims.

[0020] The rest of this disclosure is divided into sections, for ease of reference.

[0021] I. Introduction

[0022] As described above, binarization enforcement during training is challenging since the thresholding process is not differentiable or it would provide mostly zero gradients during the backward pass. This challenge is only at training time, whereas in inference a real binarization can be employed. However, a real binarization process may still be used for training only the decoder, or anyway all the neural network components which are processed or executed after the real binarization process (in the order specified by a forward pass).

[0023] In information theory, entropy encoding is a lossless data compression scheme that may be applied for example on a byte-stream. The key concept is that if the entropy of the symbol is low, then a high coding efficiency can be satisfied with entropy coding. A symbol may be for example a byte. A symbol with low entropy is a symbol which is easily predictable and which may occur frequently. This is usually coded by an entropy encoder by assigning to the symbol a short binary sequence. A symbol with high entropy is instead a symbol which occurs rarely and which is less easily predictable. This is usually coded by an entropy encoder by assigning to the symbol a longer binary sequence.

[0024] In the instant disclosure, we consider a specific case of a neural network, for example a convolutional autoencoder, where the encoded representations (e.g., output of encoder) are a bitstream or an approximation of a bitstream, such as values between 0 (zero) and 1 (one). Normally, i.e., when an auto-encoder is trained using only a reconstruction loss, this bitstream can be of any entropy since during training conventionally there is no constraint controlling the entropy.

[0025] In order to increase compression rates, it would be highly beneficial to train the autoencoder in such a way that the encoder's representation has as low entropy as possible. In this disclosure, we propose methods in exemplary embodiments that target such problems.

[0026] As an overview, the techniques herein relate to the problem of training a neural network so that its intermediate representations have as low entropy as possible. As an example, we consider the use case of compressing data, although the techniques are not limited to this. The neural network's intermediate representations, i.e., the output of the neural encoder, represent the information whose entropy could be minimized. Entropy is minimized by training the encoder to output representations whose symbols (bytes) have as low entropy as possible, for example so that each symbol is output as frequently as possible.

[0027] We consider neural autoencoders, which include a neural encoder and a neural decoder. The intermediate representations are the output representations obtained by the neural encoder. However, our techniques are not limited to autoencoders. For example, recurrent neural networks may be used for obtaining a representation of the input data which is more easily compressible. In general, the techniques may be applied to those cases where there is a neural network which extracts features or representations from data, where these representations are more easily compressible than the original data, where these representations are binarized, and where the binarized representations are entropy-encoded.

[0028] In exemplary embodiments, we propose a new loss in order to train the autoencoder so that the encoder's representations have low entropy and a new way to entropy code these representations. We show that the exemplary proposed methods greatly improve the coding efficiency. One exemplary embodiment includes a method that adds a constraint (e.g., an additional loss) to the usual autoencoder loss, which penalizes encoder's representations which are not smooth. Another exemplary embodiment includes a method for handling entropy between different bytes. The usual auto-encoder loss is the reconstruction loss, which may be the mean squared error between the output of the decoder and the original data which is input to the encoder. Alternatively or in addition to the mean squared loss, other common auto-encoder losses are the L1 loss and the adversarial loss provided by an auxiliary neural network.

[0029] II. Additional detail and examples

[0030] Now that an introduction has been provided, additional detail is provided. Autoencoders contain two parts: a neural encoder and a neural decoder. An autoencoder may also be referred to as an autoencoder neural network. The encoder encodes the input to a variable or fixed length code depending on the type of autoencoder. For example, with fully convolutional encoders, it is possible to encode to variable size codes, where the code size depends on the input size. Autoencoders are usually trained with reconstruction loss, such as mean squared error between the input to autoencoder and output from the autoencoder. Other losses include adversarial loss (such as the losses used in Generative Adversarial Networks), L1 (layer 1) loss, and the like. Autoencoders trained in this way do not impose any constraints on the codes.

[0031] It should be noted that the techniques herein are not limited to autoencoders. For example, recurrent neural networks may be used for obtaining a representation of the input data which is more easily compressible. In general, the techniques may be applied to those cases where there is a neural network which extracts features or representations from data, where these representations are more easily compressible, where these representations are binarized, and where the binarized representations are entropy-encoded.

[0032] Here we propose training the autoencoders by an additional loss, which we call entropy-friendly loss. Please see an exemplary overview on FIG. 1, which includes a neural network 100 and a training method 101, of an exemplary proposed approach for training the neural network to produce entropy-friendly neural network representations. In this example, the neural network is an autoencoder 100, which comprises two parts: a neural encoder 120

and a neural decoder 130. The neural encoder 120 operates on input data 110 and creates encoder's representations 125. The neural decoder operates on the encoder's representations 125 and outputs reconstructed data 135. The training method 101 comprises operations 140, 160, 170, and 180.

[0033] The neural encoder 120 has several layers 105, which may be convolutional layers, batch normalization layers, activation functions, and the like. These are shown as Layer 1 105-1 through layer N 105-N. Between nodes of each layer 105 are weights 106, which can be adjusted, e.g., during training. The last layer may be a sigmoid layer 107, so that the output tensor has elements whose values range between 0 and 1. The last layer of the encoder may also be another similar squashing function, whose output range is limited, and optionally followed by a re-scaling operation to modify the output to the range from 0 to 1. For example, a hyperbolic tangent activation function can be used, which outputs a range between -1 and 1, followed by a rescaling operation. After the sigmoid layer 107, the values go through a simulation of the binarization process (illustrated as simulate binarization 108), which may be for example the addition of a particular noise sample. This simulation of binarization (e.g., 108) may be used only at training stage, whereas at an inference stage it may be replaced by a real binarization process, such as a thresholding operation.

[0034] The neural decoder 130 may also comprise multiple layers 135, in this example Layer 1 135-1 through Layer M 135-M. The layers 135 also have weights 136 between the layers 135. These weights 136 may be adjusted, e.g., during a training phase.

[0035] The training method 101 on FIG. 1 comprises an operation 140 to compute entropy-friendly loss, which outputs the entropy-friendly loss 150. The operation 180 in the training method 101 computes a mean squared error (MSE) loss based on the input data 110 and the reconstructed data 135. The MSE loss 190 is also referred to herein as a reconstruction loss. Put differently, MSE loss is one type of reconstruction loss 190. Alternatively, any other method for determining a difference between input data 110 and reconstructed data 135 may be used to determine the reconstruction loss 190. The operation 160 in the training method 101 operates on the entropy-friendly loss 150 and the mean squared error (MSE) loss 190 to combine losses as a final loss 165. The final loss can be obtained by combining the two losses by for example a linear combination, where the weights are either predefined or adaptive to the input content. The final loss 165 is used in a training operation 170 (of the training method 101), the outputs 171, 172 of which are respectively routed back to the neural encoder 120 and the neural decoder 130. The outputs 171, 172 are

used to adjust at least some of the weights 106 and/or weights 136, respectively. As an example, the training operation 170 may differentiate the final loss with respect to the learnable parameters of the neural network, to obtain gradients. These gradients are used to compute updates for the learnable parameters (such as the weights), by using one of the available optimization routines, such as Stochastic Gradient Descent, Adam, and the like.

[0036] Regarding operation 140 to compute entropy-friendly loss, determining the entropy-friendly loss 150 may be based on calculating a difference, for example MSE, between shifted versions of the output representation of the encoder. According to one example let x be the output of the encoder, i.e., the encoder's representations 125. This may be a vector or a tensor of floating-point values ranging from 0 to 1. Referring to FIG. 2, which is a block diagram illustrating computation of entropy-friendly loss, in an exemplary embodiment, we first obtain two vectors out of x by simply making two copies of the single vector x. In FIG. 2, the vector x is labeled as 205, operation 210 performs the copy, creating copies 205-1 and 205-2. Then one copy 205-1 is concatenated from the left-hand side (see operation 220) with 0 (zero) and the other copy 205-2 is concatenated from the right-hand side (see operation 230) with 0 (zero). We call them x_{left} 225-1 and x_{right} 225-2 respectively.

[0037] Our introduced loss $l_{entropy}$ may be then determined as follows. $l_{entropy} = mse(x_{left}, x_{right})$, where mse stands for mean squared error and is computed as $mse(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$. See operation 240 in FIG. 2, which results in the entropy-friendly loss 150. $I_{entropy}$ is therefore obtained by calculating MSE between shifted versions of encoder representations 125. Minimizing $I_{entropy}$ therefore minimizes variations between consecutive elements of the encoder representation vectors. It should be noted here that we are not restricted to MSE, but can use basically any similarity loss between two vectors. One example is cosine similarity.

[0038] The description above and in particular FIGS. 1 and 2 primarily relate to a training phase. Once the neural encoder 120 and neural decoder 130 have been trained, they would then be used in an inference stage, e.g., where input data 110 would possibly be (e.g., likely be) data not seen before. The improvements made in the training phase are also expected to create improvements in the inference phase, as described in more detail below. Before proceeding with additional description of these and other improvements, it is helpful to describe how a neural network at an inference stage might be configured.

[0039] Turning to FIG. 7, this figure is a block diagram of an exemplary neural network 700 at an inference stage, in accordance with an exemplary embodiment. For this example, the input data is an input image 710, and the output data is output image 770, but this is only exemplary and other data may be used. The neural network 700 comprises a neural encoder 720, e.g., a trained neural encoder 120, and also comprises a neural decoder 730, e.g., a trained neural decoder 130. This is similar to the neural network 100 of FIG 1, but there are additional elements in the neural network 700. One such element is a binarizer 725, which produces a bitstream 735, an entropy coder 745 that produces compressed data 755, an entropy decoder 765 that provides output to the neural decoder 730, which outputs an output image 770. It is to be noted that neural network 700 may be distributed between different devices. For example neural encoder 720, binarizer 725, and entropy coder 745 may be implemented at an encoder or transmitter device. Entropy decoder 765 and neural decoder 730 may be implemented at a decoder or a receiver device. Both of these complementary parts of neural network 700 may be considered to comprise an individual neural network.

[0040] There are multiple ways to implement the entropy coder 745. Note that the design of the entropy decoder 765 would match with the design of the entropy coder 745, but the details of the entropy decoder 765 should be able to be implemented once the design of the entropy coder 745 is known. Two examples of implementations of the entropy coder 745 are illustrated in FIGS. 8A and 8B.

[0041] For instance, FIG. 8A is an example of an entropy coder 745 from FIG. 7, with an option of plain coding. The entropy coder 745-1 in FIG. 8A operates on bitstream 735 from the binarizer 725 and outputs compressed data 755. The entropy coder 745-1 comprises a convert to byte vectors operation 810 followed by entropy coding 820.

[0042] FIG. 8B is an example of an entropy coder 745 from FIG. 7, with an option of differential coding. The entropy coder 745-2 operates on bitstream 735 from the binarizer 725 and outputs compressed data 755. The entropy coder 745-2 comprises a concatenate a zero to the left of the bitstream operation 830, followed by differential coding 840, followed by a convert to byte vectors operation 810, which is followed by entropy coding 820.

[0043] The elements in FIGS. 7, 8A, and 8B, if not previously described, are referred to and described in more detail below. This description occurs because the improvements by incorporation of the entropy-friendly loss 140 in FIG. 1 also flow toward the neural network 700, and the design of neural network 700 also influences overall improvement (in

combination with techniques using the entropy-friendly loss 140) in a system using such neural networks 100/700.

[0044] This entropy-friendly loss 140 is not only trying to reduce the variation within the code, but also trying to make the rightmost and leftmost bit zero. This is very useful for continuous coding. For example, consider a case where the autoencoder can encode a limited size input which is only a portion of entire file to be coded. In this case, since we enforce the start and end of vectors to be zero, we automatically ensure a smooth transition with the next or previous code, since their beginnings and ends are also enforced to be zero. Then, a bitstream 735 corresponding to the entire file to be coded can be obtained with low entropy.

[0045] During entropy coding 820, the size of the entropy alphabet is also important. A very widely used alphabet is bytes, but one can also use bits or longer data words such as for example corresponding to 16 or 32 bits. An average size alphabet may be typically used and in the following example we adopt the byte-alphabet. For this, we convert our bitstream to bytes. This can simply be made by bit packing, where the binary code is represented by a byte for every 8-bit sequence within the bitstream. If the code length is not divisible by 8, then a simple zero-padding can be implemented. To prevent this, we may use code size (i.e., size of the encoder's output vector) that is a multiple of 8.

[0046] We can further improve the entropy coding 820 as follows. Consider the following code: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]. This can be converted to bytes as follows: [0 0 255 255].

[0047] However, one can alternatively only entropy-code the difference vector: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], which can be converted to bytes as [0 0 128 0], which has lower entropy than the previous code. The difference vector can be calculated via taking the absolute value of difference of one element with the previous element. As a simpler example, say a code is [0 1 1 1]. We add one 0 (in this example) in the beginning and make it [0 0 1 1 1], and then difference coding 840 would make it: [0 1 0 0]. While reverting this (e.g., in the entropy decoder 765), we know that in this example the artificial bit in the beginning was 0. So we can easily reconstruct the code according to the change indicators in the difference coded vector and obtain the original code: [0 1 1 1].

[0048] The sign of the difference does not matter, since in a binary vector the direction of change does not matter, i.e., if there is a change in 0 only alternative is 1 anyway, so no need to keep signs. The difference related to first element is taken with 0, therefore one can also know the first element, during decoding.

[0049] The proposed entropy-friendly loss 140 may be combined with one or more traditional autoencoder losses, such as MSE between reconstructed data and input data, adversarial loss provided by a third neural network, L1 loss, and the like. The combination process may comprise applying a linear combination (where the weights may be determined for example by minimizing the combined loss on a validation dataset or learned by another neural network) or a meta-learning neural network (which would learn to combine the losses in an improved or optimal way).

[0050] As an example, one possible combination is the following:

[0051] Loss = $l_{MSE} + l_{entropy}$.

[0052] As one set of results, FIG. 3 presents memory requirements (in megabytes) for codes for a validation set of images that were not seen by the network during training. The MSE related to all cases are similar to each other. As can be seen with the new loss techniques herein, the memory requirements are lower for both new loss with plain coding (e.g., see FIGS. 7 and 8A) and new loss with difference coding (e.g., see FIGS. 7 and 8B), relative to without new loss but with entropy coding and also without entropy coding.

[0053] III. Example system

[0054] Turning to FIG. 5A, this figure is a block diagram of an example of a system 500 suitable for performing exemplary embodiments. The system 500 comprises a training device 510, such as a server, coupled to at least one using device 560, such as a mobile phone or an Internet of Things (IoT) device. These devices are coupled through one or more networks 550, such as wireless or wired networks including possibly local area networks, wide area networks, and/or the Internet.

[0055] The training device 510 comprises one or more processors 515-1, one or more memories 520-1, and one or more network (N/W) interfaces (I/F(s)) 535-1. The one or more processors 515-1 are circuitry configured to cause the training device 510 to perform operations as described herein. These may be single-core or multiple-core general purpose processors and/or application specific integrated circuits and/or programmable logic devices and/or graphics processing units (GPUs). It should be noted that the one or more processors 515 may contain memory, such as cache memory, latches, and/or registers. The one or more memories 520-1 may be volatile or non-volatile memories. The one or more memories 520-1 comprise computer program code 525-1. The one or more network interfaces 535-1 may be wired or wireless network interfaces and are used to communicate via the network(s) 550. The training method 501 (e.g., including training method 101 and/or other methods and

techniques as described above) and the neural network 100 may be implemented in hardware, as part of the one or more processors 515-1, implemented in the computer program code 525-1 (which the one or more processors 515-1 would retrieve and execute, to cause the training device 510 to perform the operations described herein), or implemented in both hardware or computer program code. For instance, the structure of the neural network 100 could be implemented in the one or more processors 515-1, say as programmable logic, while the weights 106, 136 or other variables are in the one or more memories 520-1. The training method 501 accesses the training images 530, and trains the neural network 100, as previously described.

[0056] It is noted that the training device 510 may also be the device that uses the trained neural network 700. That is, the training device 510 may also be the using device 560, as this device is described below.

[0057] Assuming the training device 510 and the using device 560 are separate devices (as illustrated in FIG. 5A), the training device 510 can send information 540 to the using device 560. The using device 560 is similar to the training device 510 and comprises one or more processors 515-2, one or more memories 520-2 (comprising the computer program code 525-2), and one or more network (N/W) interfaces (I/F(s)) 535-2. These are similar to the one or more processors 515-1, one or more memories 520-1 (comprising the computer program code 525-1), and one or more network (N/W) interfaces (I/F(s)) 535-1 and will not be described further. The using device 560 may also have a camera (e.g., and audio) system 590, which can take and store image(s) 555, including a series of the same (along with audio) as video. In this example, the using device 560 implements the neural network 700 in one or both of the one or more processors 515-2 and memories 520-2.

[0058] In one example, the using device 560 receives information 540, such as indications of the weights 106 and 136, from the training device 510, and programs or modifies the programming of the neural network 700. The using device 560 can then take an image 555 (or access the same) and apply the encoder 120 of the neural network 700 to the image 555 to create a compressed image 555-1, e.g., as encoder's representation 125. This compressed image 555-1 could be stored, e.g., in the one or more memories 520-2. In order to view or use the image, the using device 560 could apply the decoder 130 to the compressed image 555-1 to create the decompressed image 555-2, which is a lossy version of the original image 555.

[0059] In another example, the using device 560 receives one or more compressed images 555-1, applies the decoder 130 to these images, and creates the decompressed images 555-2, which would be lossy versions of the corresponding original images 555.

[0060] IV. Combining entropy coding with block-based processing

[0061] In an exemplary embodiment, the entropy-friendly loss 140 that is used enforces the beginning and the end of each code to be zero. This is very useful if an image is divided into blocks and each block is coded separately. It enables even using different encoders or encoder parameters per block. For example, let a block be represented by: [0 0 0 0 0 1 1 1 1 0 0 0 0] and another by: [0 0 1 0 0 0]. Notice that the bit-rates are varying. In the end, since these two vectors are concatenated, and the end and start bits of both vectors are enforced to be zero, then it will give rise to a smooth concatenated vector as follows: [0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0]. FIG. 4 reproduces this vector. The encircled bits represent the end bit 410 of the first vector 405-1 and start bit 420 of the second vector 405-2. Notice that the final code is smooth and suitable for entropy coding. Below, we describe an embodiment which follows this described enabler.

[0062] The following is described with reference to FIG. 5B, which is similar to FIG. 5A, so only the differences will be described. In this example, there are multiple neural networks 100A, 100B, and 100C, used in a training phase, that are trained using various techniques, some of which are described below. There are a corresponding number of neural networks 700A, 700B, and 700C, used in an inference phase. Only three NNs 100/700 are shown, but this is merely exemplary, and there could be two or four or other numbers of NNs. As part of the using device 560, there is also a neural network selection process 595, which selects which one of the neural networks 700A, 700B, or 700C should be used, based on certain criteria.

[0063] In this embodiment, we make use of our entropy coding loss's benefits of enabling encoding with different encoders and use three different neural networks 100 with different encoding rates (i.e., the three neural networks 100 A-C have different dimensionality of the encoder's output representations). For example, the neural networks 100 A-C take a 32x32 patch (e.g., a crop of an image) and encode it to a fixed bit rate which may be different for each neural network: 64, 216 and 368 bit rates, respectively. We train each neural network 100 individually. During inference, we set a target peak signal-to-noise ratio (PSNR) for each block and encode each block by the lowest bit rate network 700 A, B, or C possible which can achieve the target PSNR. This is performed by the neural network

selection process 595. This is performed in order to use low bitrates for easier blocks and to save from the overall bits. This also allows using high bit rates for more complex patches.

[0064] We have made an experiment on 102 high quality validation images. A single neural network with fixed 216 bit rate provides 26.889 PSNR, whereas the combination of networks provides 27.53 PSNR while keeping the required memory space fixed (4.7 MB for both). It is noted that memory space may also be referred to as storage space.

[0065] An alternative approach to select (via process 595) the neural network 700 A-C to be used for a certain block is to compute the lowest rate-distortion cost, for example a Lagrangian such as Cost = rate + lambda * distortion, where the rate may be the encoder's output dimension (e.g., 64) and distortion may be derived from the PSNR obtained by the corresponding decoder. Lambda may be tuned in order to achieve a target size of the encoded data.

[0066] Also, as an additional embodiment, the neural networks 100 A-C with different encoder's output sizes may be trained in two steps. In the first step, they are pre-trained on all blocks of the training images 530. In the second step, each neural network 100 A, B, or C is trained on a different type of block, where the type may be specified by the frequency range (e.g., a block may have mostly low frequency content, another block may have high frequency content), or by any other suitable measure of the "difficulty" to encode that block such as the rate-distortion cost. One further option is to perform directly the step 2, i.e., training directly on different types of blocks. These "expert" neural networks 700 A-C may then be used at inference time based on the same evaluation metric used during training. That is, the neural network selection process 595 selects the appropriate one of the neural networks 700 A, B, or C based on the evaluation metric.

[0067] Concerning additional block-based embodiments, the input blocks (e.g., for training images 530 or input data 110) may be residual blocks, i.e., the difference between one block and the neighboring block in the same image.

[0068] Furthermore, in inference (no training), optionally, instead of concatenating the blocks' codes (before entropy encoding 820), we can concatenate the difference of one block's code with respect to the previous block's code. Only the first block's code is sent as is. When using multiple neural networks (with different encoder's output sizes), this technique may be applied to only the blocks with same encoder's output sizes.

[0069] IV. Distributed training embodiments

[0070] As additional embodiments, we propose how to use the present techniques in the context of distributed training. Distributed training includes having two or more devices (e.g., mobile phones, IoT devices, and the like) referred to as training devices performing partial trainings of a neural network, and a server which collects the results from the partial trainings and combines them for training the neural network. The results of partial trainings may be updates to the some or all of the weights 106, 136 of the neural network 100. The combination of weight updates may be performed by averaging or by any other suitable means.

[0071] Turning to FIGS. 6A and 6B, these figures are block diagrams of examples of another system 600 and corresponding methods suitable for performing exemplary embodiments. The entities in FIGS. 6A and 6B are the same, but the information being transferred is not. This is described in more detail below. In this example, there is a server 610 that communicates with each of the training devices 660. There are X training devices 660-1 through 660-X, each of which implements a corresponding neural network 100-1 through 100-X (each with a corresponding encoder 120-1 through 120-X and decoder 130-1 through 130-X). Each of the training devices 660-1 through 660-X operates its corresponding neural network (NN) 100-1 through 100-X based on corresponding (e.g., different) training images 530-1 through 530-X. In some embodiments a subset of neural network , e.g. a subset of layers, may be implemented at each of the training devices 660-1 through 660-X.

[0072] Now, we consider the case where the neural network to be trained is a network for performing image or video compression, as the example described in this disclosure. Each training device 660 may use different data for performing its partial training, such as images (e.g., training images 530) captured by a local camera. Each training device will then input the data to the encoder, obtain the encoder's representations 125, and compute the decoder's output as reconstructed data 135.

[0073] In one implementation of this embodiment, shown in FIG. 6A, both the encoder's representations 125 and the decoder's output (reconstructed data 135) are sent from the training devices 660 to the server 610, which will use those for computing the entropy-friendly loss 125 and the reconstruction loss (e.g., MSE loss 190). In order to compute the reconstruction loss, the server needs to have also the input image to the auto-encoder. This input image may be already present at the server side or it may be sent to (or accessed by) the server from the training device or from a third-party entity which provides input images to the

training device. The final loss 165 is derived by combining all entropy-friendly losses 125 and reconstruction losses 190 from all devices 660, and is (or are) sent back to the devices 660 for computing the weight update.

[0074] In terms of FIG. 6A, the devices 660 perform method 602-1. Note that the method 602-1 would be implemented in one or both of the one or more processors 515-2 and memories 520-2. Each device 660 uses the NN (neural network) 100 to determine the information of the encoder's representations 125 and the reconstructed data 135. See operation 690-1. Note that this operation may also occur during training of the NN 100, but this is not necessary. In operation 695-1, the training device 660 sends (e.g., indications of) the determined information of the encoder's representations 125 and the reconstructed data 135 toward the server 610. This is illustrated by the training device 660-1 sending the information 640-1 (comprising indications of the encoder's representations 125-1 and the reconstructed data 135-1) toward the server 610, and by the training device 660-X sending the information 640-X (comprising indications of the encoder's representations 125-X and the reconstructed data 135-X) toward the server 610.

[0075] The server 610 performs the combining method 601-1 using this information 640-1 through 640-X. The method 601-1 would be implemented in one or both of the processor(s) 515-1 and memory/memories 520-1. The server 610 in operation 615-1 of method 601 computes losses 150, 190 using received information (125, 135) to determine final loss 165. The server then distributes (operation 620-1) final loss 165 to the training devices 660. This is illustrated by information 680 (e.g., indication(s) of the determined final loss 165) being distributed from the server 610 toward the training devices 660-1 through 660-X.

[0076] The training devices 660 in block 696-1 receive the revised information 680, comprising indication(s) of the final loss 165, and in block 697-1 compute weights 106, 136 using the received final loss 165.

[0077] In another implementation, illustrated in FIG. 6B, each training device 660 computes both the partial entropy-friendly loss 150 and the partial reconstruction loss 190 and sends them to the server 610, which then combines them with the other partial losses coming from all devices 660. The combined loss is then sent back to the devices 660 for computing the weight update.

[0078] In the example of FIG. 6B, this is illustrated as follows. The devices 660 perform method 602-2. Note that the method 602-2 would be implemented in one or both of

the one or more processors 515-2 and memories 520-2. Each device 660 trains the NN (neural network) 100 to determine the entropy-friendly loss 150 and the reconstruction (e.g., MSE) loss 190. See operation 690-1. In operation 695-2, the training device 660 sends (e.g., indications of) the determined information of the entropy-friendly loss 150 and the reconstruction (e.g., MSE) loss 190 toward the server 610. This is illustrated by the training device 660-1 sending the information 650-1 (comprising indications of the entropy-friendly loss 150-1 and the reconstruction loss 190-1) toward the server 610, and by the training device 660-X sending the information 650-X (comprising indications of the entropy-friendly loss 150-X and the reconstruction loss 190-X) toward the server 610.

[0079] The server 610 performs the combining method 601-2 using this information 650-1 through 650-X. The method 601-2 would be implemented in one or both of the processor(s) 515-1 and memory/memories 520-1. The server 610 in operation 615-2 of method 601 combines received information (150, 190) to determine a final loss 165. The server then distributes (operation 620-2) revised information (the final loss 165) to training devices. This is illustrated by information 680 (e.g., indication(s) of the final loss 165) being distributed from the server 610 toward the training devices 660-1 through 660-X.

[0080] The training devices 660 in block 696-2 receive the revised information 680, comprising indication(s) of the final loss 165, and in block 697-2 compute updates of weights 106, 136 using the received final loss 165.

[0081] In both implementations, there may be two streams of data from each training device 660 to the server 610: one from the encoder's output (either the encoder's representations 125 in FIG. 6A or the entropy-friendly loss 150 in FIG. 6B) and one from the decoder's output (either the reconstructed data 135 in FIG. 6A or the reconstruction loss 190 in FIG. 6B). Optionally, the training device sends also the input data, if the server does not have it by other means.

[0082] The operations described above are operations of an exemplary method or methods, results of execution of computer program instructions embodied on a computer readable memory, functions performed by logic implemented in hardware, and/or interconnected means for performing functions in accordance with exemplary embodiments.

[0083] Without in any way limiting the scope, interpretation, or application of the claims appearing below, a technical effect of one or more of the example embodiments disclosed herein is a reduction in memory (e.g., storage) requirements for neural networks. Another technical effect of one or more of the example embodiments disclosed herein is

encoder's representations have lower entropy than with previous techniques. Another technical effect of one or more of the example embodiments disclosed herein is encoder's representations which are smoother than with previous techniques and therefore have lower entropy. Another technical effect of one or more of the example embodiments disclosed herein is improved handling entropy between different bytes.

[0084] Embodiments herein may be implemented in software (executed by one or more processors), hardware (e.g., an application specific integrated circuit), or a combination of software and hardware. In an example embodiment, the software (e.g., application logic, an instruction set) is maintained on any one of various conventional computer-readable media. In the context of this document, a "computer-readable medium" may be any media or means that can contain, store, communicate, propagate or transport the instructions for use by or in connection with an instruction execution system, apparatus, or device, such as a computer, with one example of a computer described and depicted, e.g., in FIGS. 5A, 5B, 6A, and 6B. A computer-readable medium may comprise a computer-readable storage medium (e.g., memories 520-1 and 520-2 or other device) that may be any media or means that can contain, store, and/or transport the instructions for use by or in connection with an instruction execution system, apparatus, or device, such as a computer. A computer-readable storage medium does not comprise propagating signals.

[0085] If desired, the different functions discussed herein may be performed in a different order and/or concurrently with each other. Furthermore, if desired, one or more of the above-described functions may be optional or may be combined.

[0086] Although various aspects are set out above, other aspects comprise other combinations of features from the described embodiments, and not solely the combinations described above.

[0087] It is also noted herein that while the above describes example embodiments of the invention, these descriptions should not be viewed in a limiting sense. Rather, there are several variations and modifications which may be made without departing from the scope of the present invention.

[0088] The following abbreviations that may be found in the specification and/or the drawing figures are defined as follows:

GPU	graphics processing unit
I/F	interface
IoT	Internet of things

PSNR	peak signal-to-noise ratio
MB	megabytes
MSE	mean squared error
NN	neural network
N/W	network

CLAIMS

What is claimed is:

1. A method, comprising:

performing training on a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the representations into reconstructed data, the training comprising:

computing a first loss from the encoder's representations; and

using at least the first loss to train the neural network.

2. The method of claim 1, wherein:

the training further comprises:

computing one or more second losses for the neural network;

determining a final loss based on the first loss and one or more second losses; and

using at least the first loss to train the neural network further comprises using the final loss to train the neural network.

3. The method of claim 2, wherein:

computing one or more second losses for the neural network further comprises computing a second loss between the input data and reconstructed data; and

determining a final loss based on the first loss and one or more second losses further comprises determining the final loss as a combination of the first loss and the second loss between the input data and reconstructed data.

4. The method of claim 3, wherein computing a second loss between the input data and reconstructed data further comprises computing the second loss as a mean squared error between the input data and reconstructed data.

5. The method of claim 2, wherein:
using the final loss to train the neural network further comprises using final loss to update weights for one or both of the encoder and decoder.

6. The method of any of claims 1 to 5, wherein computing a first loss from the encoder's representations further comprises:
copying a vector from the encoder's representations to create two copies;
concatenating a first of the two copies from a left-hand side of the first copy with a zero to create a left-hand vector;
concatenating a second of the two copies from a right-hand side of the second copy with a zero to create a right-hand vector; and
computing a similarity between the left-hand vector and the right-hand vector, the similarity being the first loss.

7. The method of claim 6, wherein computing a similarity between the left-hand vector and the right-hand vector further comprises computing a mean squared error between the left-hand vector and the right-hand vector.

8. The method of any of claims 1 to 7, wherein:
performing training on a neural network performs training on multiple neural networks, each neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the representations into reconstructed data;
each of the multiple neural networks has a different fixed bit rate; and
the training is performed using a plurality of same blocks of input data for each of the multiple neural networks at their individual different fixed bit rates.

9. The method of claim 8, wherein the training is performed in two steps:
in a first step, the multiple neural networks are pre-trained on all blocks of input data;
in a second step, each of the multiple neural networks is trained on a different type of block selected for an individual one of the multiple neural networks based on an evaluation metric.

10. The method of claim 9, wherein:
the training creates multiple trained neural networks; and
an inference phase using the multiple trained neural networks comprises using the evaluation metric to select which one of the multiple trained neural networks is used to encode blocks of input data.

11. The method of claim 8, wherein:
the training creates multiple trained neural networks; and
an inference phase using the multiple trained neural networks comprises setting a target for each input block and encoding each block by a lowest fixed bit rate one of the multiple trained neural networks that can achieve the target.

12. The method of claim 11, wherein the target is a peak signal-to-noise ratio.

13. The method of claim 11, wherein the target is a rate-distortion cost.

14. The method of claim 13, wherein the rate-distortion cost is cost = rate + lambda * distortion, where the rate is an encoder's output dimension, distortion is derived from a peak signal-to-noise ratio obtained by a corresponding decoder, and lambda is tuned in order to achieve a target size of encoded data.

15. The method of any of the above claims, wherein the input data comprises input blocks from a same image that are residual blocks, containing a difference between one block and a neighboring block in the same image.

16. The method of any of the above claims, wherein:
the training creates a trained neural network; and
an inference phase using the trained neural networks comprises concatenating a difference of one block's code with respect to a previous block's code, wherein only a first block's code is sent as is and the concatenating a difference is performed on all subsequent blocks.

17. The method of any of the above claims, wherein at least one of the neural networks comprises an autoencoder.

18. A method, comprising:

receiving information from a plurality of devices, each of the plurality of devices implementing a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the encoder's representations into reconstructed data, wherein the information comprises information used to create a final loss used to train the neural networks;

determining a final loss based on the information; and

distributing one or more indications of the final loss toward the plurality of devices to be used by the devices to train the neural networks.

19. The method of claim 18, wherein:

the information comprises indications of the encoder's representations and the reconstructed data; and

the determining a final loss based on the information further comprises determining a first loss based on the encoder's representations and a second loss based on the reconstructed data, and determining the final loss based on the first loss and the second loss.

20. The method of claim 18, wherein:

the information comprises indications of a first loss determined using the encoder's representations and a second loss determined using the input data and the reconstructed data; and

the determining a final loss based on the information further comprises determining the final loss based on the first loss and the second loss.

21. A method, comprising:

determining information at a device, the device implementing a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the encoder's

representations into reconstructed data, wherein the information comprises information used to create a final loss used to train the neural network;

sending indications of the information toward the server;

receiving from the server and in response to the information a final loss based on the information; and

training the neural network based at least on the final loss.

22. The method of claim 21, wherein:

the information comprises the encoder's representations and the reconstructed data and indication of the reconstructed data.

23. The method of claim 21, wherein:

the information comprises a first loss determined using the encoder's representations and a second loss determined using the input data and the reconstructed data.

24. A computer program, comprising code for performing any of the methods of claims 1 to 23, when the computer program is run on a processor.

25. The computer program according to claim 24, wherein the computer program is a computer program product comprising a computer-readable medium bearing computer program code embodied therein for use with a computer.

26. A computer program product comprising a computer-readable storage medium bearing computer program code embodied therein for use with a computer, the computer program code comprising code for performing any of the methods of claims 1 to 23.

27. An apparatus, comprising:

one or more processors; and

one or more memories including computer program code,

the one or more memories and the computer program code configured, with the one or more processors, to cause the apparatus to perform any of the methods of claims 1 to 23.

28. An apparatus, comprising:

means for performing training on a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the representations into reconstructed data, the training comprising:

means for computing a first loss from the encoder's representations; and

means for using at least the first loss to train the neural network.

29. The apparatus of claim 28, wherein:

the means for training further comprises:

means for computing one or more second losses for the neural network;

means for determining a final loss based on the first loss and one or more second losses; and

means for using at least the first loss to train the neural network further comprises using the final loss to train the neural network.

30. The apparatus of claim 29, wherein:

the means for computing one or more second losses for the neural network further comprises means for computing a second loss between the input data and reconstructed data; and

the means for determining a final loss based on the first loss and one or more second losses further comprises means for determining the final loss as a combination of the first loss and the second loss between the input data and reconstructed data.

31. The apparatus of claim 30, wherein the means for computing a second loss between the input data and reconstructed data further comprises means for computing the second loss as a mean squared error between the input data and reconstructed data.

32. The apparatus of claim 29, wherein:

the means for using the final loss to train the neural network further comprises means for using final loss to update weights for one or both of the encoder and decoder.

33. The apparatus of any of claims 28 to 32, wherein the means for computing a first loss from the encoder's representations further comprises:

means for copying a vector from the encoder's representations to create two copies;

means for concatenating a first of the two copies from a left-hand side of the first copy with a zero to create a left-hand vector;

means for concatenating a second of the two copies from a right-hand side of the second copy with a zero to create a right-hand vector; and

means for computing a similarity between the left-hand vector and the right-hand vector, the similarity being the first loss.

34. The apparatus of claim 33, wherein the means for computing a similarity between the left-hand vector and the right-hand vector further comprises means for computing a mean squared error between the left-hand vector and the right-hand vector.

35. The apparatus of any of claims 28 to 34, wherein:

means for performing training on a neural network comprises means for performing training on multiple neural networks, each neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the representations into reconstructed data;

each of the multiple neural networks has a different fixed bit rate; and

the training is performed using a plurality of same blocks of input data for each of the multiple neural networks at their individual different fixed bit rates.

36. The apparatus of claim 35, wherein the training is performed in two steps:

in a first step, the multiple neural networks are pre-trained on all blocks of input data;

in a second step, each of the multiple neural networks is trained on a different type of block selected for an individual one of the multiple neural networks based on an evaluation metric.

37. The apparatus of claim 35, wherein:
the training creates multiple trained neural networks; and
an inference phase using the multiple trained neural networks comprises means for using the evaluation metric to select which one of the multiple trained neural networks is used to encode blocks of input data.

38. The apparatus of claim 35, wherein:
the training creates multiple trained neural networks; and
an inference phase using the multiple trained neural networks comprises means for setting a target for each input block and encoding each block by a lowest fixed bit rate one of the multiple trained neural networks that can achieve the target.

39. The apparatus of claim 38, wherein the target is a peak signal-to-noise ratio.

40. The apparatus of claim 38, wherein the target is a rate-distortion cost.

41. The apparatus of claim 40, wherein the rate-distortion cost is cost = rate + lambda * distortion, where the rate is an encoder's output dimension, distortion is derived from a peak signal-to-noise ratio obtained by a corresponding decoder, and lambda is tuned in order to achieve a target size of encoded data.

42. The apparatus of any of claims 28 to 41, wherein the input data comprises input blocks from a same image that are residual blocks, containing a difference between one block and a neighboring block in the same image.

43. The apparatus of any of claims 28 to 42, wherein:
the training creates a trained neural network; and
an inference phase using the trained neural networks comprises means for concatenating a difference of one block's code with respect to a previous

block's code, wherein only a first block's code is sent as is and the concatenating a difference is performed on all subsequent blocks.

44. The apparatus of any of claims 28 to 43, wherein at least one of the neural networks comprises an autoencoder.

45. An apparatus, comprising:

means for receiving information from a plurality of devices, each of the plurality of devices implementing a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the encoder's representations into reconstructed data, wherein the information comprises information used to create a final loss used to train the neural networks;

means for determining a final loss based on the information; and

means for distributing one or more indications of the final loss toward the plurality of devices to be used by the devices to train the neural networks.

46. The apparatus of claim 45, wherein:

the information comprises indications of the encoder's representations and the reconstructed data; and

the means for determining a final loss based on the information further comprises means for determining a first loss based on the encoder's representations and a second loss based on the reconstructed data, and means for determining the final loss based on the first loss and the second loss.

47. The apparatus of claim 45, wherein:

the information comprises indications of a first loss determined using the encoder's representations and a second loss determined using the input data and the reconstructed data; and

the means for determining a final loss based on the information further comprises means for determining the final loss based on the first loss and the second loss.

48. An apparatus, comprising:

means for determining information at a device, the device implementing a neural network comprising an encoder which extracts representations from input data, wherein a decoder in the neural network is configured to decode the encoder's representations into reconstructed data, wherein the information comprises information used to create a final loss used to train the neural network;

means for sending indications of the information toward the server;

means for receiving from the server and in response to the information a final loss based on the information; and

means for training the neural network based at least on the final loss.

49. The apparatus of claim 48, wherein:

the information comprises the encoder's representations and the reconstructed data and indication of the reconstructed data.

50. The apparatus of claim 48, wherein:

the information comprises a first loss determined using the encoder's representations and a second loss determined using the input data and the reconstructed data.

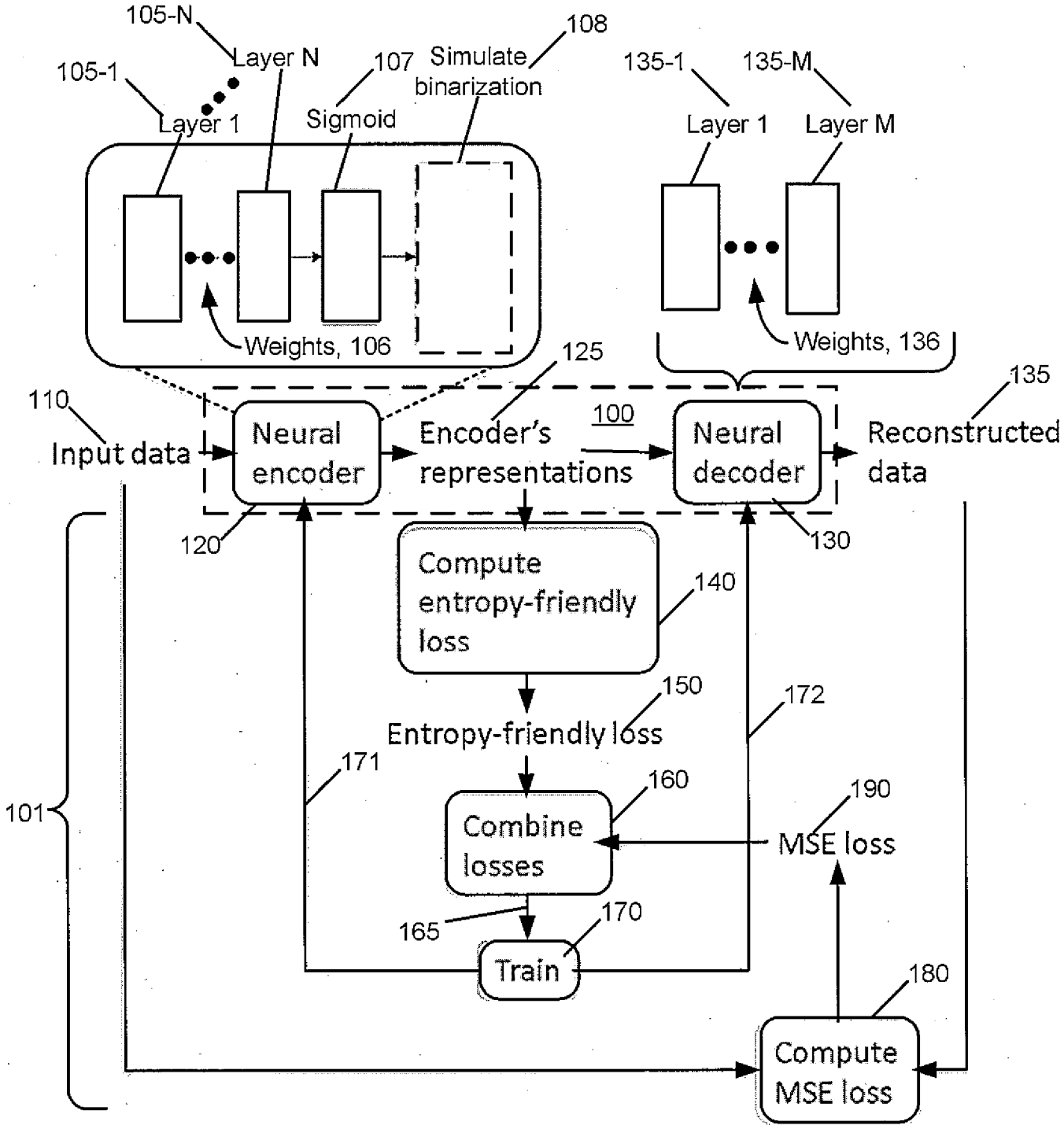
105-N
Layer N
105-1
Layer 1
107
Sigmoid
108
Simulate binarization
Weights, 106
135-1
Layer 1
135-M
Layer M
Weights, 136
125
110
Input data
Neural encoder
Encoder's representations
100
Neural decoder
135
Reconstructed data
120
130
Compute entropy-friendly loss
140
150
Entropy-friendly loss
172
171
101
160
Combine losses
190
MSE loss
165
170
Train
180
Compute MSE loss

FIG. 1

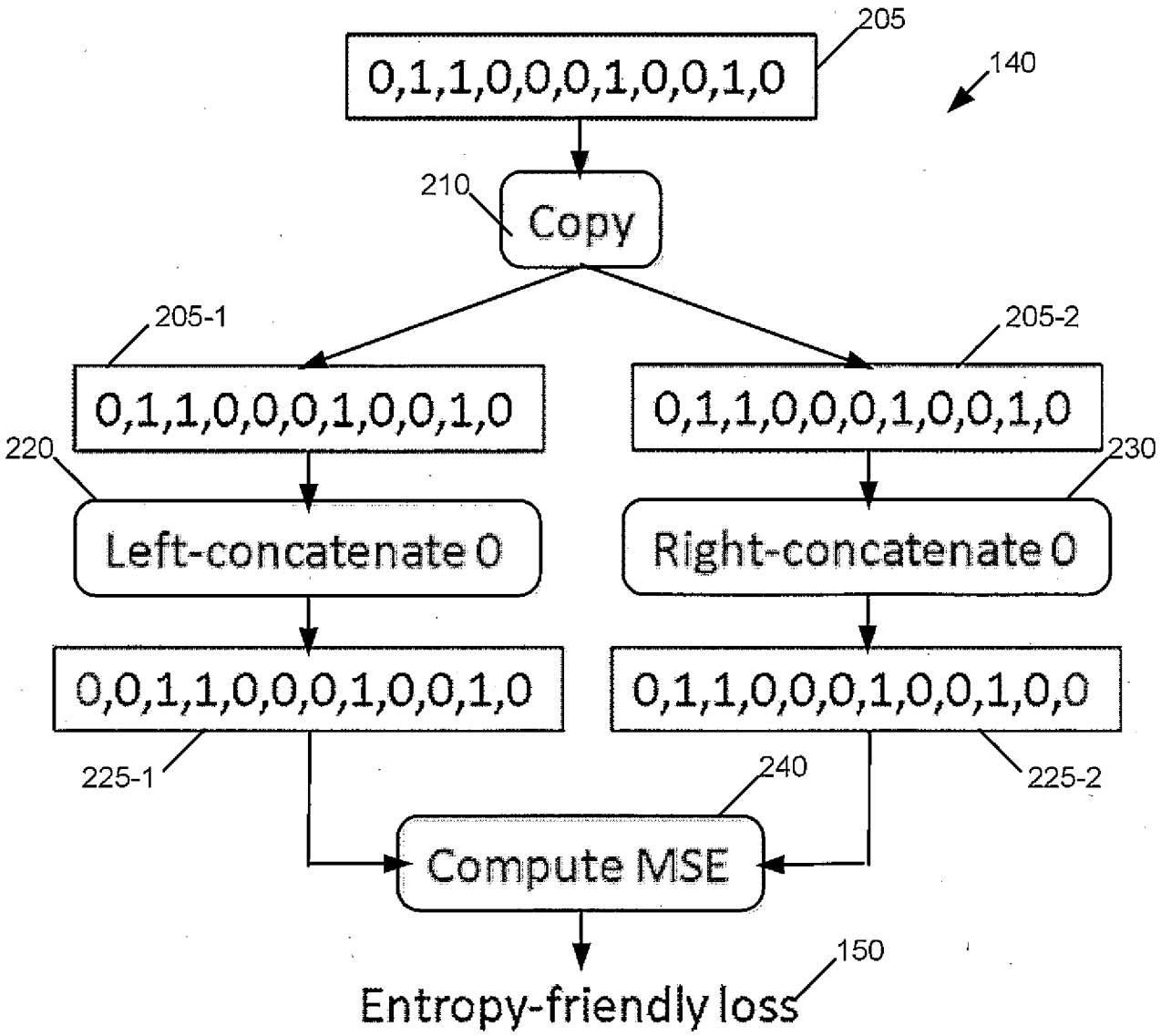

FIG. 2

Coding technique		Memory requirements (MB)
Without Entropy Coding		7.8
With Entropy Coding	Without new loss	7.3
	With new loss + plain coding	5.1
	With new loss + difference coding	4.8

FIG. 3

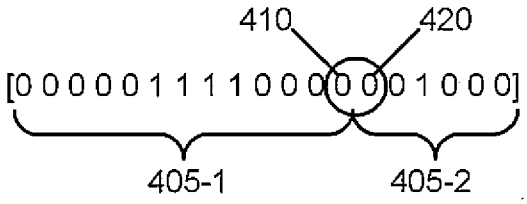

FIG. 4

175

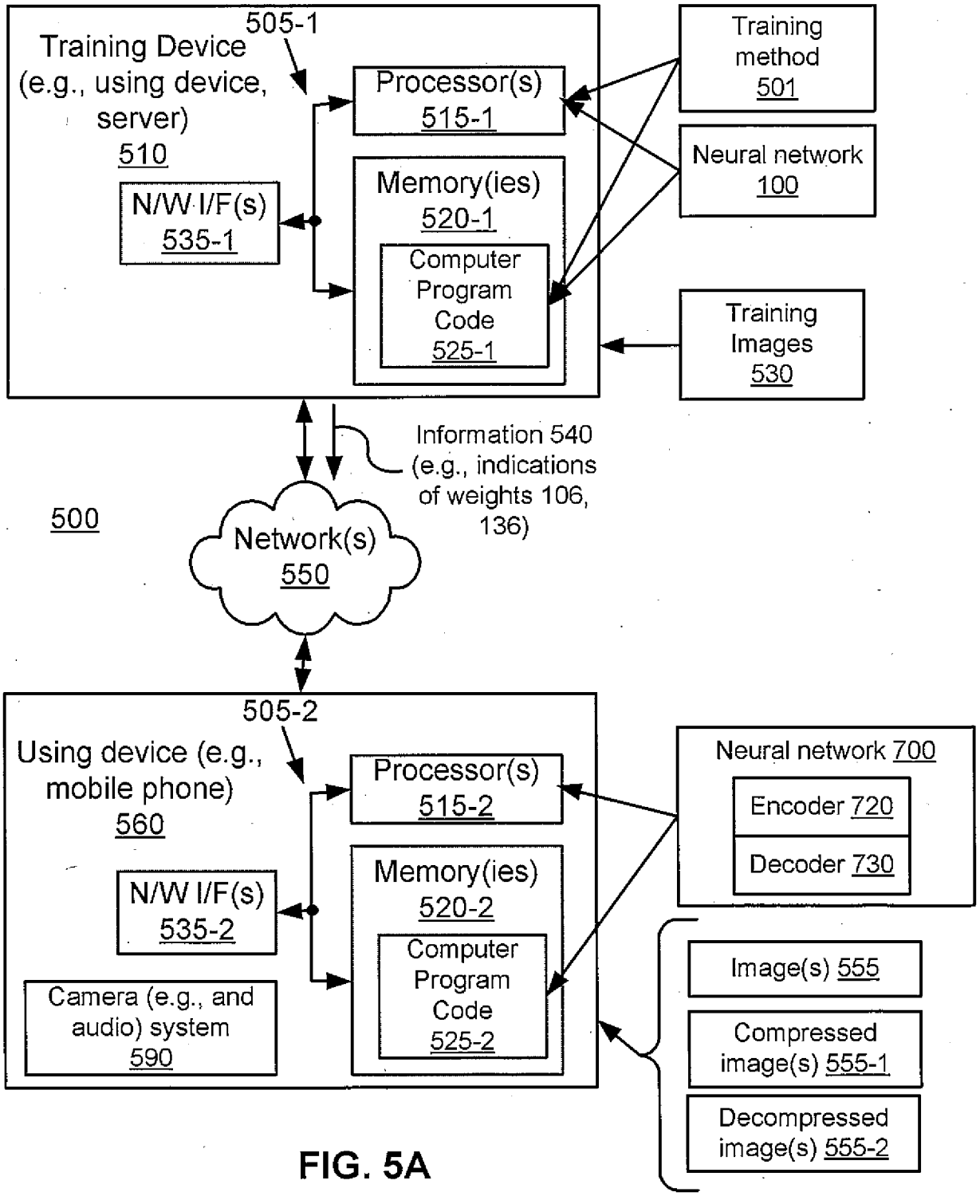

FIG. 5A

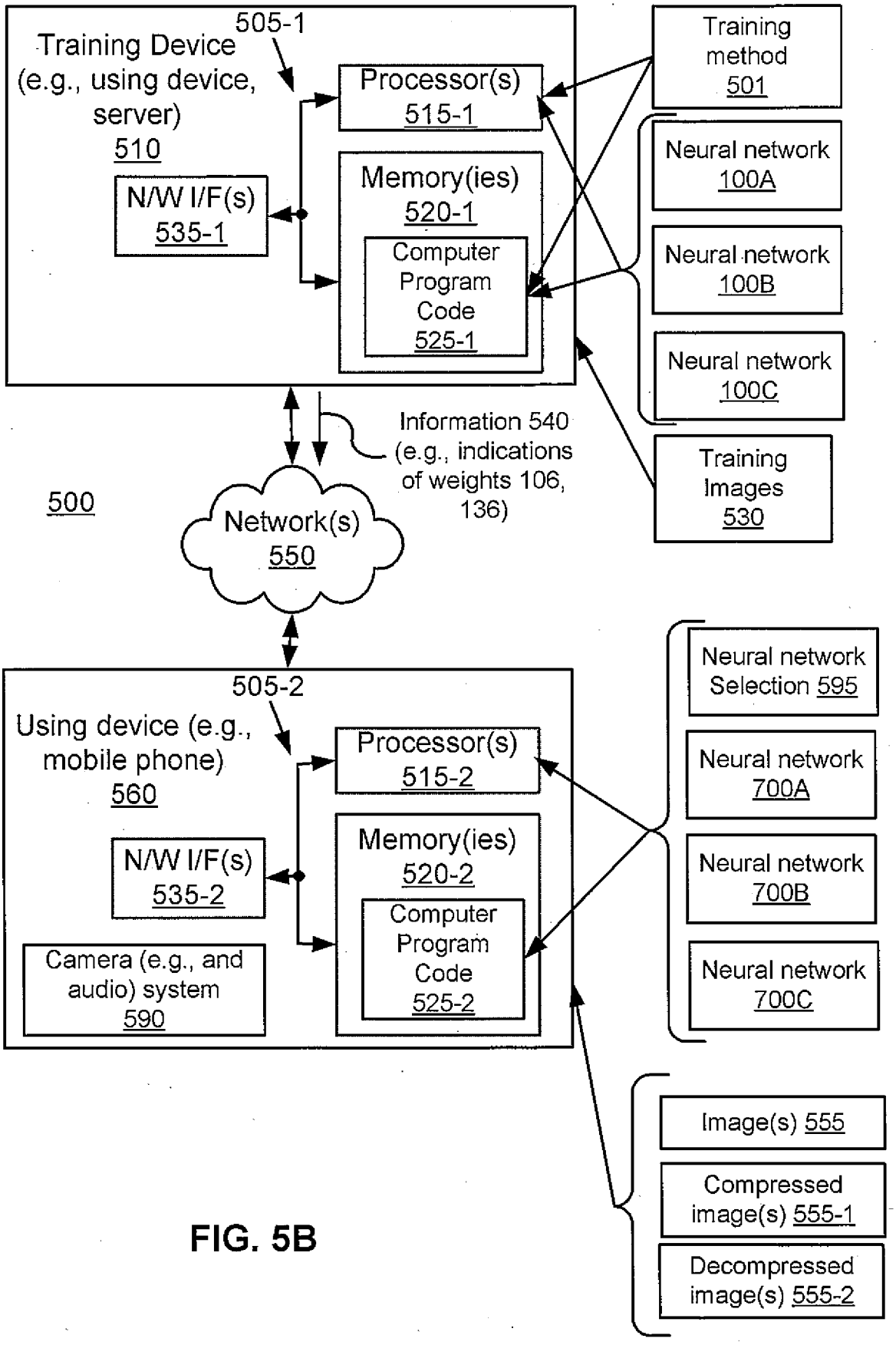
505-1
Training Device (e.g., using device, server) 510
Processor(s) 515-1
Memory(ies) 520-1
Computer Program Code 525-1
N/W I/F(s) 535-1
Training method 501
Neural network 100A
Neural network 100B
Neural network 100C
Training Images 530
Information 540 (e.g., indications of weights 106, 136)
500
Network(s) 550
505-2
Using device (e.g., mobile phone) 560
Processor(s) 515-2
Memory(ies) 520-2
Computer Program Code 525-2
N/W I/F(s) 535-2
Camera (e.g., and audio) system 590
Neural network Selection 595
Neural network 700A
Neural network 700B
Neural network 700C
Image(s) 555
Compressed image(s) 555-1
Decompressed image(s) 555-2

FIG. 5B

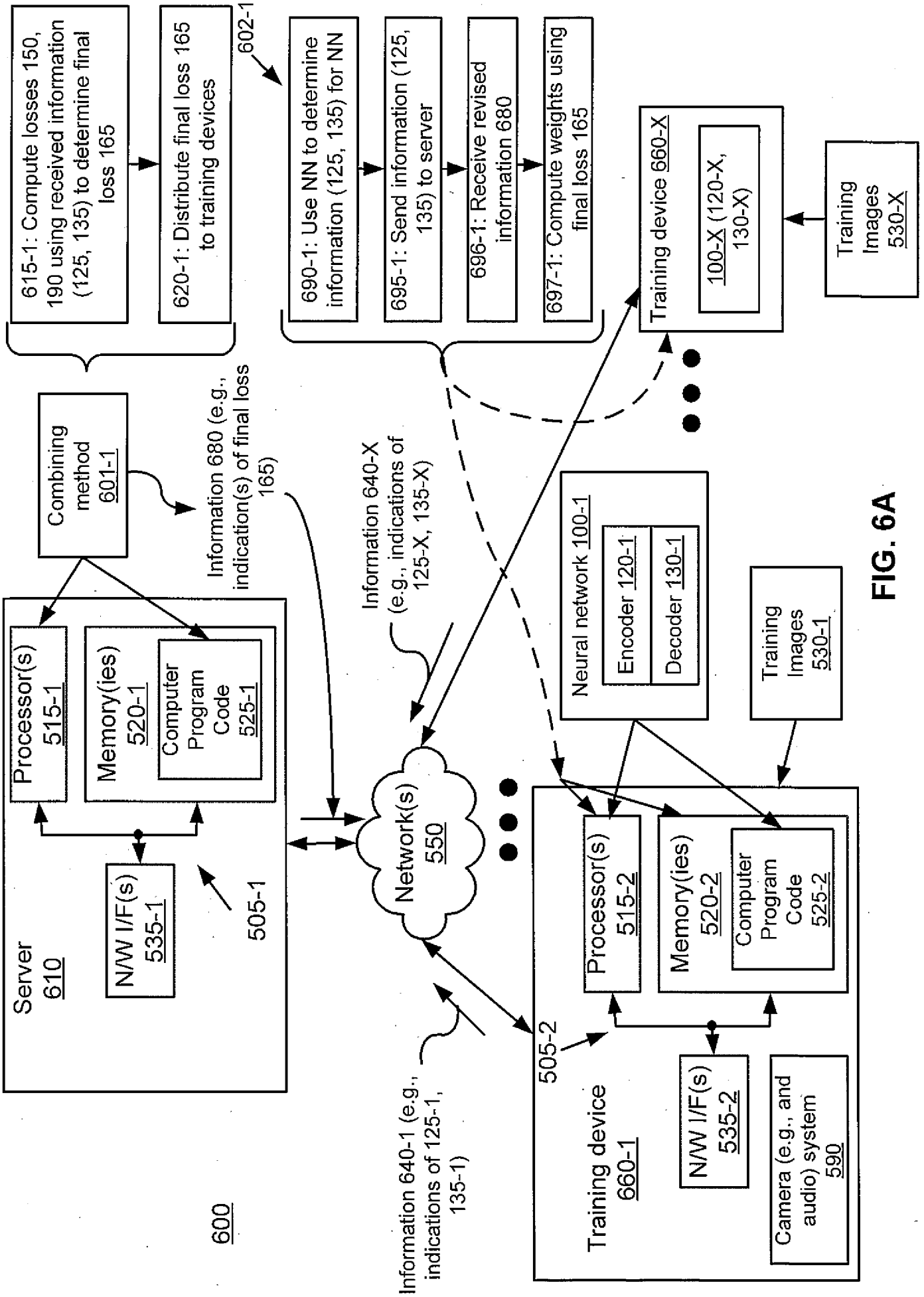
600
Server 610
Processor(s) 515-1
Memory(ies) 520-1
Computer Program Code 525-1
N/W I/F(s) 535-1
505-1
Combining method 601-1
Information 680 (e.g., indication(s) of final loss 165)
615-1: Compute losses 150, 190 using received information (125, 135) to determine final loss 165
620-1: Distribute final loss 165 to training devices
602-1
690-1: Use NN to determine information (125, 135) for NN
695-1: Send information (125, 135) to server
696-1: Receive revised information 680
697-1: Compute weights using final loss 165
Training device 660-X
100-X (120-X, 130-X)
Training Images 530-X
Information 640-X (e.g., indications of 125-X, 135-X)
Network(s) 550
Neural network 100-1
Encoder 120-1
Decoder 130-1
Training Images 530-1
Information 640-1 (e.g., indications of 125-1, 135-1)
Training device 660-1
505-2
Processor(s) 515-2
Memory(ies) 520-2
Computer Program Code 525-2
N/W I/F(s) 535-2
Camera (e.g., and audio) system 590

FIG. 6A

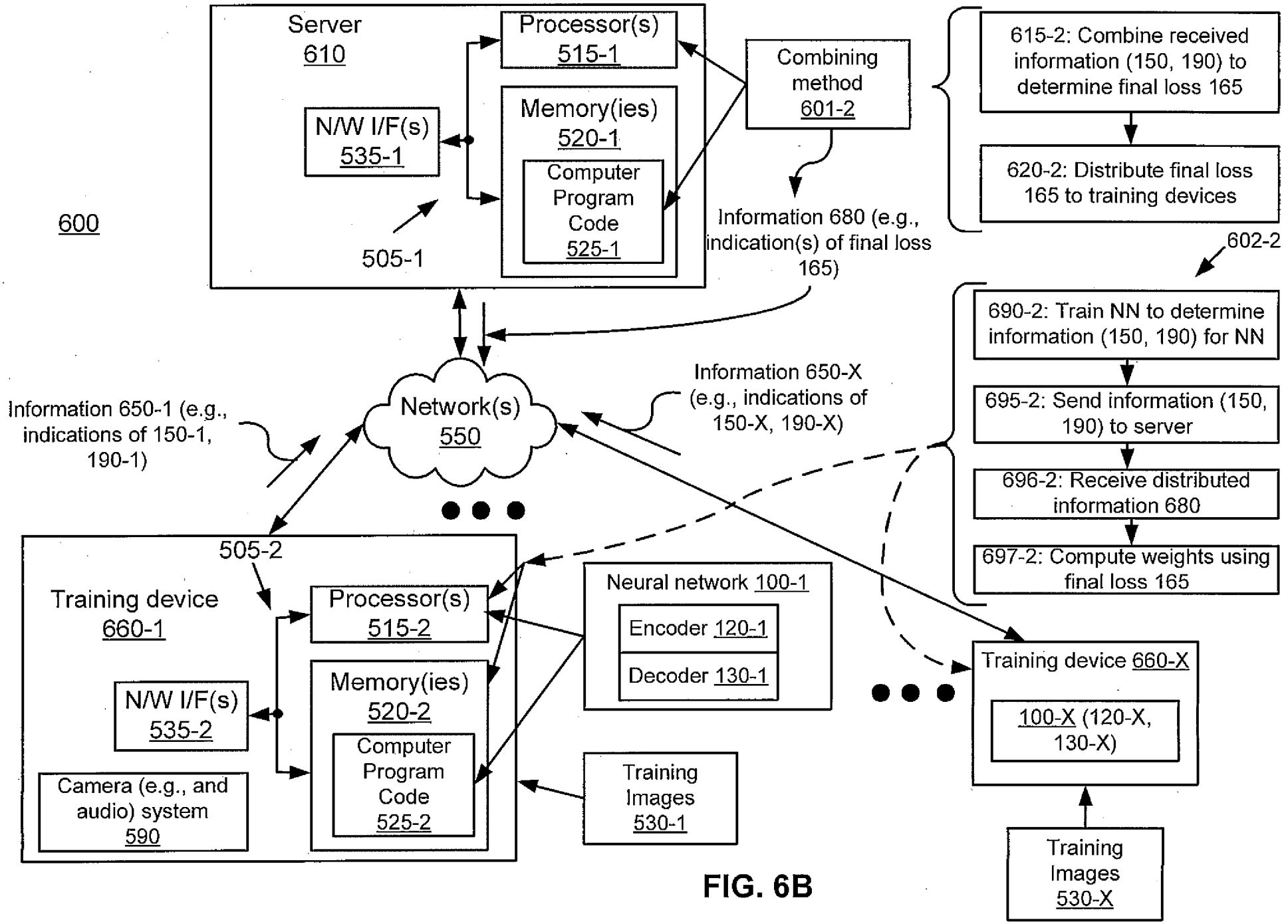
600
Server
610
Processor(s)
515-1
Memory(ies)
520-1
Computer Program Code
525-1
N/W I/F(s)
535-1
505-1
Combining method
601-2
Information 680 (e.g., indication(s) of final loss 165)
615-2: Combine received information (150, 190) to determine final loss 165
620-2: Distribute final loss 165 to training devices
602-2
690-2: Train NN to determine information (150, 190) for NN
695-2: Send information (150, 190) to server
696-2: Receive distributed information 680
697-2: Compute weights using final loss 165
Network(s)
550
Information 650-1 (e.g., indications of 150-1, 190-1)
Information 650-X (e.g., indications of 150-X, 190-X)
505-2
Training device
660-1
Processor(s)
515-2
Memory(ies)
520-2
Computer Program Code
525-2
N/W I/F(s)
535-2
Camera (e.g., and audio) system
590
Neural network 100-1
Encoder 120-1
Decoder 130-1
Training Images
530-1
Training device 660-X
100-X (120-X, 130-X)
Training Images
530-X

FIG. 6B

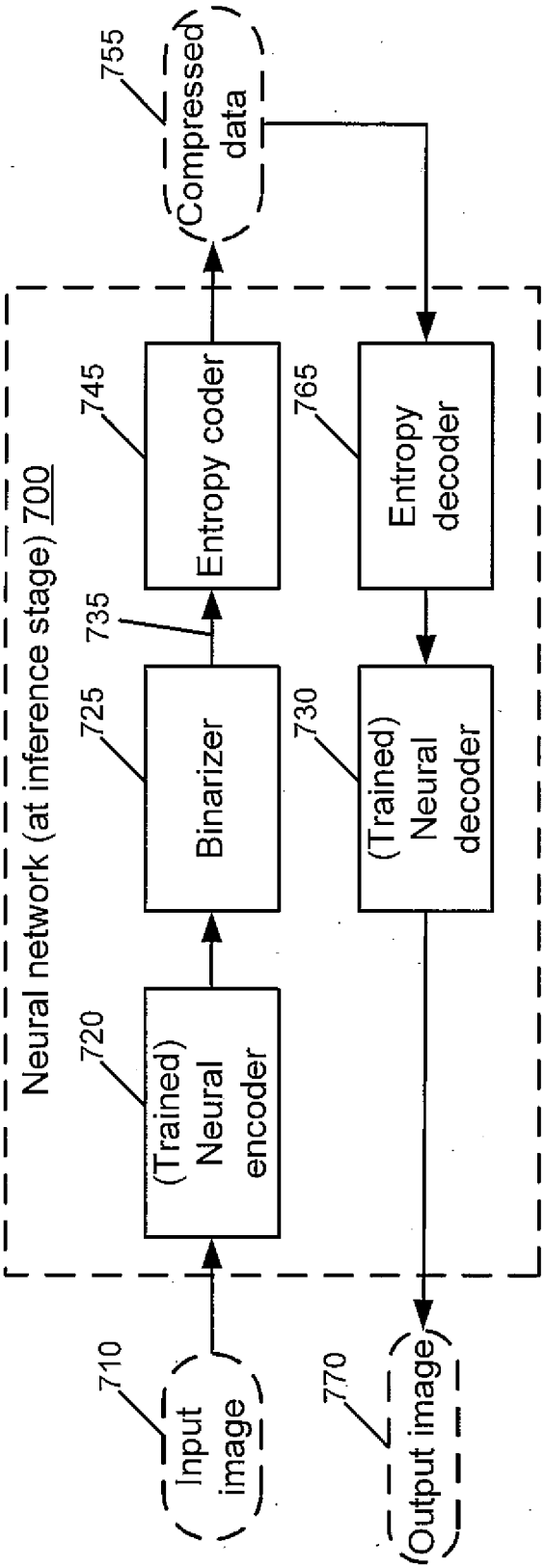
Neural network (at inference stage) 700
710
Input image
720
(Trained) Neural encoder
725
Binarizer
730
(Trained) Neural decoder
735
745
Entropy coder
755
Compressed data
765
Entropy decoder
770
Output image

FIG. 7

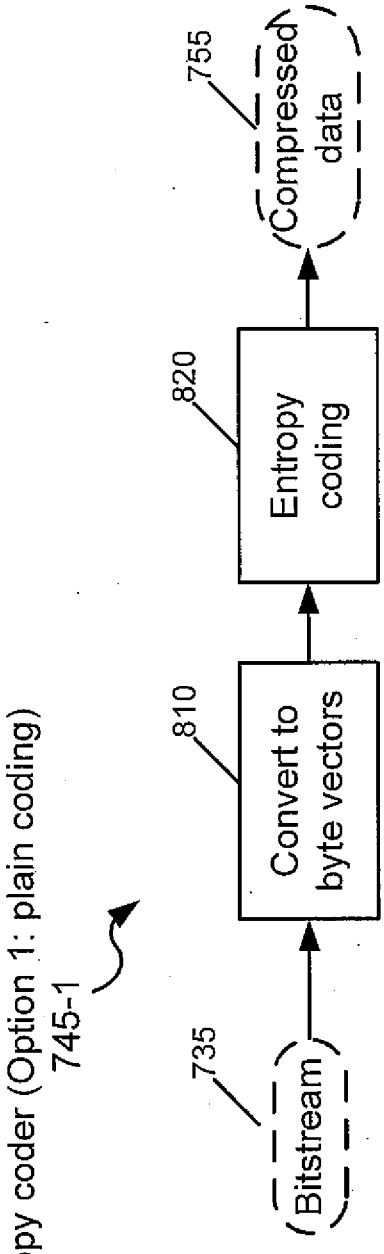
Entropy coder (Option 1: plain coding)
745-1
735
Bitstream
810
Convert to byte vectors
820
Entropy coding
755
Compressed data

FIG. 8A

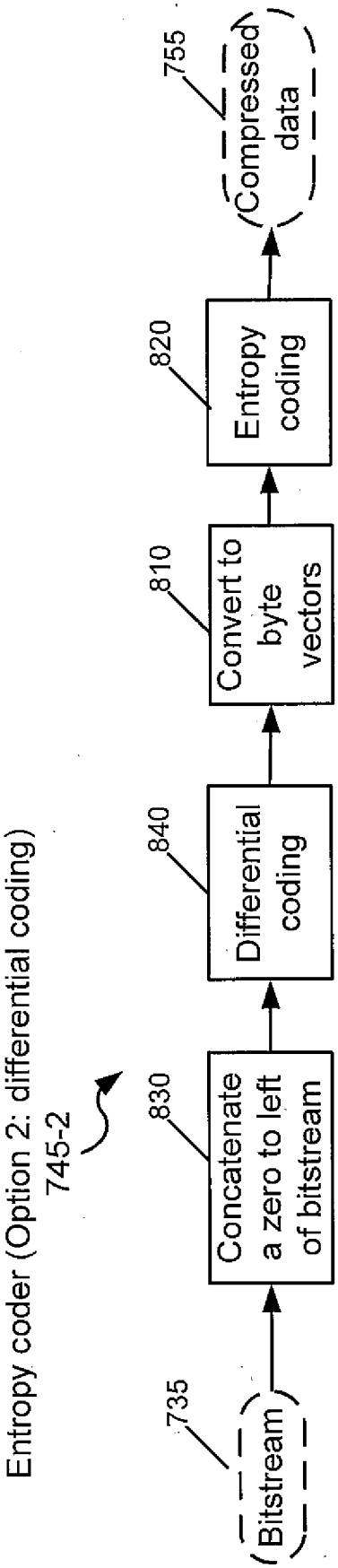
Entropy coder (Option 2: differential coding)
745-2
735
Bitstream
830
Concatenate a zero to left of bitstream
840
Differential coding
810
Convert to byte vectors
820
Entropy coding
755
Compressed data

FIG. 8B

INTERNATIONAL SEARCH REPORT

International application No.
PCT/FI2019/050238

A. **CLASSIFICATION OF SUBJECT MATTER**

See extra sheet

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC: G06N, G06T, H04N, H04L, H03M

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

FI, SE, NO, DK

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

EPODOC, EPO-Internal full-text databases, Full-text translation databases from Asian languages, WPIAP, COMPDX, INSPEC, TDB, NPL, XP3GPP, XPAIP, XPESP, XPETSI, XPI3E, XPIEE, XPIETF, XPIOP, XPIPCOM, XPJPEG, XPMISC, XPOAC, XPRD, XPTK, Internet: Google Search/Images/Patents

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	THEIS, L. et al. Lossy Image Compression with Compressive Autoencoders. In: arXiv.org, Cornell University Library [online], 2017-03-01, pages 1-19, [retrieved on 2019-03-19]. Retrieved from <https://arxiv.org/abs/1703.00395v1> introduction to section 2, section 2.3, 3.1-3.3; Fig. 4	1-17, 24-44
X	BALLE, J. et al. End-to-end Optimized Image Compression. In: arXiv.org, Cornell University Library [online], 2017-03-03, pages 1-27, [retrieved on 2019-06-04]. Retrieved from <https://arxiv.org/abs/1611.01704v3> sections 2-3	1-17, 24-44
X	DUMAS, T. et al. Autoencoder based image compression: can the learning be quantization independent?. In: arXiv.org Cornell University Library [online], 2018-02-23, pages 1-5, [retrieved on 2019-06-04]. Retrieved from <https://arxiv.org/abs/1802.09371v1>, section 2.2	1-17, 24-44

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

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 June 2019 (10.06.2019)	17 June 2019 (17.06.2019)

Name and mailing address of the ISA/FI
Finnish Patent and Registration Office
FI-00091 PRH, FINLAND
Facsimile No. +358 29 509 5328

Authorized officer
Janne Viljas
Telephone No. +358 29 509 5000

INTERNATIONAL SEARCH REPORT

International application No.
PCT/FI2019/050238

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SHIRAKAWA, S. et al. Extraction of easily interpretable representation using five-layered autoencoder. In: 2017 International Conference on Advanced Informatics, Concepts, Theory, and Applications (ICAICTA). IEEE Xplore [online], 2017-11-02, pages 1-4, [retrieved on 2019-03-20]. Retrieved from <https://ieeexplore.ieee.org/document/8090988>, <DOI:10.1109/ICAICTA.2017.8090988> sections III-IV	1-17, 24-44
X	MAKKIE, M. et al. Fast and Scalable Distributed Deep Convolutional Autoencoder for fMRI Big Data Analytics. In: arXiv.org, Cornell University Library [online], 2018-03-04, pages 1-10, [retrieved on 2019-06-05]. Retrieved from <https://arxiv.org/abs/1710.08961v3> section 5.3; Figs. 1-3	18-23, 45-50

INTERNATIONAL SEARCH REPORT

International application No.
PCT/FI2019/050238

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

See Extra Sheet.

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☒ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/FI2019/050238

CLASSIFICATION OF SUBJECT MATTER

IPC
G06N 3/08 (2006.01)
G06T 9/00 (2006.01)
H04N 19/48 (2014.01)
H04L 29/08 (2006.01)
H04N 19/50 (2014.01)
H04N 19/124 (2014.01)
H04N 19/154 (2014.01)
H03M 7/30 (2006.01)

Extra Sheet

The application lacks unity within the meaning of Rule 13.1 PCT. According to Rule 13.1 PCT, an application shall relate to one invention only or to a group of inventions so linked as to form a single general inventive concept. Further, according to Rule 13.2 PCT, the requirement of unity of invention is fulfilled only when there is a technical relationship among the claimed inventions involving one or more of the same or corresponding special technical features. The expression “special technical features” means those technical features that define a contribution which each of the claimed inventions, considered as a whole, makes over the prior art.

Claims 18-23, 45-50 lack unity with respect to claims 1-17 and 24-44 *a priori*, because the only features shared by these groups of claims relate to training a neural network comprising an encoder and a decoder, which is well known (and non-technical, as such) and therefore cannot form a special technical feature shared by the groups. Moreover, since independent claims 1, 24 and 26-28 were found to lack novelty in view of the prior art (e.g., document D1), some of the dependent claims lack unity with each other *a posteriori*.

The International Searching Authority therefore considers that this international application contains at least the following inventions or groups of inventions, which are not so linked as to form a single general inventive concept under Rule 13.1 PCT and Rule 13.2 PCT:

Invention 1: Claims 1-7, 17, 24-34, 44 (training an encoder-decoder neural net by calculating a loss from the encoder's representations and, in claims 6-7, 33-34, using a particular loss function to reduce the entropy of the encoder's representations)

Invention 2: Claims 8-14, 35-41 (training multiple neural networks that can be selected to optimize the encoding result for different kinds of blocks of input data)

Invention 3: Claims 15-16, 42-43 (input blocks are residual blocks containing differences between neighboring blocks, and encoded blocks are sent by concatenating differences of successive blocks in a differential-coding manner, which appear to relate to improving efficiency and/or robustness of coding)

Invention 4: Claims 18-23, 45-50 (using a particular distributed training scheme based on distributing losses)