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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 September 2020 (24.09.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/190772 A1

(51) International Patent Classification:
G06N 20/00 (2019.01)

(21) International Application Number:
PCT/US2020/022753

(22) International Filing Date:
13 March 2020 (13.03.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/819,463 15 March 2019 (15.03.2019) US

(71) Applicant: **FUTUREWEI TECHNOLOGIES, INC.** [US/US]; 5700 Tennyson Parkway, Suite 600, Plano, Texas 75024 (US).

(72) Inventors: **WANG, Wei**; 5700 Tennyson Parkway, Suite 600, Plano, Texas 75024 (US). **JIANG, Wei**; 5700 Tennyson Parkway, Suite 600, Plano, Texas 75024 (US). **AUYEUNG, Cheung**; 5700 Tennyson Parkway, Suite 600, Plano, Texas 75024 (US). **CHEN, Jianle**; 6168 Seafaring Way, San Diego, California 92130 (US). **CHUNG, Yu Ting**; 5700 Tennyson Parkway, Suite 600, Plano, Texas 75024 (US). **ZHU, Jiafeng**; 860 Castlewood Place, Pleasanton, California 94566 (US).

(74) Agent: **SCHEER, Bradley W.** et al.; Schwegman, Lundberg & Woessner, P.A., P. O. Box 2938, Minneapolis, Minnesota 55402 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, 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, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: NEURAL NETWORK MODEL COMPRESSION AND OPTIMIZATION

(57) Abstract: Apparatus and methods for compressing a deep convolutional neural network (CNN) compress the CNN feature map and weight tensors with relatively high inference speed to optimize a rate-distortion-speed (RDS) objective function. The method reorders a weight tensor into blocks compatible with a matrix multiplication operation. The reordered weight tensor is then quantized to provide a quantized reordered weight tensor. An input feature map is multiplied by the quantized reordered weight tensor to provide an output feature map. The output feature map and the weight tensor are compressed for transmission to other devices to allow the CNN to be reconstituted on the other devices.

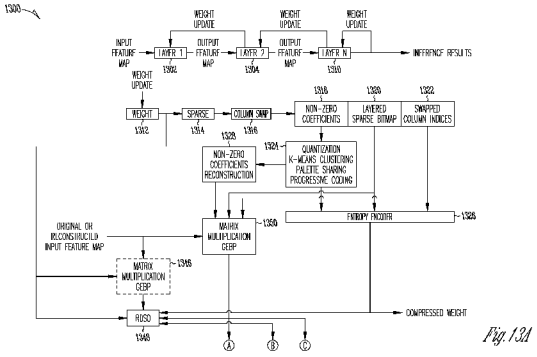

Fig. 13A

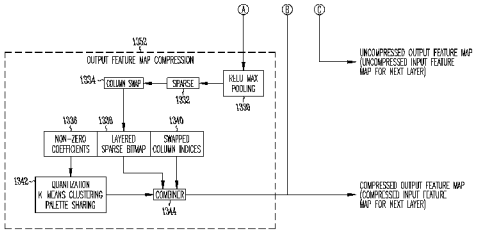

Fig. 13B

[Continued on next page]

WO 2020/190772 A1

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

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

NEURAL NETWORK MODEL COMPRESSION AND OPTIMIZATION

CLAIM FOR PRIORITY

[0001] This application claims the benefit of priority to U.S. Provisional App. Serial No. 62/819,463, filed March 15, 2019, the contents of which are hereby incorporated in its entirety.

TECHNICAL FIELD

[0002] The disclosure generally relates to generation of a compressed neural network (NN), and in particular, to a compressed NN with increased inference speed and reduced rate distortion.

BACKGROUND

[0003] Machine learning describes a wide range of algorithms by which a computer can learn to solve a problem without being explicitly programmed. One class of machine learning algorithm is artificial neural networks. An artificial neural network comprises a set of interconnected nodes. One or more input nodes receive external input data. The input nodes apply an activation function to the input and may output the result to one or more other nodes (referred to as "hidden nodes"). The hidden nodes receive input from one or more previous nodes (i.e., the input nodes or another hidden node), applying different weighting factors to each input. The hidden nodes then apply an activation function in much the same way as the input nodes. The output is then passed on to additional nodes, which process it as input. This process continues until the original input has propagated through the artificial neural network and reaches one or more output nodes. An output node applies an activation function in the same manner as other nodes, but rather than passing its output to another node, it outputs a result.

[0004] A common approach in building neural networks is to train them using a training data set before using them to solve real problems. In the training phase, input data for which the correct answer is already known is provided to the neural network, and the resulting output is used to train the network by adjusting the input weightings for one or more nodes. Many trained neural networks employ tens or hundreds of millions of parameters to achieve good performance. Storing neural networks that employ a large number of parameters takes significant amount of storage space. Neural networks may be used on devices with limited storage space, for example, mobile devices. Furthermore, these neural networks

may have to be transmitted via computer networks from one system to another. Transmitting such large neural networks via computer networks from one system to another can be a slow and inefficient process. Often times, in order to limit the size of the neural network for storage or transmission, the neural network may be compressed for storage and transmission, and decompressed by the computing device using the neural network.

BRIEF SUMMARY

[0005] The examples below describe apparatus and methods for generation of a compressed deep convolutional neural network (CNN), and in particular to compressing the CNN feature map and weight tensors with relatively high inference speed. A CNN has multiple layers where each layer has a feature map, and a weight tensor that defines the weighting applied to the results of the layer. The apparatus and methods compress the weight tensor and feature map of each level to optimize a rate-distortion-speed (RDS) objective function. The compressed data may be sent through a network to implement the CNN on other devices.

[0006] These examples are encompassed by the features of the independent claims. Further embodiments are apparent from the dependent claims, the description and the FIGs.

[0007] According to a first aspect, a method of generating a compressed representation of a neural network includes reordering a weight tensor into blocks compatible with a matrix multiplication operation. The reordered weight tensor is then quantized to provide a quantized reordered weight tensor. An input feature map is multiplied by the quantized reordered weight tensor to provide an output feature map. The output feature map and the weight tensor are compressed.

[0008] In a first implementation of the method according to the first aspect as such, the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients. The reordering of the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients, and the compressing of the reordered weight tensor includes: generating a map of the swapped columns of the reordered weight tensor, and quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

[0009] In a second implementation of the method according to the first aspect as such, the compressing of the reordered weight tensor further includes entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

[0010] In a third implementation of the method according to the first aspect as such, the output feature map includes zero-valued feature values and non-zero valued feature values. The reordering of the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values. The compressing of the reordered output feature map includes: generating a map of the swapped columns of the output feature map, quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values, and combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

[0011] In a fourth implementation of the method according to the first aspect as such, the method is performed by a layer of a multi-layer convolutional neural network (CNN). The method further includes: obtaining the input feature map as the output feature map from a previous layer of the CNN, obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN, and providing the output feature map as the input feature map to the subsequent layer of the multi-layer CNN.

[0012] In a fifth implementation of the method according to the first aspect as such, the method further includes: decompressing a compressed weight tensor from a previous layer of the multi-layer CNN, multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map, compressing the inference output feature map to provide an inference compressed output feature map, and determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

[0013] According to a second aspect, an apparatus for generating a compressed representation of a neural network, includes a memory having program instructions and a processor, coupled to the memory, wherein the instructions condition the processor to perform operations including: obtaining a weight tensor and an input feature map for the neural network, reordering the

weight tensor into blocks compatible with a matrix multiplication operation, compressing the reordered weight tensor to generate a compressed weight tensor, quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor, multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map, and compressing the output feature map to provide a compressed output feature map.

[0014] In a first implementation of the second aspect as such, the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients. The operation of reordering the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients. The operation of compressing the reordered weight tensor includes: generating a map of the swapped columns of the reordered weight tensor, and quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

[0015] In a second implementation of the second aspect as such, the operation of compressing the reordered weight tensor further includes entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

[0016] In a third implementation of the second aspect as such, the output feature map includes zero-valued feature values and non-zero valued feature values. The operation of reordering the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values. The operation of compressing the reordered output feature map includes: generating a map of the swapped columns of the output feature map, quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values, and combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

[0017] In a fourth implementation of the second aspect as such, the processor and the program instructions implement a layer of a multi-layer convolutional neural network (CNN). The operations further include: obtaining the input feature map as the output feature map from a previous layer of the CNN, obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN, and providing the output

feature map as the input feature map to the subsequent layer of the multi-layer CNN.

[0018] In a fifth implementation of the second aspect as such, the operations further include: decompressing a compressed weight tensor from a previous layer of the multi-layer CNN, multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map, compressing the inference output feature map to provide an inference compressed output feature map, and determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

[0019] According to a second aspect a computer readable medium includes program instructions that configure a computer processing system to perform operations to generate a compressed representation of a neural network. The operations include: obtaining a weight tensor and an input feature map for the neural network, reordering the weight tensor into blocks compatible with a matrix multiplication operation, compressing the reordered weight tensor to generate a compressed weight tensor, quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor, multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map, and compressing the output feature map to provide a compressed output feature map.

[0020] In a first implementation of the second aspect as such, the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients. The operation of reordering the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients. The operation of compressing the reordered weight tensor includes: generating a map of the swapped columns of the reordered weight tensor, and quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

[0021] In a second implementation of the second aspect as such, the operation of compressing the reordered weight tensor further includes entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

[0022] In a third implementation of the second aspect as such, the output feature map includes zero-valued feature values and non-zero valued feature

values. The operation of reordering the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values. The compressing of the reordered output feature map includes: generating a map of the swapped columns of the output feature map, quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values, and combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

[0023] In a fourth implementation of the second aspect as such, the computer processing system includes a layer of a multi-layer convolutional neural network (CNN). The operations further include: obtaining the input feature map as the output feature map from a previous layer of the CNN, obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN, and providing the output feature map as the input feature map to the subsequent layer of the multi-layer CNN.

[0024] In a fifth implementation of the second aspect as such, the operations further include: decompressing a compressed weight tensor from a previous layer of the multi-layer CNN, multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map, compressing the inference output feature map to provide an inference compressed output feature map, and determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

[0025] According to a fourth aspect, an apparatus for generating a compressed representation of a neural network, includes means for obtaining a weight tensor and an input feature map for the neural network, means for reordering the weight tensor into blocks compatible with a matrix multiplication operation, means for compressing the reordered weight tensor to generate a compressed weight tensor, means for quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor, means for multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map, and means for compressing the output feature map to provide a compressed output feature map.

[0026] In an implementation of the fourth aspect, the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients. The means for reordering the weight tensor includes means for swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients. The means for compressing the reordered weight tensor includes: means for generating a map of the swapped columns of the reordered weight tensor, and means for quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

[0027] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter. The claimed subject matter is not limited to implementations that solve any or all disadvantages noted in the Background.

BRIEF DESCRIPTION OF THE DRAWINGS

[0028] Aspects of the present disclosure are illustrated by way of example and are not limited by the accompanying FIGs. for which like references indicate elements.

[0029] FIG. 1 illustrates an example system in which a deep convolutional neural network (CNN) training system can operate according to an example embodiment.

[0030] FIG. 2 illustrates an example system for implementing a training engine to train a CNN according to an example embodiment.

[0031] FIGs. 3 and 4 illustrate convolutional operation according to example embodiments.

[0032] FIG. 5 illustrates an example memory hierarchy when performing a general block times panel (GEBP) matrix multiplication operation according to an example embodiment.

[0033] FIGS. 6A, 6B, 6C, and 6D illustrate examples of different sparse structures in CNNs.

[0034] FIGS. 7A and 7B respectively illustrate general panel times panel (GEPP)/GEBP and a general panel times matrix (GEPM)/GEBP sparse bitmap layouts of a kernel tensor according to example embodiments.

[0035] FIG. 8 illustrates an extension of GEPP/GEBP to a four dimensional (4D) convolution layer according to an example embodiment.

[0036] FIG. 9 illustrates an extension of GEPM/GEBP to a four dimensional (4D) convolution layer according to an example embodiment.

[0037] FIG. 10 illustrates an example quad-tree partitioning structure according to an example embodiment

[0038] FIGs. 11 and 12 illustrate an example of processing performed by a convolutional neural network (CNN) according to an example embodiment.

[0039] FIG. 13 illustrates a CNN training system according to an example embodiment.

[0040] FIG. 14 illustrates a rate-distortion-speed optimizer (RDSP) according to an example embodiment.

[0041] FIG. 15 illustrates a neural network feature map decoder according to an example embodiment.

[0042] FIG. 16 illustrates an example neural network processing system according to an example embodiment.

[0043] FIG. 17 is a block diagram of a processing system according to an example embodiment.

DETAILED DESCRIPTION

[0044] It should be understood at the outset that although an illustrative implementation of one or more embodiments is provided below, the disclosed systems, methods, and/or apparatuses described with respect to FIGs. may be implemented using any number of techniques, whether currently known or not yet in existence. The disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the example designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

[0045] In the following description, reference is made to the accompanying drawings that form a part hereof, and in which are shown, by way of illustration, specific embodiments which may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the inventive subject matter, and it is to be understood that other embodiments may be utilized, and that structural, logical, and electrical changes may be made

without departing from the scope of the present disclosure. The following description of example embodiments is, therefore, not to be taken in a limiting sense, and the scope of the present disclosure is defined by the appended claims.

[0046] The technology relates to generation of a compressed deep convolutional neural network (CNN), and in particular to compressing the CNN feature map and weight tensors with relatively high inference speed. As used herein, "inference" refers to the knowledge obtained from the CNN and "inference speed" refers to an amount of time used to process an input value through the CNN to produce a result. A CNN has multiple layers where each layer has a feature map, and a weight tensor that defines the weighting applied to the results of the layer. As the CNN processes training data, the layers select features to include in the feature map and each layer back-propagates weights to apply to the features used by previous layers in order to train the CNN to classify the input data. Each layer of the CNN receives the feature map from the prior layer and produces a feature map to be applied to a subsequent layer. The trained CNN includes multiple feature maps and weighting tensors, one set for each level of the CNN.

[0047] Weight tensors and feature maps to be compressed are received from each level of a neural network level, where a feature map refers to a filter kernel and weight tensor refers to, for example, a four-dimensional (4D) data container and each element in the data container is a floating-point number. The example system reorders both the feature map and the weight tensor to provide respective inner two-dimensional (2D) arrays and corresponding 2D sparse bitmaps. The example system compresses the layered structure that represents the reordered feature maps and weight tensors. The system selects respective encoding modes to generate a quantized reordered feature map and a quantized reordered weight tensor using one of a codebook (palette) or direct quantization. One component of the compression is to rearrange the data in the feature maps and weight tensors to consolidate zero-valued entries. This consolidation results in zero-valued factors for matrix multiplication operations, which allow some of the matrix multiplication operations to be skipped. The consolidation of the zero-valued entries is achieved through column swapping in which one column in the 2D array is swapped for another. As described below, the swapping can be reversed after the matrix multiplication. The example system encodes the column swapped, quantized, and reordered feature maps and weigh tensors to provide the

compressed representation of the neural network. This compressed representation may be transmitted to a target system for decompression and use.

[0048] Artificial neural networks have become an important tool for extraction of descriptors from multimedia content, for the classification and encoding of multimedia content, and other applications. Artificial neural networks have been adopted for a broad range of tasks in multimedia analysis and processing, media coding, data analytics and many other fields. While the underlying technology has been known for decades, the recent success is based on two main factors: (1) the ability to process much larger and complex neural networks than in the past, and (2) the availability and capacity of large-scale training data sets. These two aspects not only make trained networks powerful, but also mean that they generate a large number of parameters (feature maps and weights), resulting in quite large sizes of the trained neural networks (e.g., several hundred MBs).

[0049] The neural networks used in an application can be improved incrementally (e.g., training on more data, including feedback from validation of results), so that updates of already deployed networks may be necessary. In addition, the neural networks for many applications (e.g., classification) start from neural network that has been pre-trained on a general dataset, and then adapted and retrained for the specific problem. Thus, different applications may use neural networks that share large parts among them.

[0050] In conventional application of neural network compression, it has been shown that significant compression is feasible, with little or no impact on the performance of the neural network in a particular use case. As the description of the network topology is rather small compared to the parameters/weights, compression technology will in particular address compression of weights, e.g., by reducing their number, quantizing them, representing them more compactly etc.

[0051] A particular use case may deploy a trained neural network (and its updates) to a number of devices, which potentially run on different platforms. These use cases may benefit from using a compressed representation of the neural network. Compression enables an application to have smaller representations of neural networks sent across network connections, and potentially also neural networks having a smaller memory footprint in use. While exchange formats for

neural networks exist (e.g., ONNX, NNEF), they do not yet address compression and incremental updates. What is currently missing is a representation of the compressed parameters of a trained network, complementing the description of the network structure/architecture in existing (exchange) formats for neural networks.

[0052] Some of the use cases or applications for compressed neural networks include but are not limited to, a camera application with object recognition, a translation application including language, data, and or protocol translation, among other things, large-scale public surveillance, visual pattern recognition (VPR), processing and/or handling of electronic health record and genomic data, dynamic adaptive media streaming, audio classification / acoustic scene classification, audio classification / sound event detection, personalized machine reading comprehension (MRC) application, distributed training and evaluation of neural networks for media content analysis, compact descriptors for video analysis (CDVA), image/video compression, and distribution of neural networks for content processing, among others. These applications benefit from CNN representation for devices with limited memory and bandwidth, allowing efficient re-use of neural networks among different applications,

[0053] FIG. 1 illustrates an example system in which a CNN training system can operate. The system 100 includes one or more computing devices 102(1) – 102(N), including servers 104(1) – 014(N), that may communicate with one another via one or more networks 106. Networks 106 may be wired or wireless and include public networks or private networks including, but not limited to local area networks (LAN), wide area networks (WANs), satellite networks, cable networks, WiMaX networks, and communication networks, such as LTE and 5G networks. Networks 106 may also include any number of different devices that facilitate network communications, such as switches, routers, gateways, access points, firewalls, base stations, repeaters, backbone devices, etc.

[0054] Computing device(s) 102(1) – 102(N) may include, but are not limited to, any number of various devices, such as client or server based devices, desktop computers, mobile devices, special purposes devices, wearable devices, laptops, tablets, cell phones, automotive devices, servers, telecommunication devices, network enabled televisions, games consoles or devices, cameras, set top boxes, personal data assistants (PDAs) or any other computing device configured

to use a CNN training or operation as described herein. In one embodiment, computing devices 104(1) – 104(N) may include one or more processor(s) 110 connected to one or more computer readable media 112. The processor(s) may operate to execute computer readable and executable instructions stored on the computer readable media 112, which may be for example, an operating system (O/S) 112A, a CNN training engine 112B, a CNN operation engine 112C, and other programs or applications executable by processor(s) 110.

[0055] Processor(s) 110 may include, but is not limited to, one or more single-core processors, multi-core processors, central processing units (CPUs), graphics processing units (GPUs), general purpose graphics processing units (GPGPUs) or hardware logic components, such as accelerators and field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), system-on-a-chip (SoCs), complex programmable logic devices (CPLDs) and digital signal processors (DSPs).

[0056] Computer readable media 112 (or memory) may include computer storage media and/or communication media, which may comprise tangible storage units such as volatile memory, non-volatile memory or other persistent or auxiliary computer storage media, removable and non-removable computer storage media implemented in any method or technology for storage of information such as computer readable instructions, data structures or other data. Computer readable media 112 may include tangible or physical forms of media found in device or hardware components, including but not limited to, random access memory (RAM), static RAM, dynamic RAM, read only memory (ROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM), flash memory, optical storage, magnetic storage, storage arrays, network storage, storage area networks or any other medium that may be used to store and maintain information for access by a computing device, such as computer devices 102(1) – 102(N) and 104(1) – 104(N). In some embodiments, computer readable media 112 can store instructions executable by the processor(s) 110, which processor(s) 110 may be included in one or more of the computer devices 102(1) – 102(N) and 104(1) – 104(N). In still other embodiments, the computer readable media 112 may store an operating system which includes components to enable or direct the computing devices 102(1) – 102(N) and 104(1) – 104(N) to receive data via various input (e.g., memory devices, user controls,

network interfaces, etc.) and process the data using processor(s) 110 to generate output (e.g., and image for display, data for storing in memory, etc.) and which may enable a user to interact with various units of the training engine 112B.

[0057] In the disclosed embodiment, the computer-readable media 112 includes O/S 112A, a CNN training engine 112B and a CNN operation engine 112C. The O/S 112A may include software that allows applications to execute on the computing devices 102(1) – 102(N) and 104(1) – 104(N) and manages hardware resources, including input devices (e.g., keyboard and mouse), output devices (e.g., displays and printers), network devices (e.g., routers, network connections, etc.) and storage device (e.g., internal and external drives). Additionally, the O/S 112A may provide services to facilitate the efficient execution and management of, and memory allocations for, additionally installed software applications.

[0058] CNN training engine 112B may be implemented by the computing devices 102(1) – 102(N) and 104(1) – 104(N) to train a neural network model, such as CNN 113B. In one embodiment, the CNN training engine 112B includes an algorithm or software 120 (executable by the processor(s)) to train one or more CNNs. Training a CNN 112B may be performed by multiple nodes (e.g., computing devices) in parallel to reduce training time or by a single node sequentially implementing each level of the CNN. Accordingly, the CNN training engine 112B (and/or O/S 112A and CNN operation engine 112C) may execute on one or more of the computing devices 102(1) – 102(N) and 104(1) – 104(N). The CNN training engine 112B is described in more detail below with reference to FIG. 2. Once a CNN has been trained, operation of the trained CNN may then be implemented by a data analysis engine, such as CNN operation engine 112C, described below.

[0059] Computing device 102(1) – 102(N) and 104(1) – 104(N) can also include one or more communications interfaces 114 to enable wired or wireless communications between the computing device 102(1) – 102(N) and 104(1) – 104(N) involved in CNN training. Communications interface(s) 114 may include one or more transceiver devices, for example, network interface controllers (NICs) such as Ethernet NICs, to send and receive communications over a network, such as network 101. In one embodiment, the processor(s) 110 may exchange data through the communications interface 114. For example, the communications

interface 114 may be a Peripheral Component Interconnect express (PCIe) transceiver. Other examples include the communications interface 114 being a transceiver for cellular, Wi-Fi, Ultra-wideband (UWB), BLUETOOTH or satellite transmissions. The communications interface 122 can include a wired I/O interface, such as an Ethernet interface, a serial interface, a Universal Serial Bus (USB) interface, an INFINIBAND interface other wired interfaces.

[0060] FIG. 2 illustrates an example system 200 for implementing a training engine to train a CNN. The system 200 uses an algorithm, such as algorithm 204, to train one or more CNNs, and implements a data analysis engine, such as CNN operation engine 112C in which to operate the trained CNN 206. The training engine 112B and CNN operation engine 112C may be implemented using one or more computing devices, such as computing device 104(N). In one embodiment, the CNN training engine 112B and CNN operation engine may be implemented by the same computing device. In another embodiment, the CNN training engine 112B and CNN operation engine may be implemented by different computing devices. The computing device 104(N), as noted above, may include one or more processor(s) 110, which may exchange data through a bus or a network (not shown) as well as execute instructions of the CNN training engine 112B and the training data 203.

[0061] CNN training can be performed by multiple nodes (e.g. computing devices) in a parallel manner to reduce the time required for training. In one embodiment, the CNN training engine 112B uses an algorithm 204 to train the CNN 202 to perform data analysis. In the example as illustrated, the CNN 202 is a multi-layer convolutional neural network. Accordingly, the CNN 202 may include an input layer 202(N) and an output layer 202(1), and multiple intermediate layers, between the input and output layers. The training engine 112B may use training data 203 to train the CNN 202. In one embodiment, the training data 203 may include a collection of audio data that includes speech samples. For example, the audio data may include speech samples collected from speakers in North America or other languages, such as Chinese, Japanese or French. Still other kinds of training data may be collected for different applications such as handwriting recognition or image classification.

[0062] In one embodiment, computations performed by the algorithm 204 may be parallelized across processor(s) 110 and across different computing

devices 104(N). For example, during back-propagation, a computation on input data 208 performed by a first processor 110 may produce a first computation result. The first computation result may be pipelined to a second processor 110 for further computation to generate a second computation result. Concurrent with the generation of the second computation result, the first processor 110 may be processing additional input data 208 to generate a third computation result. Similarly, concurrent with the generation of the second computation result, the first processor 110 may be transferring at least part of the first computation result to another processor 110. Such concurrent computations by the processors 110 may result in a pipelining of computations that train the CNN training algorithm 204. Accordingly, computation time may be reduced due to the resulting parallelism of computation.

[0063] By using the algorithm 204 together with the training data 203, the CNN training engine 12B may produce trained CNN 206 from the CNN 202. CNN operation engine 112C may then use the trained CNN 206 to produce output data 210 from the input data 208. For example, the CNN operation engine 112C may perform pattern recognition and data analysis, such as speech recognition, speech synthesis, regression analysis or other data fitting, image classification, or face recognition (e.g., face recognition for determining driver distraction or images of a face in photos). In one specific example, the CNN operation engine 112C may receive image data from a camera or image processing components or a media file or stream. The input data may use a trained CNN 206 to recognize the output images (output data) 210 based on input images (input data) 208.

[0064] In one further embodiment, the computing device 104(N) may include a data store (not shown) that has data storage, such as a database or data warehouse. In one embodiment, data store includes a relational database with one or more tables, arrays, indices, stored procedures and the like which enable data access including one or more of hypertext markup language (HTML) tables, resource description framework (RDF) tables, web ontology language (OWL) tables, extensible markup language (XML) tables, etc. Data stored in data store may include, but is not limited to, data for the operations of processes, applications, components or modules stored in computer-readable media 112 or executed by processor(s) 110. In one embodiment, the data store stores training

data 203, a CNN 202 or other mathematical model, a trained CNN 206 or any combination thereof.

[0065] Additional CNNs are described below with reference to the various FIGs.

[0066] FIG. 3 illustrates an example convolutional operation in accordance with conventional techniques. CNNs have been successful at reducing storage and computational costs of large neural networks. As the number of layers and nodes in these networks increases, and devices (e.g., mobile devices) implementing these networks increasing have limited memory and computational resources, there exists a need to continually reduce storage and computational costs. Many conventional techniques exist to implement a convolutional operation—Caffe uses direct convolution using im2col (a method of rearranging image blocks into columns), Caffe2 uses Fast Fourier Transform (FFT) based convolution, and Tensorflow uses Winograd based convolution. In the examples that follow, and for purposes of discussion, if an input feature map is $D \in \mathbb{R}^{CHW}$ and a convolution filter is $F \in \mathbb{R}^{KCRS}$, the output feature map is represented by $O \in \mathbb{R}^{KPQ}$, where $P = f(H, R, u, pad\_h)$ and $Q = f(W, S, v, pad\_w)$. That is, the height and width of the output feature map depend on the height and width of the input feature map and filter, as well as the choice of padding and striding. The variables are defined in Table I.

C	Number of input feature maps
H	Height of input image
W	Width of input image
K	Number of output feature maps
R	Height of filter kernel
S	Width of filter kernel
u	Vertical stride
v	Horizontal stride
pad_h	Height of zero-padding
pad_w	Width of zero-padding

Table I

[0067] In the example embodiment, a direct convolution using im2col is illustrated. Filter (F) 302 is reshaped to a two-dimensional (2D) matrix F[K, CxRxS] 304, and the input feature map (D) 306 is reshaped to 2D matrix D[CxRxS, HxW] 308 after applying im2col([R, S]) to each pixel in the input feature map. The resulting output feature map 310 is O=F·D ('·' indicates matrix multiplication).

[0068] FIG. 4 illustrates another example convolutional operation in accordance with conventional techniques. The convolutional operation illustrated is a direct convolution using im2col, axis reorder. Filter (F) 402 may also be reshaped and reordered to form a [1, RxS] matrix 404, where each element of the matrix F_n is a [K, C] sub-matrix. The input feature map (D) 406 can be reshaped and reordered to form a [RxS, 1] matrix 408, where each element of the matrix D_n is a [C, HxW] sub-matrix. The output feature map 410 may then be generated as $O= \sum_{RxS} F_n \cdot D_n$.

[0069] An advantage to FFT based convolution is that it is significantly faster for large kernels because it transforms the input feature and kernel to a Fourier domain and multiplies them together to generate transformed output feature. A reverse transform is then performed to generate the output feature in a pixel domain. However, as most CNNs adopt deep layers with a small kernel (such as 3x3), Winograd based convolution outperforms FFT based convolution under these circumstances.

[0070] Applying the Winograd method, the output feature matrix is calculated using formula ('$\odot$' indicates element-wise matrix multiplication) as shown in equation (1):

$$o = a^T \cdot [\,(b \cdot f \cdot b^T) \odot (c^T \cdot d \cdot c)\,] \cdot a \qquad (1)$$

[0071] For the output feature matrix o[2, 2] and kernel f[3, 3] configuration, the input feature matrix d is a 4x4 matrix. Therefore, matrices a, b and c are given by equation (2):

$$a^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}, \quad c^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad (2)$$

[0072] A vec operator is then defined to create a column vector from matrix A by stacking the column vectors of $A = [a_1\ a_2 \ldots a_n]$ below one another. For matrix multiplication of $Y=M\cdot X\cdot N$, a Kronecker product may be applied so that $Y_{vec} = (N^T \otimes M)\cdot X_{vec}$ (where '$\otimes$' indicates the Kronecker product of two matrixes). Given that the number of input feature maps is C, one output feature is calculated by summing all the convolution result between the input features and their responding kernels. The Winograd formula can be implemented according to the following procedure:

1. F_{w_vec} = reshaping along outer axis of $(B\cdot F_{vec})$ to form a [N] elements vector, each vector element $F_{w\_vec}[n]$ is a [K, C] matrix.
2. D_{w_vec} = reshaping along outer axis of $(C\cdot D_{vec})$ to form a [N] elements vector, each vector element $D_{w\_vec}[n]$ is a [C, number_input_tile] matrix.
3. $O_{w\_vec} = F_{w\_vec} \odot\cdot D_{w\_vec}$ ($\odot\cdot$ indicates element-wise multiplication of vector F_{w_vec} and D_{w_vec}, while matrix multiplication is performed for each element pair since they are 2D matrixes, O_{w_vec} is a [N] elements vector, each vector element $O_{w\_vec}[n]$ is a [K, number_input_tile] matrix.
4. $O_{vec} = A\cdot O_{w\_vec}$.
5. Generate final output feature map O by reshaping O_{vec} to its proper output layout.

[0073] The Winograd configuration of the output feature matrix $o[2, 2]$ and kernel $f[3, 3]$ is used in a Tensorflow convolution layer implementation. The definition and dimension of each tensor in the Tensorflow implementation are listed in Table II below. Under this configuration, the dimension of F_{w_vec} is [16, [K, C]], the dimension of D_{w_vec} is [16, [C, number_input_tile]], and the dimension of O_{w_vec} is [16, [K, number_input_tile]].

$B= b \otimes b$	filter transform matrix	[16, 9]
$C= c^T \otimes c^T$	input data transform matrix	[16, 16]
$A= a^T \otimes a^T$	output transform matrix	[4, 16]

F_{vec}	vectorized filter tile	[9, [K, C]]
F_{w_vec}	F_{vec} in Winograd domain	[16, [K, C]]
D_{vec}	vectorized input tile	[16, [C, number_input_tile]]
D_{w_vec}	D_{vec} in Winograd domain	[16, [C, number_input_tile]]
O_{w_vec}	O_{vec} in Winograd domain	[16, [K, number_input_tile]]
O_{vec}	vectorized output tile	[4, [K, number_input_tile]]
O	reshaped output tile	[2, 2, K, number_input_tile]

Table II

[0074] For the Winograd configuration of output feature matrix $o[2, 2]$ and kernel $f[3, 3]$, matrices A, B and C are shown in equations (3)-(5):

$$A=\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 1 & -1 & -1 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & -1 & 1 & 1 & 0 & -1 & 1 & 1 \end{bmatrix} \quad (3)$$

$$B=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & -1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & -1/4 & 1/4 & 1/4 & -1/4 & 1/4 & 1/4 & -1/4 & 1/4 \\ 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & -1/2 & 0 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 & 1/4 & -1/4 & 1/4 \\ 0 & 0 & 1/2 & 0 & 0 & -1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & -1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4)$$

$$C=\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \quad (5)$$

[0075] As can be seen from the description above, corresponding to FIGs. 3 and 4, the direct convolution method using im2col is a matrix multiplication of 2D matrix [K, C] and [C, HxW], and the Winograd based convolution method is similarly a matrix multiplication of 2D matrix [K, C] and [C, number_input_tile]. Accordingly, it follows that both methods benefit from the use of a high efficiency matrix multiplication, for example, General Block Panel Multiplication (GEBP).

[0076] FIG. 5 illustrates an example memory hierarchy when performing a GEBP operation (or General Panel Panel Multiplication (GEPP) operation). Taking an example, and for purposes of discussion, where C+=A·B, and A, B and C are [(m x k), (k x n)], and [m x n] respectively, A, B and C are split into sub-blocks such that the multiplication operation can take full advantage of underlying hardware memory, cache, register and arithmetic logic unit (ALU) resources. The sub-blocks A, B and C are illustrated from left to right in the diagram. As shown, A[m, p] 502 is partitioned to column panels A_{main}[m, kc]; A_{main}[m, kc] is partitioned to row panels A_{L2}[mc, kc] 504; A_{L2}[mc, kc] is partitioned to row panels A_{L2r}[mr, kc] 506; A_{L2r}[mr, kc] is partitioned to columns panel A_{reg}[mr, 1] 508; B[p, n] is partitioned to column panels B_{main}[p, nc] 510; B_{main}[p, nc] is partitioned to row panels B_{L3}[kc, nc] 512; B_{L3}[kc, nc] is partitioned to column panels B_{L1}[kc, nr] 514; B_{L1}[kc, nr] is partitioned to row panels B_{reg}[1, nr] 516; Inner kernel brings the next A_{L2}[mc, kc] 504 to L2 cache, B_{L3}[kc, nc] 512

to L3 cache, and B_{L1}[kc, nr] 514 from L3 to L1 cache. It also brings the next A_{reg}[mr, 1] 508 and B_{reg}[1, nr] 516 to the register array; Inner kernel calculates C_{reg}[mr, nr] 518 where C_{reg}[mr, nr] = A_{reg}[mr, 1] · B_{reg}[1, nr] in the register domain; and the inner kernel calculation is repeated, followed by repeating Inner kernel brings next A_{L2}[mc, kc] 502 to L2 cache, B_{L3}[kc, nc] 512 to L3 cache, and B_{L1}[kc, nr] 514 from L3 to L1 cache. A_{reg}[mr, 1] 508 and B_{reg}[1, nr] 516 are then brought to the register array, until all blocks of matrix multiplication are completed.

[0077] Although not illustrated, a conventional memory hierarchy when performing General Panel Matrix Multiplication (GEPM) / General Block Panel Multiplication (GEBP) operation is similar to the GEBP/GEPP operation. Using this technique, A[m, p] is partitioned to row panels A_{main}[mc, p]; A_{main}[mc, p] is partitioned to column panels A_{L2}[mc, kc]; A_{L2}[mc, kc] is partitioned to row panels A_{L2r}[mr, kc]; A_{L2r}[mr, kc] is partitioned to columns panel A_{reg}[mr, 1]; B[p, n] is partitioned to column panels B_{main}[p, nc]; B_{main}[p, nc] is partitioned to row panels B_{L3}[kc, nc]; B_{L3}[kc, nc] is partitioned to column panels B_{L1}[kc, nr]; B_{L1}[kc, nr] is partitioned to row panels B_{reg}[1, nr]; Inner kernel brings next A_{L2}[mc, kc] to L2 cache, B_{L3}[kc, nc] to L3 cache, and B_{L1}[kc, nr] from L3 to L1 cache. A_{reg}[mr, 1] and B_{reg}[1, nr] are then brought to the register array; Inner kernel calculates C_{reg}[mr, nr] = A_{reg}[mr, 1] · B_{reg}[1, nr] in the register domain; followed by repeating the Inner kernel brings next A_{L2}[mc, kc] to L2 cache, B_{L3}[kc, nc] to L3 cache, and B_{L1}[kc, nr] from L3 to L1 cache, until all blocks of matrix multiplication are completed.

[0078] Parameters C, K for the 2D matrix are treated as syntax elements and may be stored in syntax table (not shown). In one embodiment, parameters kc, mc, mr, nc, nr, and p, are treated as syntax elements, and may also be stored in the syntax table. In another embodiment, parameters kc, mc, mr, nc, nr, and p are pre-defined, and it is not necessary to store them in the syntax table. In another embodiment, some of the parameters, kc, mc, mr, nc, nr, and p, are treated as syntax elements and stored in the syntax table, while other parameters are pre-defined and not stored in the syntax table.

[0079] Based on the above description, matrix A is more efficient if stored in a column-major order, or A_{L2r}[mr, kc] is transposed to become a column-major order, so that inner kernel can access A_{L2r} in a continuous manner.

[0080] FIGS. 6A – 6D illustrate examples of different sparse structures in CNNs. There are several methods to reduce the storage of large neural networks having a large number of layers and nodes (e.g., computing devices). Such methods include, but are not limited to, weight pruning so that a dense tensor can be represented by sparse tensor, low-rank factorization so that a large tensor can be represented by two small tensors, designing special structural convolution filters with fewer weight parameters, and training a small neural network from a large neural network.

[0081] Both weight pruning and low-rank factorization methods take a pre-trained model and perform a pruning or factorization process. The parameters used in pruning or factorization can also be quantized to binary, ternary, 4-bit, 8-bit, or X-bit value, or k-means scalar quantization can be applied to the parameter values. For example, Tensorflow Lite utilizes an 8-bit quantization method that takes a maximum and a minimum value from one layer and quantizes the parameter using the quantization step (max-min)/256.

[0082] These procedures are usually combined with model retraining so that the network accuracy is maintained. The weight pruning and low-rank factorization method can be performed in both the original filter domain and the Winograd filter domain. If weight pruning is performed in the original filter domain, the filter tensor in the Winograd filter domain is still a dense tensor.

[0083] Unlike a dense matrix, there are many ways to store a sparse matrix to save storage. Popular sparse matrix formats include Coordinate list (COO), compressed sparse row (CSR) and compressed sparse column (CSC). Bitmap is also an option to store sparse matrix where each bit indicates if the corresponding parameter is zero.

[0084] As illustrated, weight pruning structures may include fine-grained sparsity (FIG. 6A), vector-level sparsity (FIG. 6B), kernel-level sparsity (FIG. 6C) and filter-level sparsity (FIG. 6D). Low-rank factorization method offers good compression ratios. However, it does not increase inference speed due to the dense property of the restored tensor. This method does not have prior knowledge of the parameter values of the restored tensor before they are generated, consequently, it performs multiple multiplications even when the final parameter value is treated as zero, wasting processing time and increasing power consumption.

[0085] While, the fine-grained sparsity method offers a good compression ratio, it does not increase inference speed due to the irregularity of the fine-grained sparse. The vector-level sparse method explores sparse structure within individual kernels by setting row/column of parameters to zero. Kernel-level and filter-level methods set the parameters in one kernel or one filter to zero. However, since most state-of-the-art CNNs adopt deep layers with a small kernel (such as 3x3), these methods may have a more negative performance impact.

[0086] A high efficiency matrix multiplication GEBP is at the heart of convolution implementation. In order to gain the optimal performance, GEBP operation segments the matrixes in a special way, as illustrated in FIG. 4. The inference speed can be improved if the matrix sparsity is structured in the same way.

[0087] The size of input feature map (rhs, right-hand-side matrix, in GEBP) is usually bigger than the size of the filter tensor (lhs, left-hand-side matrix, in GEBP) in the first few convolution layers. Current GEBP process loads the rhs matrix from the L3 cache multiple times, the GEBP process also transfers the O (output) matrix to and from the L3 cache multiple times (and possibly from and to external Double Data Rate (DDR) memory, if L3 is not large enough to hold O matrix). The bandwidth consumption is large for devices without large L3 caches such as embedded chips, or devices without cache memory hierarchy such as field-programmable gate array (FPGA) and application specific integrated circuit (ASIC) chips.

[0088] The disclosure that follows presents various embodiments to increase neural network storage size and increase inference speed. Such embodiments may be implemented, for example, in the systems illustrated in FIGS. 1 and 2, as well as being deployed in desktop CPUs, embedded CPUs and ASIC platforms (for example, without GPU). The embodiments described below are well suited to work with existing methods, such as the direct convolution method, Winograd based convolution method and low-rank factorization method (described above).

[0089] For the direct convolution method, the convolution filter is reshaped from $F \in \mathbb{R}^{KCRS}$ to $F \in \mathbb{R}^{RSCK}$, where each element of the [R, S] kernel tensor $F_{RS} \in \mathbb{R}^{CK}$ (column-major order or transpose of the lhs matrix of GEBP operation) is a 2D matrix [C, K]. The input feature map (output feature map from

previous layer, after a rectified linear unit (relu) operation (an activation function operation), max pooling operation, and an im2col process) is reshaped to $D \in \mathbb{R}^{RSCHW}$, where each element of the [R, S] input feature map tensor $D_{RS} \in \mathbb{R}^{CHW}$ (rhs matrix of GEBP operation) is also a 2D matrix [C, HxW].

[0090] For the Winograd based convolution method, each element of the [16] kernel tensor $F_{RS} \in \mathbb{R}^{CK}$ is a 2D matrix [C, K]. It is appreciated that the solution also works with other shapes of the F.

[0091] For a direct convolution method, the input feature map (output feature map from previous layer, after relu operation, max pooling operation and Winograd transformation) is reshaped to $D \in \mathbb{R}^{16CHW}$, where each element of the [16] input feature map tensor $D_{RS} \in \mathbb{R}^{CHW}$ is a 2D matrix [C, HxW]. Similarly, the solution works with other shapes of D as well.

[0092] For the Winograd based convolution method, the input feature map (output feature map from previous layer, after relu operation, max pooling operation and Winograd transformation) is reshaped to $D \in \mathbb{R}^{16CHW}$, each element of the [16] input feature map tensor $D_{RS} \in \mathbb{R}^{CHW}$ is a 2D matrix [C, HxW]. The principle of the solution works with other shape of D as well.

[0093] The compression process for filter tensor and/or output feature map (input feature map for next layer) includes the following operations: (a) assign different bit depths to the different layer; (b) perform a sparse operation to remove unimportant coefficients; (c) partition the kernel tensor [C, K] to smaller coding tree units (CTUs) having a size of [mc, kc], based on memory and register configuration of the system; (d) adaptively partition the CTU to smaller coding units (CUs) to form a coding tree partitioning structure; (e) for each CU, scale and quantize the coefficients to integers, then represent the integer coefficients by palette table and index map; (f) using different coding methods to encode palette table and index map, calculate distortion between quantized coefficients and original coefficients, select a partitioning structure that tends to decrease rate distortion (RD) or rate distortion inference speed (RDS).

[0094] The inference process determines the RD or RDS. This process uses layered sparse bitmaps from both kernel tensor and input feature map as additional inputs to GEBP so that it skips operation for zero-valued blocks (e.g. Z[1, mr], Z[kc, mr], Z[kc, mc], Z[kc, K], Z[C, K] blocks in lhs matrix, or Z[1, nr],

Z[kc, nr], Z[kc, nc], Z[C, nc], Z[C, HxW] block in rhs matrix where "Z" indicates that all elements in this block are zero).

[0095] As an initial step, a CNN is trained. The training process includes defining a sparse bitmap, such as a 2D [C, K] sparse bitmap, to represent a sparse structure that matches an underlying GEBP lhs matrix blocking structure (described below) for each element of the kernel tensor F_{RS}. In one embodiment, the CNN may be newly trained or may comprise a pre-trained CNN. A sparse operation may be performed to F_{RS} during the training (or retraining) process. When performing the sparse operation, the weight parameter may be arbitrarily changed and/or the CNN cost function may be changed such that the weight matrix has more Z[1, mr] rows, more Z[kc, mr] blocks, more Z[kc, mc] blocks, more Z[kc, K] blocks, or more Z [C, K] blocks. Optionally, a column swap operation can then be performed if the sparse operation generates more Z[1, mr] rows, more Z[kc, mr] blocks, more Z[kc, mc] blocks, more Z[kc, K] blocks, or more Z [C, K] blocks (the operation will result in a corresponding row swap in final GEBP output). The input feature map may also be a sparse tensor during the training process. For example, each element of the sparse input feature tensor D_{RS}, a 2D [C, HxW] sparse bitmap may be defined to represent a sparse structure that matches with underlying the GEBP rhs matrix blocking structure, described below.

[0096] FIGS. 7A and 7B respectively illustrate GEPP/GEBP and a GEPM/GEBP sparse bitmap layouts of a kernel tensor. For the direct convolution method, the convolution filter is reshaped from $F \in \mathbb{R}^{KCRS}$ to $F \in \mathbb{R}^{RSCK}$, where each element of the [R, S] kernel tensor $F_{RS} \in \mathbb{R}^{CK}$ (column-major order or transpose of the lhs matrix of GEBP operation) is a 2D matrix [C, K]. For the Winograd convolution method of the output feature matrix o[2, 2] and kernel f[3, 3], each element of the [16] kernel tensor $F_{RS} \in \mathbb{R}^{CK}$ is a 2D matrix [C, K]. It is appreciated that other Winograd configurations may also be applied. For each F_{RS}, a 2D [C, K] sparse bitmap is defined to represent a sparse structure that matches with the underlying GEBP lhs matrix blocking structure.

[0097] In particular, FIG. 7A illustrates a GEPP/GEBP sparse bitmap layout of F_{RS}, and FIG. 7B illustrates a GEPM/GEBP sparse bitmap layout of F_{RS}. For purposes of discussion, the scan orders inside the [kc, mc] blocks are identical, and the scan orders of [kc, mc] blocks are different. For the GEPP/GEBP layout

(FIG. 7A), the sparse bitmap layout of F_{RS} is divided to ceil(C, kc) (where 'ceil' is a ceiling operation that converts a floating-point number to the smallest integer that is bigger than the floating-point number) row panels, where each of the row panel F_{kc} has a dimension of [kc, K] (except for the last one, if C is not dividable by kc). This row panel is further divided to ceil(K, mc) column panels, where each of the column panels F_{mc} has a dimension of [kc, mc] (except for the last one, if K is not dividable by mc). This column panel is further divided to ceil(mc, mr) column panels, where each of the column panels F_{mr} has a dimension of [kc, mr] (except for the last one, if mc is not dividable by mr). This column panel is further divided to a kc row, where each of the rows R_{mr} has dimension of [1, mr].

[0098] For the GEPM/GEBP layout (FIG. 7B), the sparse bitmap layout of F_{RS} is divided to ceil(C, mc) column panels, where each of the column panels F_{mc} has a dimension of [C, mc] (except for the last one, if C is not dividable by mc). This column panel is further divided to ceil(C, kc) row panels, where each of the row panels F_{kc} has a dimension of [kc, mc] (except for the last one, if C is not dividable by kc). This row panel is further divided to ceil(mc, mr) column panels, where each of the column panels F_{mr} has a dimension of [kc, mr] (except for the last one, if mc is not dividable by mr). This column panel is further divided to kc row, where each of this row R_{mr} has dimension of [1, mr].

[0099] For a matrix multiplication, it is well known that if two rows are swapped in the lhs matrix, the result is a corresponding row swap in the final multiplication output. This row swap (column swap in transposed lhs) operation is adopted to produce more ZR_{mr} blocks. Fox example, if one R_{mr} is [0, 0, 0, x] and another R_{mr} is [x, 0, x, 0], the fourth columns of these two R_{mr} can be swapped so that the first R_{mr} becomes a ZR_{mr}.

[0100] Both GEPP and GEPM operation utilize GEBP as underlying multiplication engine. GEPM operation is a more favorable choice for cache or bandwidth constraint device. GEPM outputs [mc, kc] block using minimum storage ([mc, kc]) for intermediate results. A carefully chosen blocking parameters mc and kc will result in the intermediate storage of [mc, kc] stays in L2 or L3 cache instead of being written to/read from DDR memory external to the processor.

[0101] GEPM also outputs matrix slice by slice so that the following relu operation, max pooling operation (if existed), im2col or Winograd transformation

can be calculated in a pipelined fashion. The uncompressed output can be generated slice by slice so that the some or all compression steps can be performed without waiting for the completion of uncompressed output.

[0102] General matrix multiplication (GEMM) operation is designed for two-dimension tensor (Matrix) multiplication but can be extended efficiently for more-than-two-dimension tensors, such as a 4D convolution layer with shape of [R, S, K, C]. As illustrated in FIGs. 8 and 9. This extension flattens the additional dimensions to form a three-dimension tensor with shape of [RS, K, C], and extends the 2D [mc, kc] block in original GEMM operation to become a 3D [RS, mc, kc] block.

[0103] The GEMM parameters such as mc, kc are recalculated so that the GEMM operation can still be performed in cache memory. This GEMM extension saves RS-1 times of DDR memory access for input feature set because block [kc, nc] from input feature can be loaded to cache memory once instead of RS times.

[0104] In an example embodiment, F_{RS} is defined as the kernel tensor of a 4D convolution layer or a 2D fully connected layer. It is in column-major order or transpose of row-major order of lhs matrix in GEBP operation. The original lhs matrix has shape of 2D [K, C] so F_{RS} has shape of 2D [C, K]. F_{RS} is partitioned with non-overlapping [kc, mc] rectangles which are the CTUs.

[0105] In one embodiment, shown in FIG. 8, the scan order between CTUs is a horizontal raster scan order to match with the scanning sequence of GEPP/GEBP operation. In another embodiment, shown in FIG. 9, the scan order between CTUs is a vertical raster scan order to match with the scanning sequence of GEPM/GEBP operation. In yet another embodiment, a syntax element may be defined for each layer or sublayer to indicate that the scan order used in the layer or sublayer.

[0106] When the GEMM operation is extended for a tensor having more than two-dimensions, such as 4D convolution layer with shape of [R, S, K, C], (e.g., F_{RS-C-K} is defined as the flattened 3D tensor of a 4D convolution layer) F_{RS} is in column-major order or transpose of row-major order of lhs matrix in GEBP operation. The original lhs matrix has shape of 2D [K, C] so F_{RS-C-K} has shape of 3D [RS, C, K].

[0107] $F_{RS\text{-}C\text{-}K}$ is partitioned with non-overlapping [RS, kc, mc] rectangles which form the 3D CTUs (CTU3D). In one embodiment, the scan order between CTU3Ds is a horizontal raster scan order to match with the scanning sequence of GEPP/GEBP operation. In another embodiment, the scan order between CTU3Ds is a vertical raster scan order to match with the scanning sequence of GEPM/GEBP operation. In yet another embodiment, a syntax element is defined for each layer or sublayer to indicate that the scan order used in this layer or sublayer.

[0108] In an example embodiment, kc=64/mc=64 are selected to match with GEBP cache size requirement for most modern processors. In another embodiment, arbitrary kc/mc values are selected, for example, to accommodate different processors or ASIC devices. In another embodiment, a syntax element is defined in the bitstream header to indicate the maximum value of a CTU dimension. For example, two bits of syntax element can be defined such that "00" indicates a first maximum CTU dimension (e.g., 128x128), "01" second maximum CTU dimension (e.g., 64x64), "10" indicates a third maximum CTU dimension (e.g., 32x32), and "11" indicates a fourth maximum CTU dimension (e.g. 16x16).

[0109] A CTU may be further partitioned to smaller coding unit (CU), each CU can be partitioned to even smaller CU to form a coding tree partitioning structure. The maximum depth of the coding tree, or smallest allowable CU size is also defined. In one embodiment, the smallest CU size is defined as 8x8. It is noted that when CTU is smaller than kc=64/mc=64, for example, CTU at right or bottom of the F_{RS}, the CU is padded with null values (e.g. zeros) to kc=64/mc=64 in order to form the coding tree partitioning structure, any CUs that are completely outside the original CTU are marked as non-existent CUs and such that the scanning processing for the nonexistent CUs is skipped.

[0110] One embodiment implements a "quad-tree" partitioning structure such that a 2Nx2N CU is partitioned into 4 NxN CUs. For example, the High Efficiency Video Coding (HEVC) standard specifies quad-tree partitioning. When the total RD of the 4 NxN CUs is smaller than the RD of 2Nx2N CU, the 2Nx2N CU is not partitioned which is signaled by a split flag=0 in the bitstream, otherwise the split flag=1 in the bitstream.

[0111] A rate-distortion-speed-optimization RDSO (e.g., the cost function for the CNN) is defined in order to improve compression quality and inference speed. The optimization that leads to the smallest RDSP is treated as the optimized RDSP, where RDSP is defined by equation (6):

$$RDSP = D + \lambda R + \mu S \ldots \quad (6)$$

where λ and μ are parameters used to weight the contributions of rate and inference speed in the RDSP calculation.

[0112] Rate, R, is the compressed bit count of the kernel tensor. Distortion, D, may be measured in several different ways. In one embodiment, distortion is measured by the difference between network accuracy performance when utilizing original weight value and network performance when utilizing the reconstructed weight value. In another embodiment, distortion is measured by a difference between the output feature map in a target layer when utilizing the original weight values and the original input feature map when compared to the output feature map in the same layer when utilizing the reconstructed weight values and the original input feature map. In yet another embodiment, distortion is measured by a difference between the output feature map in target layer when utilizing the original weight values and the original input feature map when compared to the output feature map in the same layer when utilizing the reconstructed weight values and the reconstructed input feature map. In still another embodiment, distortion is measured by the difference between original parameter value and reconstructed parameter value. Both L1 and L2 norm can be applied evaluate the distortion using any of these methods.

[0113] The rate-distortion-optimization ($RD = D + \lambda R$) is adequate if inference speed is not a concern. Otherwise, an S factor is defined to indicate the inference speed. For example, inference speed may be defined in relation to a number of multiply accumulate (MAC) operations and a number of memory accesses performed in the GEBP matrix multiplication process. To simplify the calculation, inference speed S may be represented by the number of MAC operation in GEBP process. For example, S may be defined as the number of NZR_{mr} rows, or any other definition that represents the number of MAC operations. Based on the property of matrix multiplication, the MAC operation can be skipped if either ZR_{mr} is found in lhs matrix or ZDR_{nr} is found in rhs matrix. Consequently, when the input feature map is a layered sparse bitmap it can be

combined with the layered sparse bitmap of kernel tensor to calculate S factor by discounting the MAC operations that can be skipped.

[0114] The low-rank factorization method offers a good compression ratio. However, it does not increase inference speed due to the dense property of the restored tensor. This method does not have prior knowledge of the parameter values of the restored tensor before they are generated. In order to obtain an improved RDS for low-rank factorization method, an array of swapped column indices and the layered sparse bitmap are encoded together with two low-rank matrixes. The swapped column indices and the layered sparse bitmap are used to direct low-rank factorization method to generate only non-zero coefficients of the sparse matrix, and direct underlying GEBP to skip operating on all-zero blocks in lhs matrix of GEBP operation.

[0115] For CTU/CU in the kernel tensor of a 4D convolution layer or a 2D fully connected layer, a base bit-depth are assigned to each layer. This bit-depth can be identical for all layers in neural network model or it can be different for some or all layers, depending on the statistics of each individual layer. A modified QP based quantization method (similar to the quantization method adopted by any video compression standard such as HEVC) is used to quantize and transform floating-point coefficients to integer.

[0116] First, standard deviation (layer_std) and max (layer_max), min (layer_min) value are calculated for given layer. Next, A layer_scale is defined to map layer_cmaxw to ((1 << base_bitdepth) - 1) when QP=0 is used for quantization. The expression layer_std * N (N=4) is used to determine the value of std4_maxw, other value of N can be used for different embodiment. Table III provides example program code for this calculation

```
maxw = std::max(std::abs(layer_max), std::abs(layer_min));
std4_maxw = std::min(layer_std * 4, maxw);
layer_cmaxw = (int)std::ceil(std4_maxw * 256);
layer_scale = ((1 << base_bitdepth) - 1) / (float(layer_cmaxw) / 256.0f);
invQuantScales[] = { 40, 45, 51, 57, 64, 72 };
layer_scale = layer_scale * invQuantScales[0];
```

Table III

Next, coefficients having absolute values greater than std4_maxw are assigned extra bit depth for quantization as shown in the example program code of table IV

```
qmaxw = quant_weight(maxw, false);
delta_bitdepth = std::max(get_bitdepth(qmaxw) - base_bitdepth, 0);
```

Table IV

After assigning the extra bit depth, the floating-point coefficients, w, are transformed to integer values Q and Q is clipped to stay within the max bit-depth as shown in Table V.

```
qp_per = qp / 6;
qp_rem = qp % 6;
quantScales[] = { 26214, 23302, 20560, 18396, 16384, 14564 };
point5 = 1 << (19 + qp_per);
Q = sign * Int64(layer_scale * w * quantScales[qp_rem] + point5) >> (20
        + qp_per)

clipMinimum = -(1 << (base_bitdepth + delta_bitdepth));
clipMaximum = (1 << (base_bitdepth + delta_bitdepth)) - 1;
ClipQ = std::min(std::max(Q, clipMinimum), clipMaximum);
```

Table V

[0117] De-quantization is defined to transform quantized integer Q to floating-point DQ as shown in the example program code of Table VI.

```
DQ = (invQuantScales[qp_rem] * (Q << qp_per)) / layer_scale
```

Table VI

[0118] In an example embodiment the value std4_maxw is multiplied by 256 to maintain accuracy, values other than 256 can be used in other embodiments. layer_scale is defined to map layer_cmaxw to ((1 << base_bitdepth) - 1) when QP=N is used for quantization. In an example embodiment N=0 although it is contemplated that other value can be used in other embodiments.

[0119] In one embodiment, a rounding procedure is used to set the value of DQ closer to the value of w. In another embodiment, this rounding procedure (add 0.5 and round to nearest integer) is removed so that the result can be scaled

to floating-point after the integer-to-integer version of GEMM is performed during the inference process.

[0120] For a coefficients tensor with a 1D shape, such as the bias element in a convolution/fully connected layer or the bn array in batch normal layer, the 1D tensor may be quantized with a fixed bit-depth (bitdepth = bias_bitdepth) and without using QP, if a coefficient is quantized to zero using bias_bitdepth, the quantization is performed again with another fixed bit-depth (bitdepth = bias0_bitdepth). Table VII shows example program code of Table VII

```
cmaxw = std::max(std::abs(layer_max), std::abs(layer_min));
layer_cmaxw = (int)std::ceil(cmaxw * (1 << bitdepth));
layer_scale = ((1 << bitdepth) - 1) / (float(layer_cmaxw) / (1 << bitdepth));
point5 = 1 << (20 - 1);
Q = sign * Int64(layer_scale * std::abs(w) * (1 << 20) + point5) >> (20);

clipMinimum = -(1 << bitdepth);
clipMaximum = (1 << bitdepth) - 1;
ClipQ = std::min(std::max(Q, clipMinimum), clipMaximum);
```

Table VII

[0121] A de-quantization is defined to transform quantized integer Q to floating-point DQ as shown by the example program code of Table VIII:

```
DQ = sign * std::abs(Q) / layer_scale;
```

Table VIII

[0122] A column_swap_enable flag is defined to indicate that if column swapping is enabled.

[0123] Swapped column indices are stored in column_swap_array[N], where N=K, the number of output features. Value j in entry i of column_swap_array indicates that column i is swapped to column j. Column swapping operation can also be constrained (N<K) so that the swapping operation can only be performed within every N column.

[0124] A column_swap_bit_array is defined to indicate if this column is swapped. It can be coded using deferent methods, some of the methods are

illustrated in below embodiments. An example program code statement is shown in Table IX

column_swap_bit_array[i] = (i ≠ column_swap_array[i])

Table IX

[0125] In an example embodiment, swapping operations are chained together, if entry in of column_swap_array is nth column in the swap chain (whose entry value is jn), where $0 \leq n < N$, swap chain rule ensures that the last entry value in the swap chain equals to the first entry index in the swap chain (jN-1 = i0). Furthermore, column_swap_array itself can be reordered so that all elements in each swap chain are grouped together, as illustrated in below table. Tables IX and X below.

Original index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Original value	~~*0*~~	*3*	*4*	*1*	*2*	*7*	~~*6*~~	*11*	~~*8*~~	~~*9*~~	~~*10*~~	*5*	*14*	~~*13*~~	*12*	~~*15*~~
Bit_array	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*0*	*1*	*1*	*0*	*1*	*0*

Entry index, entry value, and bit_array of column_swap_array

Table X

Inferred index	0	1	3	2	4	5	7	11	6	8	9	10	12	14	13	15
Reordered value	~~*0*~~	*3*	~~*1*~~	*4*	~~*2*~~	*7*	*11*	~~*5*~~	~~*6*~~	~~*8*~~	~~*9*~~	~~*10*~~	*14*	~~*12*~~	~~*13*~~	~~*15*~~
Reordered bit_array	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*

Inferred entry index, entry value, and bit_array for reordered column_swap_array

Table XI

[0126] In the tables above, the following operations may be performed:

1) swap_run: a run of 1s in column_swap_bit_array;

2) non_swap_run: a run of 0s in column_swap_bit_array;

3) swap_value: entry values in column_swap_bit_array, having corresponding column_swap_bit_array[] that are 1s;

4) non_swap_value entry: values in column_swap_bit_array, having corresponding column_swap_bit_array[] that are 0s;

swap_value can be encoded after all column_swap_bit_array is encoded, it can also be encoded immediately after swap_run is encoded. Coding of non_swap_values is skipped as these entries can be inferred.

[0127] In one embodiment, the encoding of column_swap_array is done on original column_swap_array without reordering. If last run is 1 (a single element run), this run (always a single element of non_swap_run) and its non_swap_value are skipped as they can be inferred. In another embodiment, the column_swap_bit_array is encoded by CABAC (Context-based Adaptive Binary Arithmetic Coding) engine one bit at a time. In yet another embodiment, the column_swap_bit_array is encoded by modified run-length coding. column_swap_bit_array[0] is encoded first (the rest of run_values are skipped as they can be inferred), followed by the sequence of runs (swap_run-1 and non_swap_run-1). In still another method, encoding of column_swap_array is done based on the reordered column_swap_array where swap_run indicates total the number of swapped columns in one swap chain (multiple swap_runs for multiple back-to-back swap chains). If column_swap_bit_array is encoded one bit at a time, a zero is inserted after swap_run of each swap chain. If last run is 1 (a single element run), this run (always a single element of non_swap_run) and its non_swap_value are skipped as they can be inferred.

[0128] Multiple column_swap_bit_array element skipping techniques may be used in this method. These include (a) skipping the coding of bits immediately after non_swap_run these bits can be inferred (always swap_run) and/or (2) when last run is 1 (a single element run), this run is skipped as it can be inferred (always a single element of non_swap_run).

[0129] Multiple column_swap_array element skipping techniques may be used in this method. These include (a) skipping coding of non_swap_values as

they can be inferred and/or skipping the coding of the last swap_value in a swap chain since it always equals to the first inferred entry index in this swap chain.

[0130] In one embodiment, the column_swap_bit_array is encoded by CABAC engine one bit at a time. A zero is inserted after swap_run of each swap chain. The skipping techniques described above are used in the coding of the reordered column_swap_bit_array and the column_swap_array.

[0131] In another embodiment, the column_swap_bit_array is encoded by run-length coding which encodes all pairs of [bits (0 or 1), runs (non_swap_run-1 or swap_run - 1)]. The skipping techniques described above are used in the coding of the reordered column_swap_bit_array and the column_swap_array.

[0132] In yet another embodment, different encoding methods can be used to encode the column_swap_array without encoding the column_swap_bit_array. For example: In one embodiment, for each swap chain, the swap_run of the swap chain can be encoded, followed by the first inferred index, followed by the sequence of swap_values in this swap chain (coding of last swap_value in a swap chain can be skipped). In another embodiment, for each swap chain, the first inferred index is encoded, followed by the sequence of swap_values in this swap chain.

[0133] In an example embodiment, a palette generation process is applied to the quantized coefficients of each CU. Each value in palette table indicates a quantized value. The maximum palette size is predefined. In one embodiment, the maximum palette sizes are identical for all CUs, in another embodiment, the maximum palette sizes are different for CUs with different dimension. A palette predictor with maximum size (P) is also defined to facilitate the palette generation process. The palette predictor is used to store the palette value generated from previous CUs.

[0134] The palette generation process first calculates the histogram of current CU and optionally merges neighboring bins to form new bins, the process counts the number of entries in the bin and sums the coefficients in the bin to calculate the center value for the bin. The method selects several bins having occurrence frequencies that are higher than a threshold and uses a quantized version of the center values as the initial values for the palette. A newly generated palette value is then compared with the previously generated palette values in the

palette predictor; the newly generated palette value is replaced by stored palette value in palette predictor if a better RD is obtained by using the palette value in palette predictor. Once the palette generation process is completed, the palette values are reordered so that the palette values that are found in predictor are always at the beginning of palette table and follow an ascending order in predictor.

[0135] Palette predictor is updated after the coding of current CU is complete. The palette table of current CU is inserted at the beginning of the palette predictor, and then any duplications are removed. When the total number of predictor values is more than the maximum size P, the method resizes the predictor so that only first P values are kept. In one embodiment, the palette predictor is initialized once and never reset. In another embodiment, palette predictor is reset at the beginning of each F_{RS}. In yet another embodiment, the palette predictor is reset at the beginning of each filter tensor F.

[0136] The palette table contains palette values that are found in predictor (inferred predictor value) and palette values that are not found in predictor (signaled values). The method encodes inferred predictor value by generating a bitmap with bit value "1" to indicate that the value of this predictor entry is inferred by palette table and with bit value "0" to indicate that the value of this predictor entry is not inferred by the palette table. There are several methods to encode this predictor bitmap.

[0137] In one method, an offset value is defined for this CU, the absolute value of delta value (total number of inferred predictor value - offset) is encoded first, followed by the sign bit of delta value if it is a non-zero value. Next, the distance between a current "1" bit value and the previous "1" bit value is encoded as the indication of the location of current "1" bit value; the location index in the predictor bitmap is encoded for the first occurrence of "1" bit value one. In one embodiment, this offset is identical across all CUs. In another embodiment, this offset is different among some or all CUs. In another embodiment, this offset is explicitly signaled as a syntax element. In yet another embodiment, this offset is inferred implicitly so that decoder can regenerate it using the statistics. In still another method, the bit value of the predictor bitmap is encoded one bit at a time. In another embodiment, the bit value of the predictor bitmap is encoded one bit at a time until the last value of one encoded, then the predictor bitmap is stuffed with one or more zeros are stuffed until the total number of inferred predictor value

plus the total number of stuffed zeros equals to maximum size of predictor (P). In another method, the (1 - bit value) of the predictor bitmap is encoded one bit at a time until the last value of one encoded, zero or one or more ones are then stuffed until the total number of inferred predictor value plus the total number of stuffed zeros equals to maximum size of predictor (P).

[0138] In another embodiment, an offset value is defined for this CU, the absolute value of delta value (total number of inferred predictor value - offset) is encoded first, followed by the sign bit of delta value if it is a non-zero value. Next, the bit value or ("1" bit value) of the predictor bitmap is encoded one bit at a time until the last value of one encoded, this method does not use stuffing bits.

[0139] In one embodiment, only one of the predictor bitmap encoding methods, described above, is selected to use so that its method ID can be implicitly inferred by decoder. In another embodiment, multiple predictor bitmap encoding methods are used and the method that generates the smallest RD is selected as the method to be used, an identifier of this method is treated as syntax element and stored in syntax table.

[0140] The method encodes the signaled values, by first encoding the total number of signaled values and then encoding the absolute value of each signaled value (followed by the sign bit if it is a non-zero value) one by one.

[0141] To generate the index map, the difference between the given quantized coefficients and the contents of palette table are calculated, when the smallest difference is smaller than a threshold, the index of the palette entry that generates smallest difference is selected as the index to represent the quantized coefficient. When the smallest difference is not smaller than a threshold, the value of palette table size (Escape index) is assigned to indicate that an escape coding process is used to represent the quantized coefficient. The materials below describe several competing coding methods may be used to encode the index map, the encoding method that generates smallest RD is selected as the encoding method to be used and an identifier of the selected encoding method is treated as syntax element and stored in the syntax table.

[0142] A first method is the string copy method. In embodiment of this method, multiple scan orders such as raster scan and traverse scan are used to encode the index map, the scan order that generate smallest RD is selected as the

scan method to be used. In another embodiment, one scan order is pre-selected to encode the index map such that the scan order identifier is not explicitly signaled.

[0143] Two copying modes are defined for scanning through the index map. A COPY_ABOVE mode indicates that the current index is identical to the index from above row and a COPY_INDEX mode indicates that the current index is identical to the index from previous scanned position. A run value is defined to indicate the number of consecutive indices with same mode.

[0144] Scanning from a start position (startp), the method first obtains run values of both copying modes for string suing startp as start position, then the mode identifier and run value are used to calculate RD for COPY_ABOVE mode and the mode identifier, run value and index value are used to calculate RD for COPY_INDEX mode. The mode that generates smallest RD is selected as the mode to be used and the new scan start position is set to be (startp + run of selected mode).

[0145] When comparing between two indices, escape indices are treated as equal, when the escape values of the indices are encoded in escape coding process. This is illustrated by the example 8x8 CU shown in Table XII.

Row 0	1	2	3	4	5	6	7	8
Row 1	9	2	3	4	5	6	7	8
Row 2	9	2	4	4	4	4	4	..
Row 3	...	..	..	..	..	..	..	..
Row 4	...	..	..	..	..	..	..	..
Row 5	...	..	..	..	..	..	..	..
Row 6	...	..	..	..	..	..	..	..
Row 7	...	..	..	..	..	..	..	..

Table XII

[0146] In this CU, first 9 indices are assigned with the values shown in Table XIII

(mode=COPY_INDEX, run=1, index=1),
(mode=COPY_INDEX, run=1, index=2),
(mode=COPY_INDEX, run=1, index=3),
(mode=COPY_INDEX, run=1, index=4),
(mode=COPY_INDEX, run=1, index=5),
(mode=COPY_INDEX, run=1, index=6),
(mode=COPY_INDEX, run=1, index=7),

(mode=COPY_INDEX, run=1, index=8),
(mode=COPY_INDEX, run=1, index=9),

Table XIII

[0147] The following 9 indices are assigned with (mode=COPY_ABOVE, run=9), and The following 5 indices are assigned with (mode=COPY_INDEX, run=5, index=4).

[0148] The coding mode is always COPY_INDEX for first row so the coding of mode ID can be skipped as it can be inferred. The coding mode after COPY_ABOVE is always COPY_INDEX so the coding of mode ID can be skipped as it can be inferred. For the index (whose index value is A) at the starting position of COPY_INDEX mode, if the mode of the index (whose index value is B) from a previously scanned position is COPY_INDEX mode, A is always not equal to B. If the mode of the index (whose index value is C) from a previously scanned position is COPY_ABOVE mode, A is always not equal to C. So the index value using (A-1) can be encoded instead of A. The run value is encoded using (run-1) instead of run as it is always greater than zero.

[0149] When palette size is greater than zero, the content of index map are the palette indices which have non-negative values. When palette size equals to zero, the content of index map are quantized value of coefficients which can be negative values.

[0150] When palette size equals to zero, the scanning order and encoding of index map are identical, the only difference is in COPY_INDEX mode, instead of encoding the non-negative palette index, the absolute value of the quantized value of coefficient is encoded, followed by the sign of the quantized value of coefficient if it is a non-zero value. In another embodiment, the sign of the quantized value of coefficient is encoded first, followed by the absolute value of the quantized value of coefficient.

[0151] A second method is the Run length method. This method is a subset of string copy method where COPY_INDEX mode is the only available mode. The coding of mode identifier can be skipped as it can be inferred. For each run string, the index value is encoded using (val-1) instead of val as indicated in previous section, the run value is also encoded using (run-1) instead of run. In one embodiment, the run value in the run string can be forced to one, under this condition, the indexes are encoded one by one. This mode is efficient when

majority of run values are small. When palette size is greater than zero, the content of index map are palette indices which are non-negative values. When palette size equals to zero, the content of index map are quantized values of the coefficients which can be negative values.

[0152] When palette size equals to zero, the scanning order and encoding of the index map are identical, the only difference is that, instead of encoding the non-negative palette index, the absolute value of the quantized value of coefficient is encoded, followed by the sign of the quantized value of coefficient, when the coefficient is a non-zero value. In another embodiment, the sign of the quantized value of coefficient is encoded first, followed by the absolute value of the quantized value of coefficient.

[0153] A third method is the layered sparse bitmap method. Because of the way the coefficients are quantized and dequantized, a value of zero indicates that the dequantized/quantized value of this coefficient is also zero. Depending on the distribution of coefficients in the CU, the quantized value of the coefficient represented by palette index zero may not always be zero. A pal_zero=N indicates that the quantized value represented by palette index N is zero, N is set to -1 if the palette table does not contain a quantized value of zero.

[0154] For each CU, a sparse bitmap is defined to represent sparse structure of the CU where value zero indicates that the value of index equals to pal_zero, value one indicates that the value of index does not equal to pal_zero. As illustrated in FIG. 11, a layered quad-tree block structure is generated using sparse bitmap as bottom layer 1102 (layer 0) until there is only one node in top layer 1104. The value of a given node in layer n+1 is the max value of 2x2 nodes in layer n according to equations (7) and (8).

Value(layer(n+1), y, x) = max(
value(layer(n), 2y, 2x), layer(n), 2y, 2x+1), layer(n), 2y+1, 2x), layer(n), 2y+1, 2x+1)) (7)

Value(layer1, y, x) =
max(value(layer0, 2y, 2x), layer0, 2y, 2x+1), layer0, 2y+1, 2x), layer0, 2y+1, 2x+1)) (8)

[0155] In one embodiment, the connection between all neighbor layers are quad-tree structure, as illustrated in FIG. 11. In another embodiment, the

connection between neighbor layers are quad-tree structure, as illustrated in FIG. 10, except that the connection between layer 1 and layer 0 is row-wise structure, as illustrated in FIG. 12, as described by equations (9)-(12).

Value(layer1, y, x) = max(
value(layer0, 2y, 2x), layer0, 2y, 2x+1), layer0, 2y, 2x+2), layer0, 2y, 2x+3)) (9)

Value(layer1, y, x+1) = max(
value(layer0, 2y+1, 2x), layer0, 2y+1, 2x+1), layer0, 2y+1, 2x+2), layer0, 2y+1, 2x+3)) (10)

Value(layer1, y+1, x) = max(
value(layer0, 2y+2, 2x), layer0, 2y+2, 2x+1), layer0, 2y+2, 2x+2), layer0, 2y+2, 2x+3)) (11)

Value(layer1, y+1, x+1) = max(
value(layer0, 2y+3, 2x), layer0, 2y+3, 2x+1), layer0, 2y+3, 2x+2), layer0, 2y+3, 2x+3)) (12)

[0156] In another method, layered sparse bitmap has an arbitrarily defined layered structure to efficiently represent original sparse bitmap. These layers are scanned using depth-wise quad-tree scan order, starting from top node. The scan order of 4 child nodes is (top-left, top-right, bottom-left and bottom right). If the row-wise connection structure between layer 1 and layer 0 shown in FIG. 12 is utilized, then the scan order of 4 child nodes (in layer 0) is (left, middle-left, middle-right, right).

[0157] While the layered bitmap is always scanned from top layer to bottom layer, the encoding procedure can start from any layer. The encoding of layers above the encoding-start-layer are skipped. In one embodiment, the index of encoding-start-layer is treated as syntax element and stored in syntax table. In another embodiment, the delta index (number of total-layer – index of encoding-start-layer - 1) of encoding-start-layer is treated as syntax element and stored in syntax table.

[0158] The bitmap value of the node in current scan position is encoded first. When the bitmap value of the node equals to zero, the bitmap value of the child nodes are all zeroes so that the scanning and encoding of these child nodes are skipped, when the bitmap value of the node equals to one, the bitmap value of at least one of the child nodes is one so that the method traverses to its child nodes using the predefined scan order. If all other child nodes have bitmap values of zero except for the last child node, even after traversing to last child node the method

continues to traverse to the child nodes of this last node, encoding the bitmap value of this node is skipped as it can be inferred (always one).

[0159] The layered bitmap method is not used in encoding-start-layer as decoder does not have knowledge of layers above encoding-start-layer. When the current layer is bottom layer, after the bitmap value of the node is encoded or skipped, the (index value – 1) of the node is encoded to bitstream. When palette size is greater than zero, the index map contains palette indices which are non-negative values. When palette size equals to zero, the index map contains quantized value of coefficients which can be negative. When palette size equals to zero, the scanning order and encoding procedure are identical, the only difference is that, instead of encoding the non-negative palette indices at the bottom layer, the sign of the quantized value is encoded followed by (absolute value of the quantized value – 1).

[0160] A fourth method is the layered bitmap method. In this method, the bitmap value of the nodes is either zero or one, where value of zero indicates that the value of index equals to pal_zero and a value of one indicates that the value of index does not equal to pal_zero. In one embodiment, the bitmap value of a given node in layer n indicates the max value of 2x2 nodes in layer n-1. In another embodiment, the bitmap value of a given node in layer n indicates the max value of 2x2 nodes in layer n-1, and the bitmap value of a given node in layer 1 indicates the max value of 1x4 nodes in layer 0.

[0161] In layered index map method, the value of the node is actual index value, because of the way coefficients are quantized and dequantized, the index value of zero indicates that the quantized/dequantized value represented by given index is also zero. In one embodiment, the value of a given node in layer n indicates the max value of 2x2 nodes in layer n-1. In another embodiment, the value of a given node in layer n indicates the max value of 2x2 nodes in layer n-1, and the value of a given node in layer 1 indicates the max value of 1x4 nodes in layer 0.

[0162] The layered index map method shares the identical partitioning and scanning order with layered bitmap method. While the layered index map is always scanned from top layer to bottom layer, the encoding procedure can also start from any layer. The encoding of layers above the encoding-start-layer are

skipped, and the index or delta index of encoding-start-layer is also treated as syntax element and stored in syntax table.

[0163] The absolute value of the node in current scan position is encoded when this node is at encoding-start-layer, the delta value (absolute value of parent node – absolute value of current node) of the node in current scan position is encoded when this node is not at the encoding-start-layer. If the node value equals to zero, it indicates that the value of its child nodes are all zeroes so that the scanning and encoding of these child nodes are skipped, if the node value does not equal to zero, it indicates that the value of at least one of the child nodes is non-zero so that traversing to its child nodes uses the predefined scan order. If all other child nodes have absolute values smaller than the absolute value of their parent node except for the last child node, even though traversing goes to the last child node and continues travel to its child nodes, encoding the absolute value of this node is skipped as it can be inferred (always the absolute value of its parent node). The node-skipping method, described above, is not used in encoding-start-layer as decoder does not have knowledge of layers above encoding-start-layer.

[0164] When palette size is greater than zero, the content of index map are palette indices which are non-negative values. When palette size equals to zero, the content of index map are quantized value of coefficients which can be negative. When palette size equals to zero, the scanning order and encoding procedure are identical, the only difference is that, when the current layer is bottom layer and the absolute value of current node is not zero, after the absolute value or delta value of the node is encoded or skipped, the sign of the node is encoded to the bitstream.

[0165] A fifth method is the escape coding method. An escape index is defined as the index value equals to palette table size, the escape value is defined as the quantized value of the coefficient whose index value is escape value.

Row 0	1	2	3	4	5	6	7	8
Row 1	9	2	3	4	5	6	7	8
Row 2	9	2	4	4	4	4	4	..
Row 3	...	..	..	..	..	..	..	..
Row 4	...	..	..	..	..	..	..	..
Row 5	...	..	..	..	..	..	..	..
Row 6	...	..	..	..	..	..	..	..
Row 7	...	..	..	..	..	..	..	..

Table XIV

[0166] Table XIV shows an example 8x8 CU. When the palette size is 9, this CU has two escape indices (at location [1,0] and [2,0]). These escape values are encoded to the bitstream.

[0167] In one embodiment, escape coding is processed after index map coding. In another embodiment, escape coding is processed before index map coding so that the decoding process can output the quantized value of the coefficient to the inference engine during the decoding of index map instead of waiting until the decoding of index map is completed. This enables the use of a software or hardware pipeline between decoding and inference. The number of escape index is treated as syntax element and stored in syntax table in this embodiment.

[0168] During escape coding, an offset value is defined for this CU, the sign bit of the escape value is encoded first, followed by the encoding of the sign of delta escape (absolute value of escape value - offset), followed by the encoding of the absolute value of the delta escape. In one embodiment, this offset is identical across all CUs. In another embodiment, this offset is different among some or all CUs. In another embodiment, this offset is explicitly signaled as syntax element. In another embodiment, this offset is inferred implicitly so that decoder can regenerate it using the statistics.

[0169] FIG. 13 illustrates an embodiment of a neural network training apparatus 1300. Input feature map is processed by number of neural layers unit 1302, 1304, and 1310 to generate inference result. The result is compared with ground truth, and the error is back propagated (BP) through these layers so that the weights of each layer can be updated.

[0170] Inside each layer unit, updated weight 1312 is processed by sparse unit 1314 and column swap unit 1316 to generate non-zero coefficients, layered sparse bitmap and swapped column indices. Non-zero coefficients 1318 are further processed by quantization, k-mean clustering, palette sharing, and progressive coding unit 1324 to generate quantized non-zero coefficients. This output, together with layered sparse bitmap 1320 and swapped column indices 1322, are encoded by entropy encoder unit 1326 to generate compressed weight.

[0171] Quantized non-zero coefficients are further processed by non-zero coefficient reconstruction unit 1328 to generate reconstructed non-zero

coefficients. This output, together with layered sparse bitmap 1320 and swapped column indices 1322, are sent to matrix multiplication GEBP unit 1350 as weight input, matrix multiplication GEBP unit 1350 also takes original uncompressed input feature map or reconstructed input feature map as input feature map input, and generate uncompressed output feature map (as uncompressed input feature map for next layer unit).

[0172] An output feature map compression unit 1352 generates the compressed output feature map (as compressed input feature map for next layer unit). Uncompressed output feature map is processed by relu, max pooling unit 1330, sparse unit, 1332 and column swap unit 1334 to generate non-zero coefficients 1336, layered sparse bitmap 1338 and swapped column indices 1340. Non-zero coefficients are further processed by quantization, k-mean clustering and palette sharing unit 1342 to generate quantized non-zero coefficients. A combiner unit 1344 combines the quantized non-zero coefficients, layered sparse bitmap and swapped column indices to generate the compressed output feature map (as compressed input feature map for next layer unit).

[0173] An RDS optimizer (RDSO) unit 1348 is used to perform joint rate-distortion-speed optimization. Depending on the underlying RDS algorithm, this unit may take part or all of following signals as input: original weight, reconstructed weight, compressed weight, uncompressed output feature map, compressed feature map, and original output feature map as input. An additional matrix multiplication GEBP unit 1346 is used to take original input feature map and uncompressed weight as input, and generate original output feature map.

[0174] FIG. 14 illustrates an embodiment of a neural network inference apparatus 1400. Input feature map is processed by the neural layer units 1302, 1304, and 1310 to generate inference result.

[0175] Inside each layer unit, entropy decoder unit 1402 decodes the received compressed weight to quantized coefficients, layered sparse bitmap 1408 and swapped column indices 1410. Quantized coefficients are further processed by palette restoration, coefficient decoding and de-quantization unit 1404 to generate reconstructed non-zero coefficients 1406. Matrix multiplication GEBP unit 1412 takes the reconstructed non-zero coefficients 1406, layered sparse bitmap, 1408 and swapped column indices 1410 as weight input, it also takes original uncompressed input feature map or reconstructed input feature map as

input feature map input, and generates uncompressed output feature map (as uncompressed input feature map for next layer unit).

[0176] As described above, an output feature map compression unit 1352 can also be presented to generate the compressed output feature map (as compressed input feature map for next layer unit).

[0177] FIG. 15 illustrates an embodiment of an apparatus 1500 for decoding of compressed input feature map. Entropy decoder unit 1502 decodes the received compressed input feature map to quantized coefficients, layered sparse bitmap 1508 and swapped column indices 1510. Quantized coefficients are further processed by palette restoration, coefficient decoding and de-quantization unit 1504 to generate reconstructed non-zero coefficients1506, a combiner unit 1512 combines the reconstructed non-zero coefficients, layered sparse bitmap and swapped column indices to generate the reconstructed input feature map.

[0178] FIG. 16 illustrates an embodiment of a network unit apparatus 1600 that includes above mentioned neural network training or inference unit or processor 1610 that processes input feature map as described above, for example, within a network or system. The network unit may comprise a plurality of ingress ports 1602, 1604, and 1606 and/or receiver units (Rx) 1608 for receiving data from other network units or components, and providing the received data to the neural network training or inference unit or processor 1610 to process data and determine the inference result, and a plurality of egress ports 1614, 1616, and 1618 and/or transmitter units (Tx) 1612 for transmitting result to the other network units. The neural network training or inference unit or processor 1610 may be configured to implement either training or inference schemes described herein, such as encoding and decoding weight and/or input feature map using sparse, column swap and palette sharing concept described above. The neural network training or inference unit or processor 1610 may be implemented using hardware, software, or both.

[0179] FIG. 17 is a block diagram of a computing device 1700, according to an embodiment. Similar components may be used in the example computing devices described herein. For example, the clients, servers, and network resources may each use a different set of the components shown in FIG. 17 and/or computing components not shown in FIG. 17. Computing devices similar to computing device 1700 may be used to implement computing devices 102(1) – 102(N) and 104(1) – 104(N) shown in FIG. 1; the training system 200,

shown in FIG. 2; the neural network training apparatus 1300, shown in FIG. 13; the neural network inference apparatus 1400, shown in FIG. 14; the decoding apparatus 1500 shown in FIG. 15; and the network unit apparatus 1600 shown in FIG. 16.

[0180] One example computing device 1700 may include a processing unit (e.g., one or more processors and/or CPUs) 1702, memory 1703, removable storage 1710, and non-removable storage 1712 communicatively coupled by a bus 1701. Although the various data storage elements are illustrated as part of the computing device 1700.

[0181] Memory 1703 may include volatile memory 1714 and non-volatile memory 1708. Computing device 1700 may include or have access to a computing environment that includes a variety of computer-readable media, such as volatile memory 1714 and non-volatile memory 1708, removable storage 1710 and non-removable storage 1712. Computer storage includes random access memory (RAM), read only memory (ROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), flash memory or other memory technologies, compact disc read-only memory (CD-ROM), digital versatile disk (DVD) or other optical disk storage devices, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium capable of storing computer-readable instructions. The memory 1703 also includes program instructions for applications 1718 that implement any of the methods and/or algorithms described above.

[0182] Computing device 1700 may include or have access to a computing environment that includes input interface 1706, output interface 1704, and communication interface 1716. Output interface 1704 may provide an interface to a display device, such as a touchscreen, that also may serve as an input device. The input interface 1706 may provide an interface to one or more of a touchscreen, touchpad, mouse, keyboard, camera, one or more device-specific buttons, one or more sensors integrated within or coupled via wired or wireless data connections to the server computing device 1700, and/or other input devices. The computing device 1700 may operate in a networked environment using a communication interface 1716. The communication interface may include one or

more of an interface to a local area network (LAN), a wide area network (WAN), a cellular network, a WLAN network, and/or a Bluetooth® network.

[0183] Any one or more of the modules described herein may be implemented using hardware (e.g., a processor of a machine, an application-specific integrated circuit (ASIC), field-programmable gate array (FPGA), or any suitable combination thereof). Moreover, any two or more of these modules may be combined into a single module, and the functions described herein for a single module may be subdivided among multiple modules. Furthermore, according to various example embodiments, modules described herein as being implemented within a single machine, database, or device may be distributed across multiple machines, databases, or devices. As described herein, a module can comprise one or both of hardware or software that has been designed to perform a function or functions (e.g., one or more of the functions described herein in connection with providing secure and accountable data access).

[0184] Although a few embodiments have been described in detail above, other modifications are possible. For example, the logic flows depicted in the FIGs. do not require the particular order shown, or sequential order, to achieve desirable results. Other steps may be provided, or steps may be eliminated, from the described flows, and other components may be added to, or removed from, the described systems. Other embodiments may be within the scope of the following claims.

[0185] It should be further understood that software including one or more computer-executable instructions that facilitate processing and operations as described above with reference to any one or all of the steps of the disclosure can be installed in and provided with one or more computing devices consistent with the disclosure. Alternatively, the software can be obtained and loaded into one or more computing devices, including obtaining the software through physical medium or distribution system, including, for example, from a server owned by the software creator or from a server not owned but used by the software creator. The software can be stored on a server for distribution over the Internet, for example.

[0186] Also, it will be understood by one skilled in the art that this disclosure is not limited in its application to the details of construction and the arrangement of components set forth in the description or illustrated in the

drawings. The embodiments herein are capable of other embodiments and capable of being practiced or carried out in various ways. Also, it will be understood that the phraseology and terminology used herein is for the purpose of description and should not be regarded as limiting. The use of "including," "comprising," or "having" and variations thereof herein is meant to encompass the items listed thereafter and equivalents thereof as well as additional items. Unless limited otherwise, the terms "connected," "coupled," and "mounted," and variations thereof herein are used broadly and encompass direct and indirect connections, couplings, and mountings. In addition, the terms "connected" and "coupled" and variations thereof are not restricted to physical or mechanical connections or couplings.

[0187] The components of the illustrative devices, systems, and methods employed in accordance with the illustrated embodiments can be implemented, at least in part, in digital electronic circuitry or in computer hardware, firmware, software, or in combinations of them. These components can be implemented, for example, as a computer program product such as a computer program, program code or computer instructions tangibly embodied in an information carrier, or in a machine-readable storage device, for execution by, or to control the operation of, data processing apparatus such as a programmable processor, a computer, or multiple computers.

[0188] A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, method, object, or another unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network. Method steps associated with the illustrative embodiments can be performed by one or more programmable processors executing a computer program, code, or instructions to perform functions (e.g., by operating on input data and/or generating an output). Method steps can also be performed by, and apparatus for performing the methods can be implemented as, special purpose logic circuitry, for example, as an FPGA (field-programmable gate array) or an ASIC (application-specific integrated circuit), for example.

[0189] The various illustrative logical blocks, modules, and circuits described in connection with the embodiments disclosed herein may be implemented or performed with a general-purpose processor, a digital signal processor (DSP), an ASIC, a FPGA or other programmable logic device, discrete gate or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein. A general-purpose processor may be a single core or multi-core microprocessor, but in the alternative, the processor may be any conventional processor, controller, microcontroller, or state machine. A processor may also be implemented as a combination of computing devices, for example, a combination of a DSP and a microprocessor, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other such configuration.

[0190] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random-access memory or both. The elements of a computer include a processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example, semiconductor memory devices, for example, electrically programmable read-only memory or ROM (EPROM), electrically erasable programmable ROM (EEPROM), flash memory devices, and data storage disks (e.g., magnetic disks, internal hard disks, or removable disks, magneto-optical disks, and CD-ROM and DVD-ROM disks). The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

[0191] Those of skill in the art understand that information and signals may be represented using any of a variety of different technologies and techniques. For example, data, instructions, commands, information, signals, bits, symbols, and chips that may be referenced throughout the above description may be

represented by voltages, currents, electromagnetic waves, magnetic fields or particles, optical fields or particles, or any combination thereof.

[0192] As used herein, "machine-readable medium" (or "computer-readable medium") means a device able to store instructions and data temporarily or permanently and may include, but is not limited to, random-access memory (RAM), read-only memory (ROM), buffer memory, flash memory, optical media, magnetic media, cache memory, other types of storage (e.g., Erasable Programmable Read-Only Memory (EEPROM)), and/or any suitable combination thereof. The term "machine-readable medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, or associated caches and servers) able to store processor instructions. A machine-readable medium or computer-readable medium shall also be taken to include any medium (or a combination of multiple media) that is capable of storing instructions for execution by one or more processors, such that the instructions, when executed by one or more processors, cause the one or more processors to perform any one or more of the methodologies described herein. Accordingly, a machine-readable medium or computer-readable medium refers to a single storage apparatus or device, as well as "cloud-based" storage systems or storage networks that include multiple storage apparatus or devices. The term "machine-readable medium" as used herein excludes signals per se.

[0193] In addition, techniques, systems, subsystems, and methods described and illustrated in the various embodiments as discrete or separate may be combined or integrated with other systems, modules, techniques, or methods without departing from the scope of the present disclosure. Other items shown or discussed as coupled or directly coupled or communicating with each other may be indirectly coupled or communicating through some interface, device, or intermediate component whether electrically, mechanically, or otherwise. Other examples of changes, substitutions, and alterations are ascertainable by one skilled in the art and could be made without departing from the scope disclosed herein.

[0194] Although the present disclosure has been described with reference to specific features and embodiments thereof, it is evident that various modifications and combinations can be made thereto without departing from the scope of the disclosure. For example, other components may be added to, or removed from, the described methods, modules, devices, and/or systems. The

specification and drawings are, accordingly, to be regarded simply as an illustration of the disclosure as defined by the appended claims, and are contemplated to cover any and all modifications, variations, combinations or equivalents that fall within the scope of the present disclosure. Other aspects may be within the scope of the following claims.

CLAIMS

What is claimed is:

1. A computer-implemented method of generating a compressed representation of a neural network, comprising:

obtaining a weight tensor and an input feature map for the neural network;

reordering the weight tensor into blocks compatible with a matrix multiplication operation;

compressing the reordered weight tensor to generate a compressed weight tensor;

quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor;

multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map; and

compressing the output feature map to provide a compressed output feature map.

2. The computer-implemented method of claim 1, wherein:

the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients;

the reordering of the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients; and

the compressing of the reordered weight tensor includes:

generating a map of the swapped columns of the reordered weight tensor; and

quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

3. The computer-implemented method of claim 2, wherein the compressing of the reordered weight tensor further comprises entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

4. The computer-implemented method of any of claims 1 to 3, wherein:

the output feature map includes zero-valued feature values and non-zero valued feature values;

the reordering of the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values; and

the compressing of the reordered output feature map includes:

generating a map of the swapped columns of the output feature map;

quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values; and

combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

5. The computer-implemented method of any of claims 1 to 3, wherein the method is performed by a layer of a multi-layer convolutional neural network (CNN) and the method further comprises:

obtaining the input feature map as the output feature map from a previous layer of the CNN;

obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN; and

providing the output feature map as the input feature map to the subsequent layer of the multi-layer CNN.

6. The computer implemented method of claim 5, wherein the method further comprises:

decompressing a compressed weight tensor from a previous layer of the multi-layer CNN;

multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map;

compressing the inference output feature map to provide an inference compressed output feature map; and

determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

7. An apparatus for generating a compressed representation of a neural network, comprising:

a memory including program instructions; and

a processor, coupled to the memory and conditioned, in response to the program instructions, to perform operations including:

obtaining a weight tensor and an input feature map for the neural network;

reordering the weight tensor into blocks compatible with a matrix multiplication operation;

compressing the reordered weight tensor to generate a compressed weight tensor;

quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor;

multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map; and

compressing the output feature map to provide a compressed output feature map.

8. The apparatus of claim 7, wherein:

the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients;

the operation of reordering the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients; and

the operation of compressing the reordered weight tensor includes:

generating a map of the swapped columns of the reordered weight tensor; and

quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

9. The apparatus of claim 8, wherein the operation of compressing the reordered weight tensor further comprises entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

10. The apparatus of any of claims 7 to 9, wherein:

the output feature map includes zero-valued feature values and non-zero valued feature values;

the operation of reordering the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values; and

the operation of compressing the reordered output feature map includes:

generating a map of the swapped columns of the output feature map;

quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values; and

combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

11. The apparatus of any of claims 7 to 9, wherein the processor and the program instructions implement a layer of a multi-layer convolutional neural network (CNN) and the operations further comprise:

obtaining the input feature map as the output feature map from a previous layer of the CNN;

obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN; and

providing the output feature map as the input feature map to the subsequent layer of the multi-layer CNN.

12. The apparatus of claim 11, wherein the operations further comprise:

decompressing a compressed weight tensor from a previous layer of the multi-layer CNN;

multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map;

compressing the inference output feature map to provide an inference compressed output feature map; and

determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

13. A computer readable medium including program instructions that configure a computer processing system to perform operations to generate a compressed representation of a neural network, the operations comprising:

obtaining a weight tensor and an input feature map for the neural network;

reordering the weight tensor into blocks compatible with a matrix multiplication operation;

compressing the reordered weight tensor to generate a compressed weight tensor;

quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor;

multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map; and

compressing the output feature map to provide a compressed output feature map.

14. The computer readable medium of claim 13, wherein:

the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients;

the operation of reordering the weight tensor includes swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients; and

the operation of compressing the reordered weight tensor includes:

generating a map of the swapped columns of the reordered weight tensor; and

quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

15. The computer readable medium of claim 14, wherein the operation of compressing the reordered weight tensor further comprises entropy coding the palette of quantized non-zero valued weight coefficients and the map of the swapped columns of the reordered weight tensor.

16. The computer readable medium of any of claims 13 to 15, wherein:

the output feature map includes zero-valued feature values and non-zero valued feature values;

the operation of reordering the output feature map includes swapping columns of the output feature map to increase a number of blocks of the output feature map having the zero-valued feature values; and

the compressing of the reordered output feature map includes:

generating a map of the swapped columns of the output feature map;

quantizing the non-zero valued feature values to provide a palette of quantized non-zero valued feature values; and

combining the map of the swapped columns of the output feature map and the quantized palette of non-zero valued feature values.

17. The computer readable medium of any of claims 13 to 15, wherein the computer processing system includes a layer of a multi-layer convolutional neural network (CNN) and the operations further comprise:

obtaining the input feature map as the output feature map from a previous layer of the CNN;

obtaining the weight tensor by combining the obtained weight tensor with a weight update from a subsequent layer in the multi-layer CNN; and

providing the output feature map as the input feature map to the subsequent layer of the multi-layer CNN.

18. The computer readable medium of claim 17, wherein the operations further comprise:

decompressing a compressed weight tensor from a previous layer of the multi-layer CNN;

multiplying the input feature map by the decompressed weight tensor to provide an inference output feature map;

compressing the inference output feature map to provide an inference compressed output feature map; and

determining an inference result based on the output feature map, the inference output feature map, the compressed output feature map and the inference compressed output feature map.

19. An apparatus for generating a compressed representation of a neural network, comprising:

means for obtaining a weight tensor and an input feature map for the neural network;

means for reordering the weight tensor into blocks compatible with a matrix multiplication operation;

means for compressing the reordered weight tensor to generate a compressed weight tensor;

means for quantizing coefficients of the reordered weight tensor to provide a quantized reordered weight tensor;

means for multiplying the input feature map by the quantized reordered weight tensor to provide an output feature map; and

means for compressing the output feature map to provide a compressed output feature map.

20. The apparatus of claim 19, wherein:

the weight tensor includes zero-valued weight coefficients and non-zero valued weight coefficients and;

the means for reordering the weight tensor includes means for swapping columns of the weight tensor to increase a number of the weight tensor blocks having the zero-valued weight coefficients; and

the means for compressing the reordered weight tensor includes:

means for generating a map of the swapped columns of the reordered weight tensor; and

means for quantizing the non-zero valued weight coefficients to provide a palette of quantized non-zero valued weight coefficients.

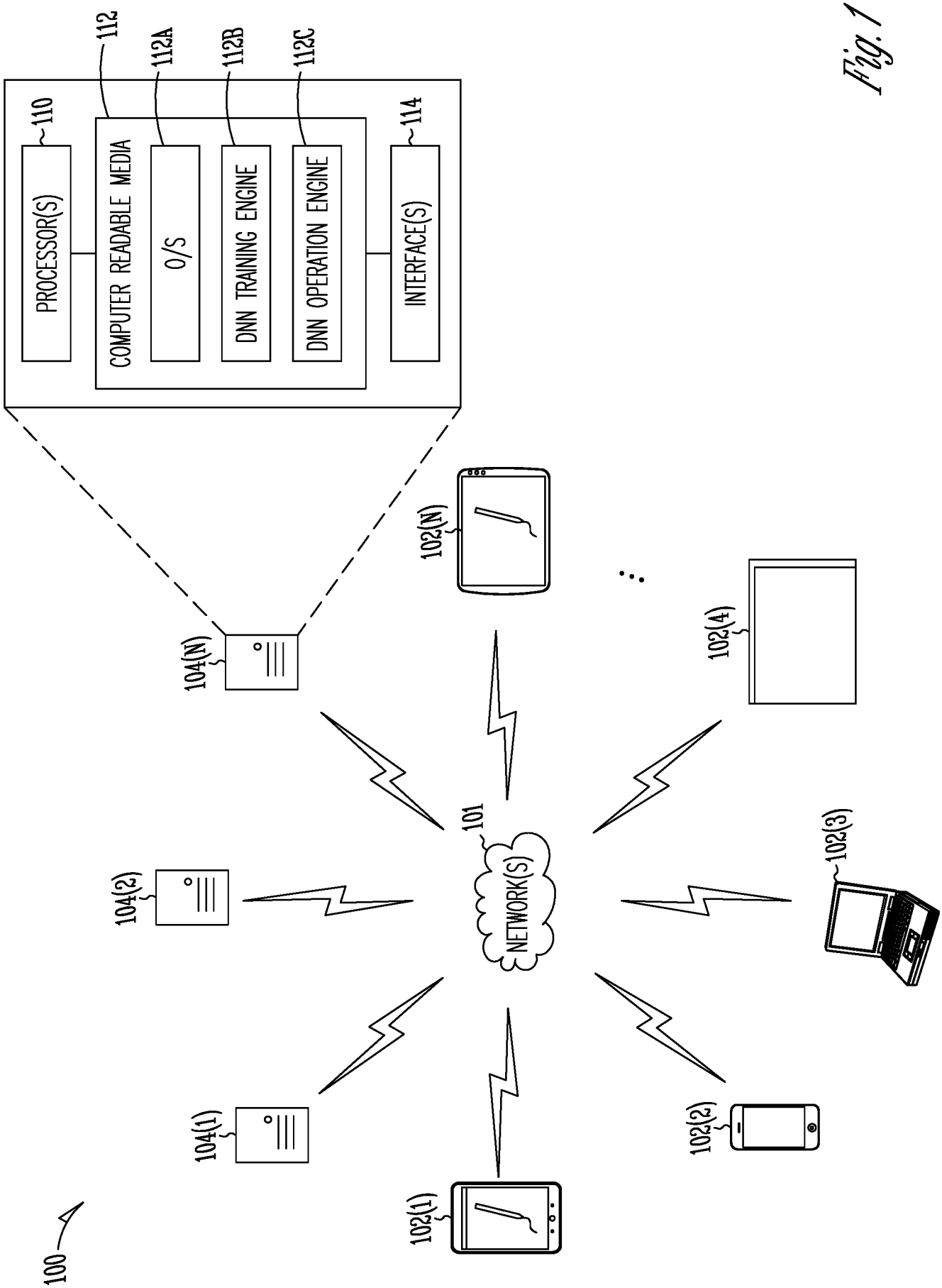
112
112A
112B
112C
110
PROCESSOR(S)
COMPUTER READABLE MEDIA
O/S
DNN TRAINING ENGINE
DNN OPERATION ENGINE
114
INTERFACE(S)
102(N)
104(N)
102(4)
101
NETWORK(S)
104(2)
102(3)
104(1)
102(2)
102(1)
100

Fig. 1

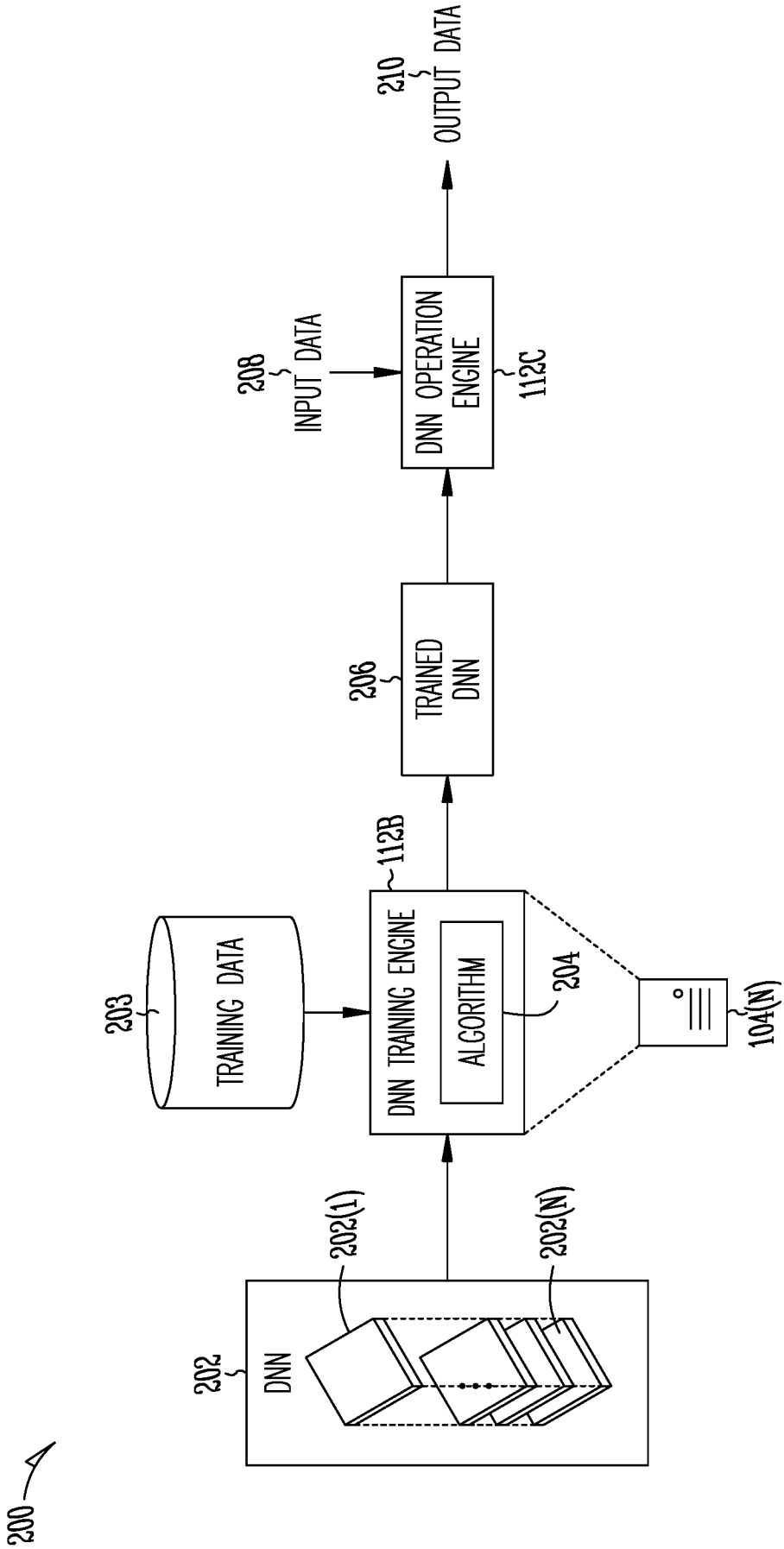
200
DNN
202
202(1)
202(N)
DNN TRAINING ENGINE
ALGORITHM
112B
204
TRAINING DATA
203
104(N)
TRAINED DNN
206
DNN OPERATION ENGINE
112C
INPUT DATA
208
OUTPUT DATA
210

Fig. 2

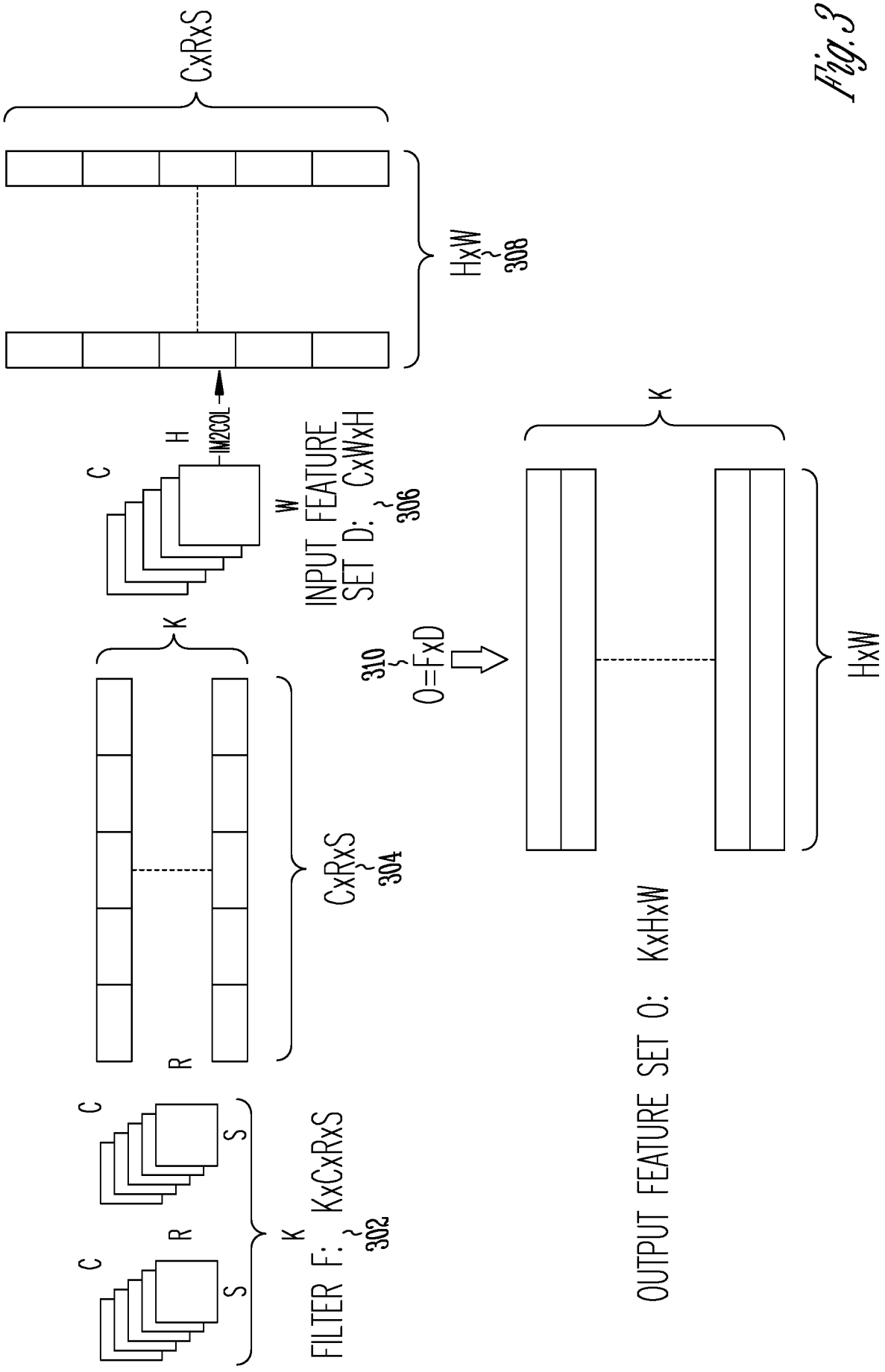
CxRxS
HxW
308
IM2COL
H
W
C
INPUT FEATURE SET D: CxWxH
306
K
CxRxS
304
R
C
S
K
FILTER F: KxCxRxS
302
310
O=FxD
K
HxW
OUTPUT FEATURE SET O: KxHxW

Fig. 3

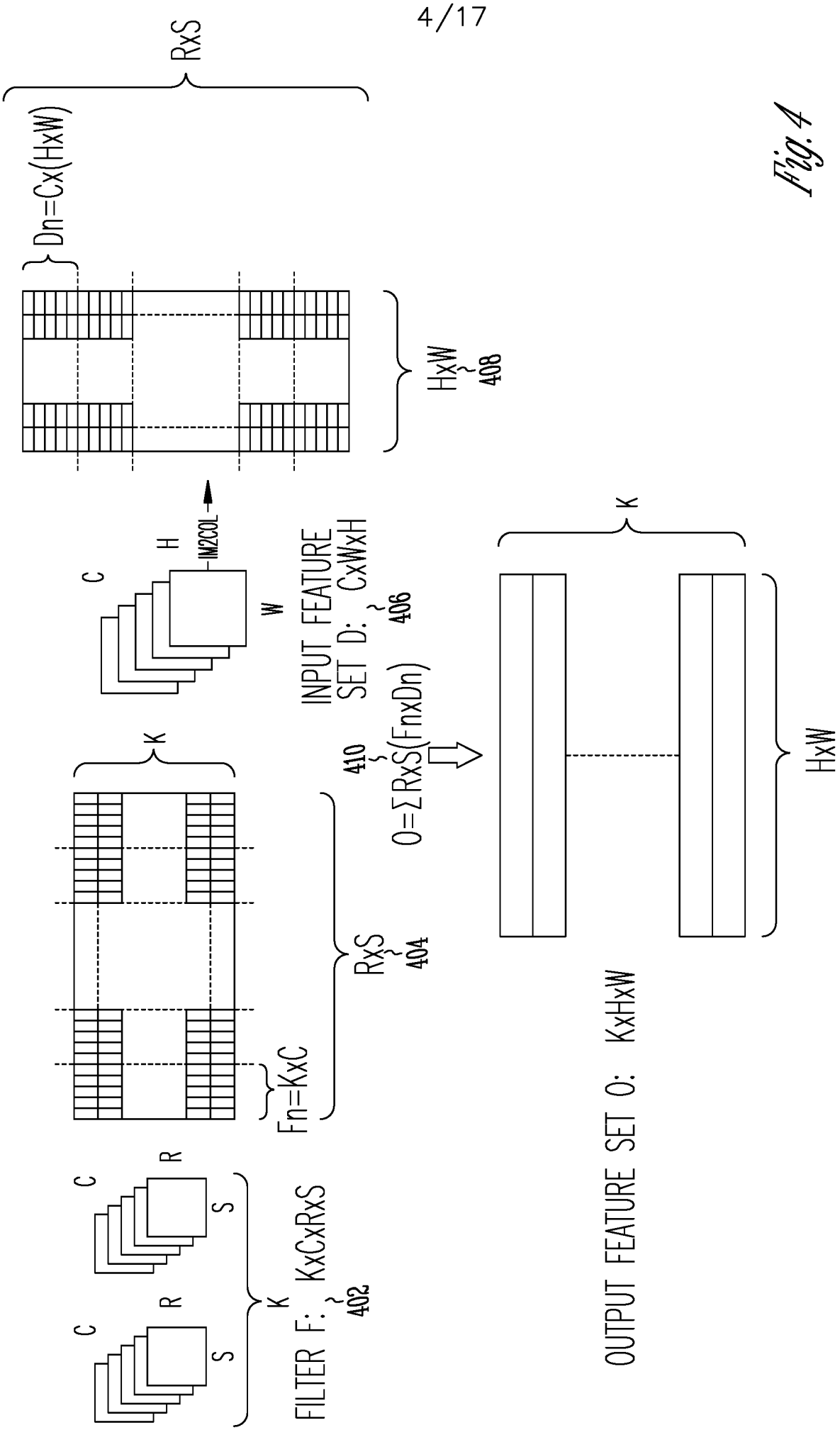
RxS
Dn=Cx(HxW)
HxW
408
IM2COL
H
W
C
INPUT FEATURE SET D: CxWxH
406
410
O=ΣRxS(FnxDn)
K
Fn=KxC
RxS
404
HxW
OUTPUT FEATURE SET O: KxHxW
FILTER F: KxCxRxS
402
R
S

Fig. 4

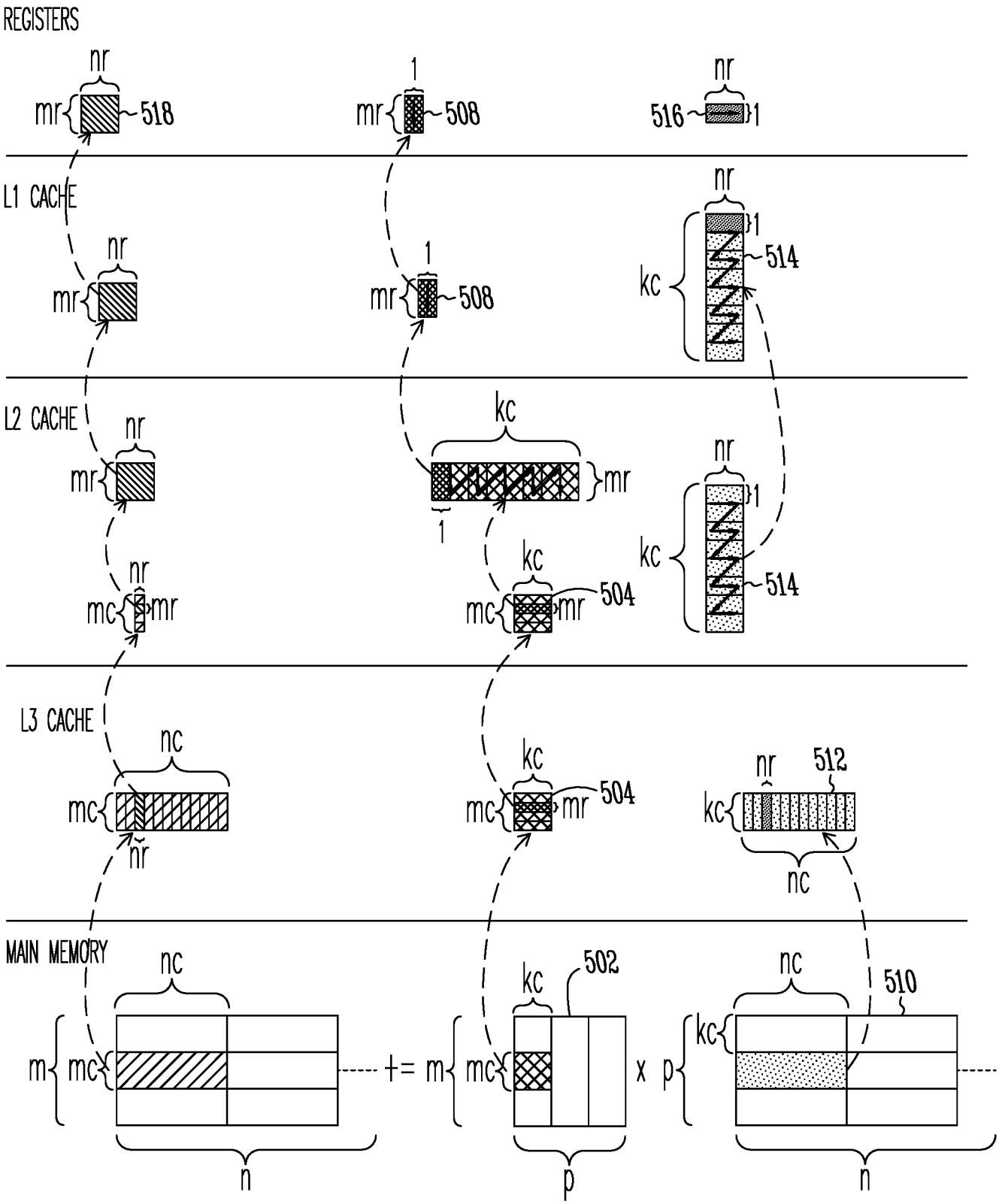
REGISTERS
nr
mr
518
1
mr
508
nr
516
1
L1 CACHE
nr
mr
1
mr
508
nr
1
514
kc
L2 CACHE
nr
mr
kc
mr
1
nr
mc
mr
kc
mc
mr
504
nr
1
kc
514
L3 CACHE
nc
mc
nr
kc
mc
mr
504
nr
512
kc
nc
MAIN MEMORY
nc
m
mc
n
+= m
kc
502
mc
p
x
p
nc
510
kc
n

Fig. 5

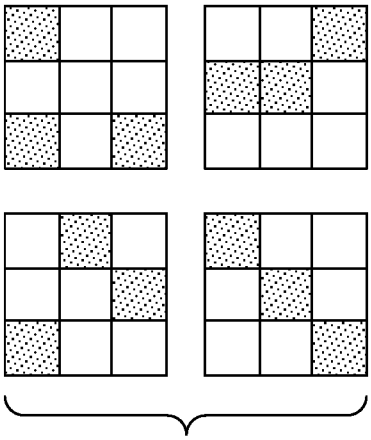

Fig. 6A

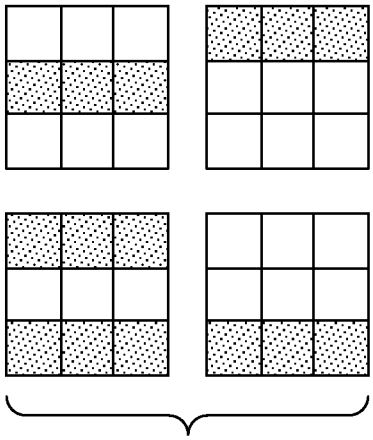

Fig. 6B

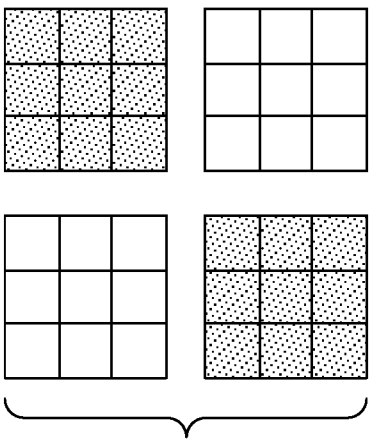

Fig. 6C

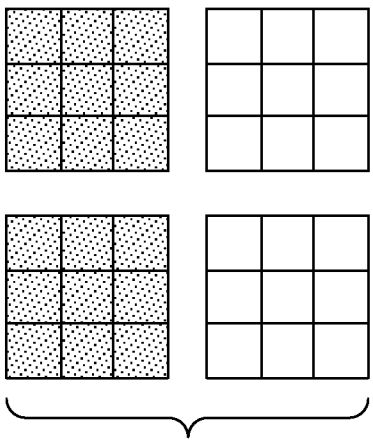

Fig. 6D

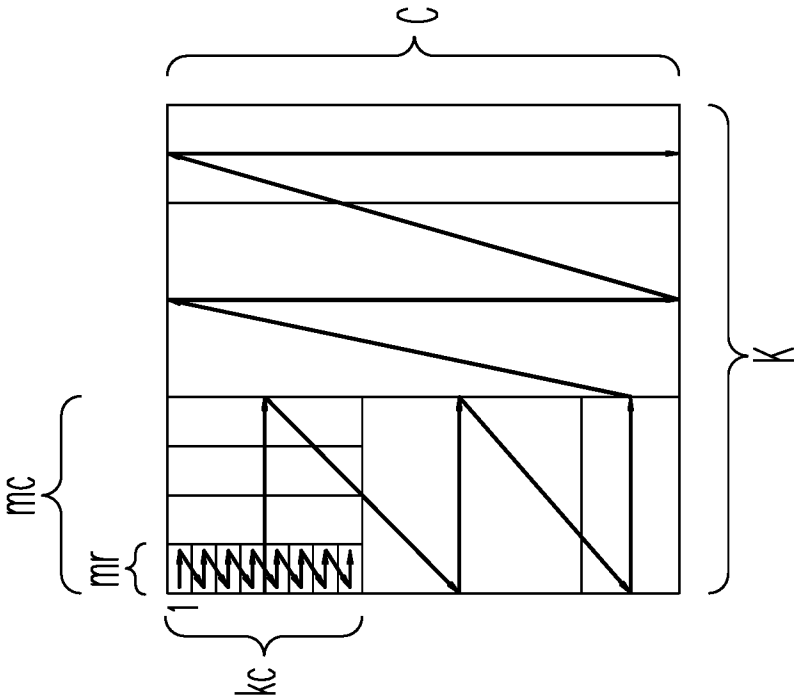

Fig. 7B

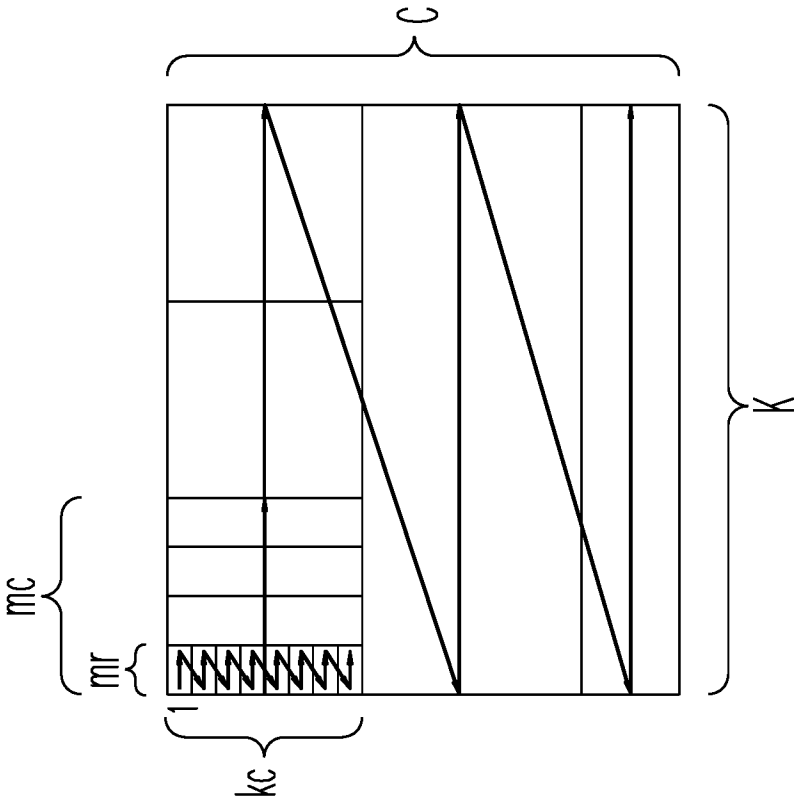

Fig. 7A

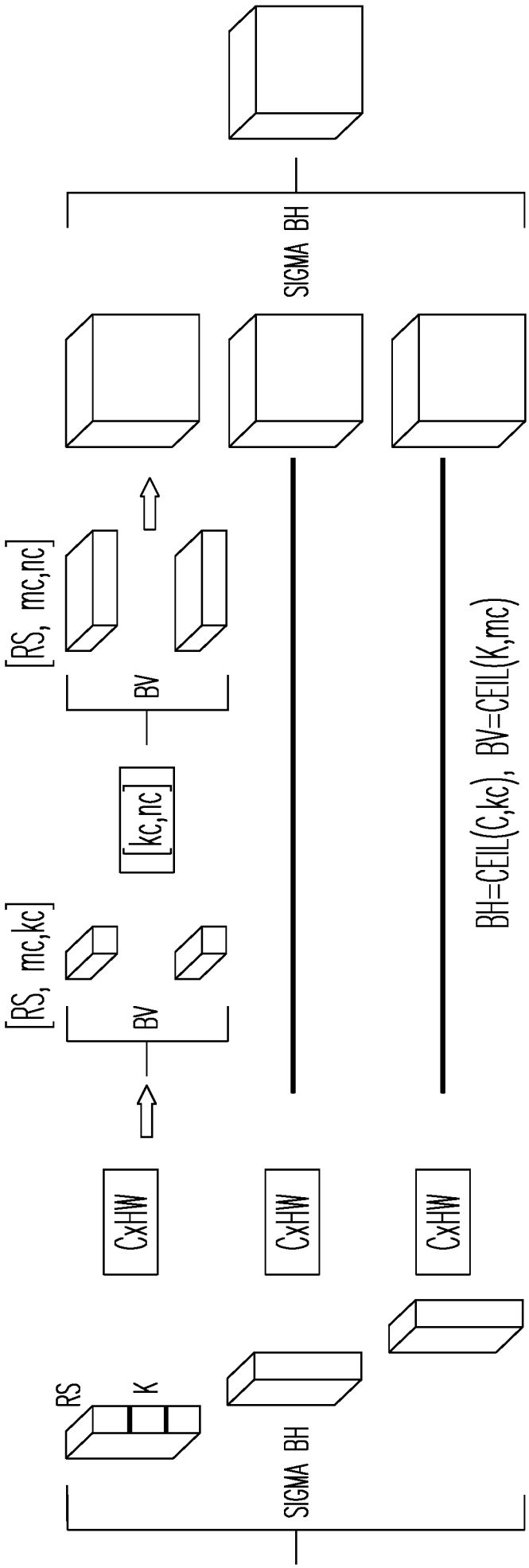
SIGMA BH
[RS, mc,nc]
BV
[kc,nc]
[RS, mc,kc]
BV
BH=CEIL(C,kc), BV=CEIL(K,mc)
CxHW
CxHW
CxHW
RS
K
SIGMA BH

Fig. 8

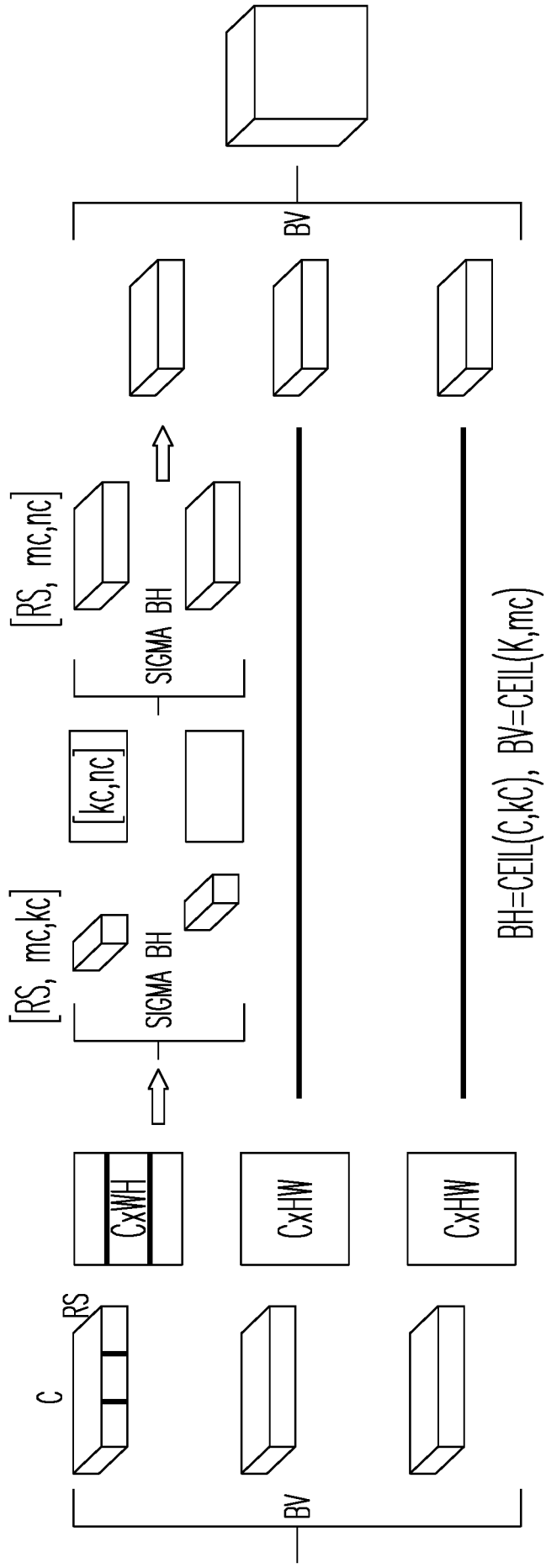
BV
[RS, mc,nc]
SIGMA BH
[kc,nc]
[RS, mc,kc]
SIGMA BH
BH=CEIL(C,kC), BV=CEIL(K,mc)
CxWH
CxHW
CxHW
C
RS
BV

Fig. 9

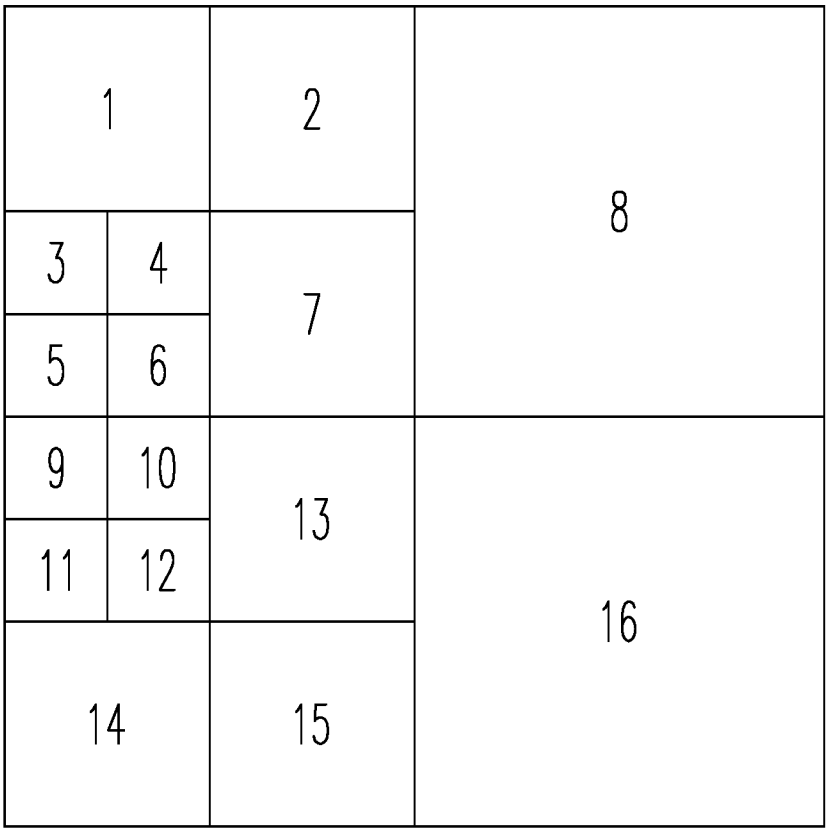

Fig. 10

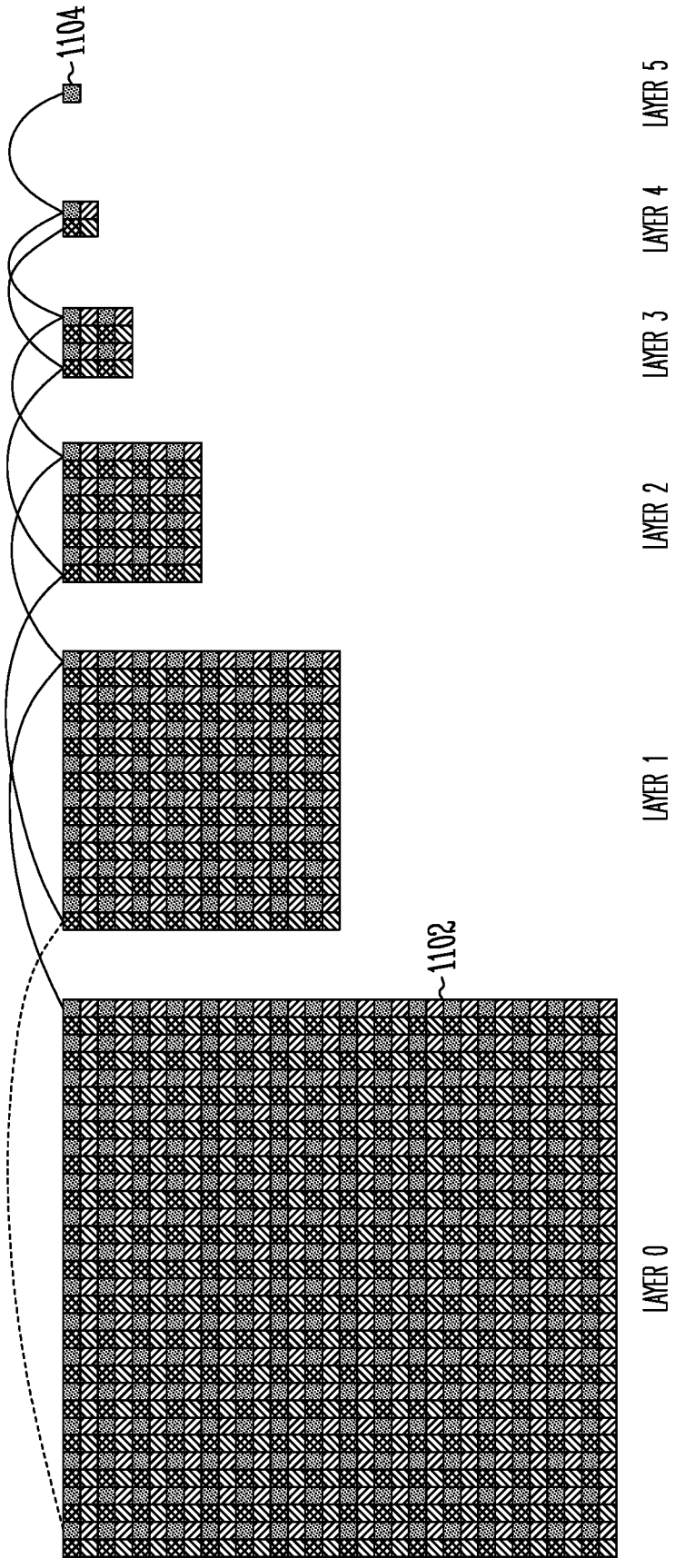

Fig. 11

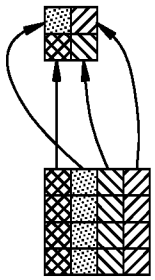

Fig. 12

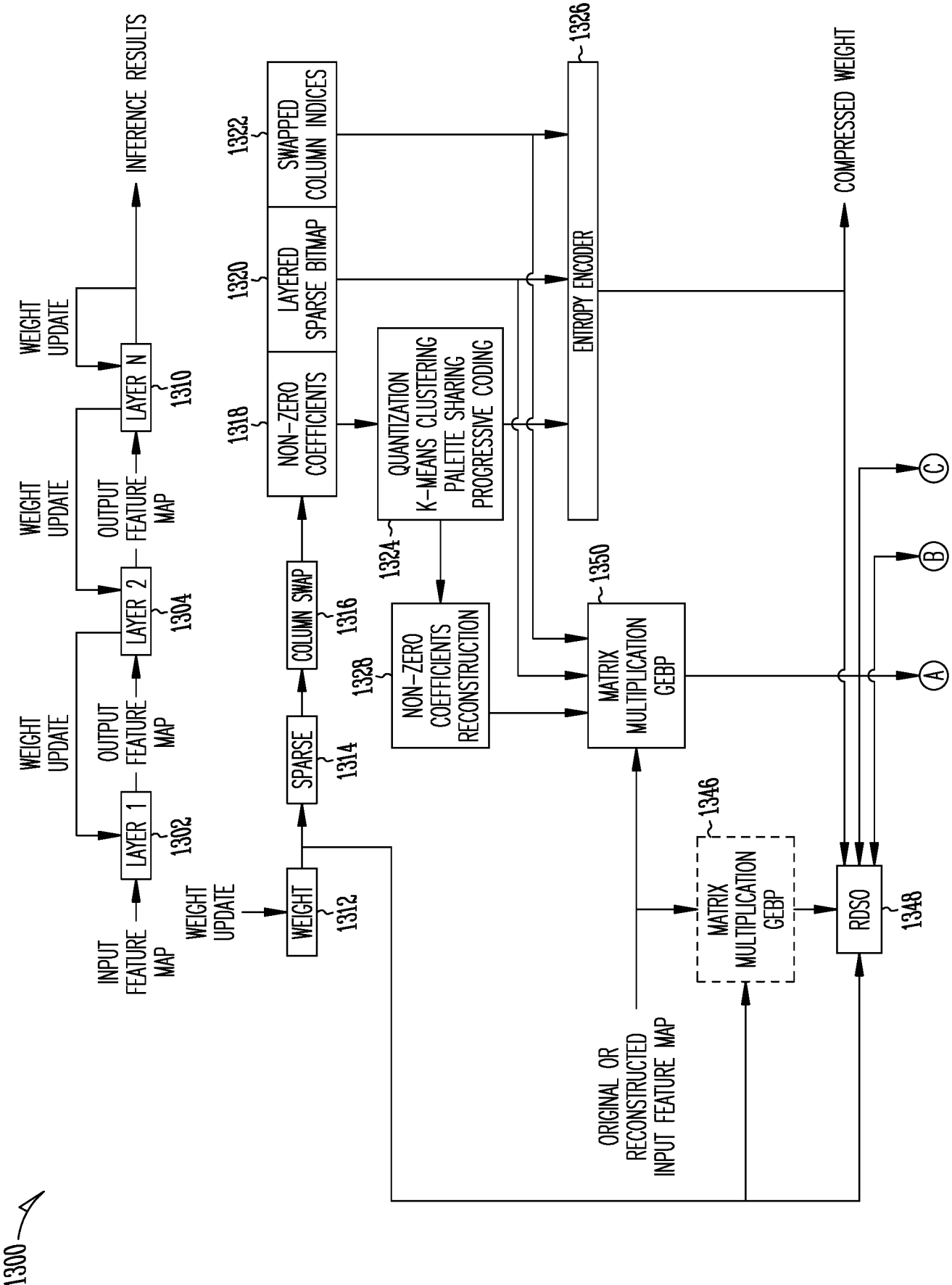
1300
INPUT FEATURE MAP
WEIGHT UPDATE
WEIGHT 1312
LAYER 1 1302
OUTPUT FEATURE MAP
WEIGHT UPDATE
LAYER 2 1304
OUTPUT FEATURE MAP
WEIGHT UPDATE
LAYER N 1310
INFERENCE RESULTS
SPARSE 1314
COLUMN SWAP 1316
NON-ZERO COEFFICIENTS 1318
LAYERED SPARSE BITMAP 1320
SWAPPED COLUMN INDICES 1322
QUANTIZATION K-MEANS CLUSTERING PALETTE SHARING PROGRESSIVE CODING 1324
NON-ZERO COEFFICIENTS RECONSTRUCTION 1328
ENTROPY ENCODER 1326
COMPRESSED WEIGHT
MATRIX MULTIPLICATION GEBP 1350
MATRIX MULTIPLICATION GEBP 1346
RDSO 1348
ORIGINAL OR RECONSTRUCTED INPUT FEATURE MAP
A
B
C

Fig. 13A

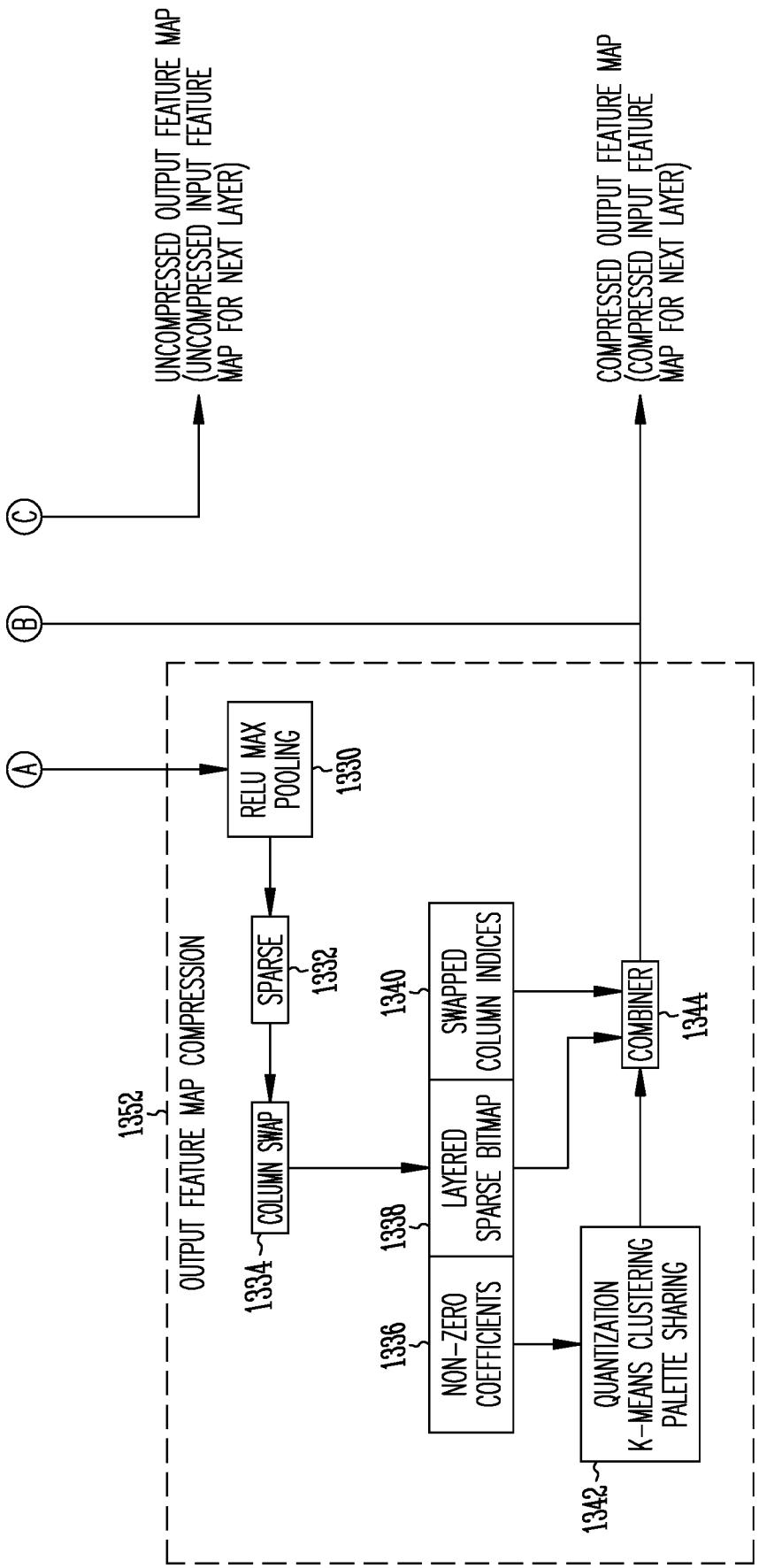
UNCOMPRESSED OUTPUT FEATURE MAP (UNCOMPRESSED INPUT FEATURE MAP FOR NEXT LAYER)
COMPRESSED OUTPUT FEATURE MAP (COMPRESSED INPUT FEATURE MAP FOR NEXT LAYER)
C
B
A
RELU MAX POOLING
1330
SPARSE
1332
COLUMN SWAP
1334
SWAPPED COLUMN INDICES
1340
LAYERED SPARSE BITMAP
1338
NON-ZERO COEFFICIENTS
1336
QUANTIZATION K-MEANS CLUSTERING PALETTE SHARING
1342
COMBINER
1344
OUTPUT FEATURE MAP COMPRESSION
1352

Fig. 13B

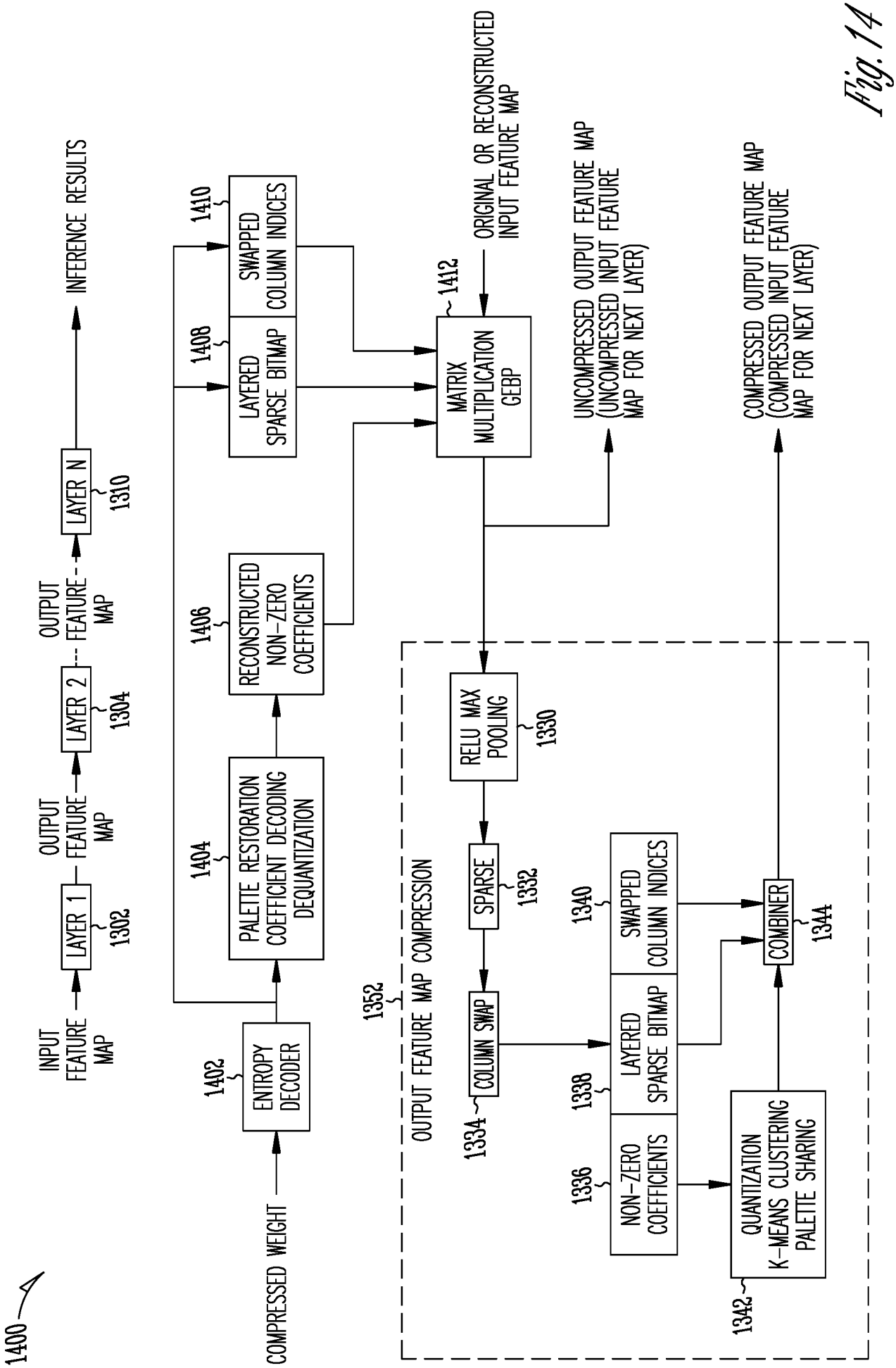
1400
INPUT FEATURE MAP
LAYER 1
1302
OUTPUT FEATURE MAP
LAYER 2
1304
OUTPUT FEATURE MAP
LAYER N
1310
INFERENCE RESULTS
COMPRESSED WEIGHT
1402
ENTROPY DECODER
1404
PALETTE RESTORATION COEFFICIENT DECODING DEQUANTIZATION
1406
RECONSTRUCTED NON-ZERO COEFFICIENTS
1408
LAYERED SPARSE BITMAP
1410
SWAPPED COLUMN INDICES
1412
MATRIX MULTIPLICATION GEBP
ORIGINAL OR RECONSTRUCTED INPUT FEATURE MAP
UNCOMPRESSED OUTPUT FEATURE MAP (UNCOMPRESSED INPUT FEATURE MAP FOR NEXT LAYER)
COMPRESSED OUTPUT FEATURE MAP (COMPRESSED INPUT FEATURE MAP FOR NEXT LAYER)
1352
OUTPUT FEATURE MAP COMPRESSION
RELU MAX POOLING
1330
SPARSE
1332
COLUMN SWAP
1334
LAYERED SPARSE BITMAP
1338
SWAPPED COLUMN INDICES
1340
NON-ZERO COEFFICIENTS
1336
QUANTIZATION K-MEANS CLUSTERING PALETTE SHARING
1342
COMBINER
1344

Fig. 14

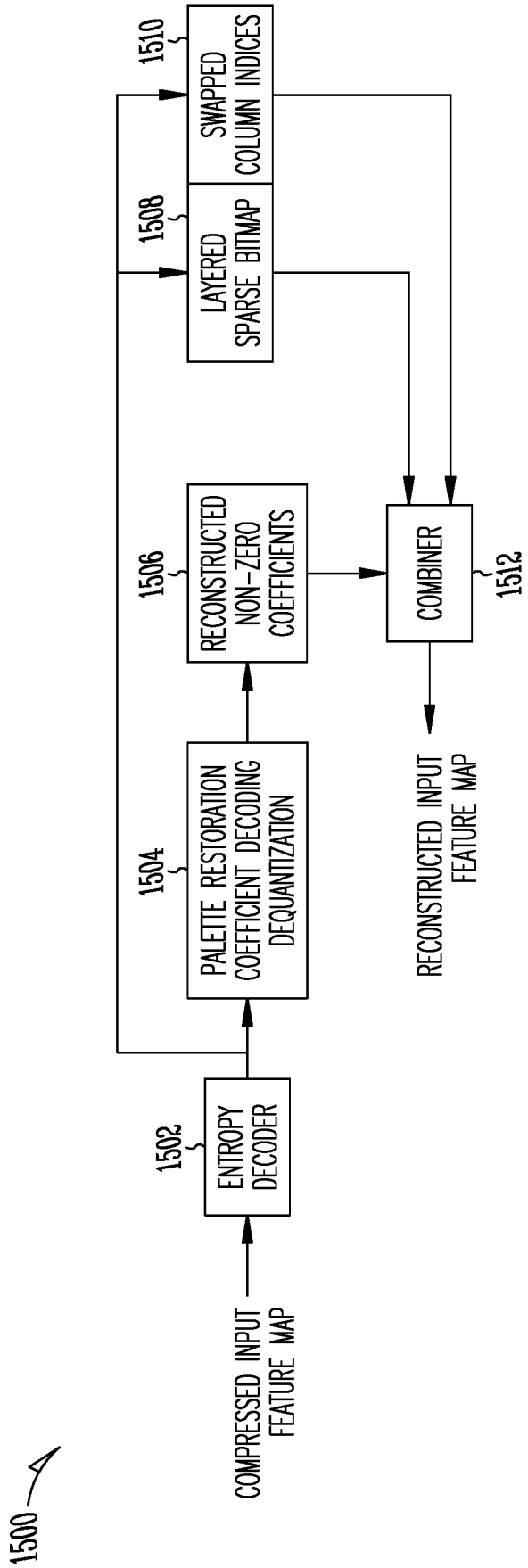
1500
COMPRESSED INPUT FEATURE MAP
1502
ENTROPY DECODER
1504
PALETTE RESTORATION COEFFICIENT DECODING DEQUANTIZATION
1506
RECONSTRUCTED NON-ZERO COEFFICIENTS
1508
LAYERED SPARSE BITMAP
1510
SWAPPED COLUMN INDICES
1512
COMBINER
RECONSTRUCTED INPUT FEATURE MAP

Fig. 15

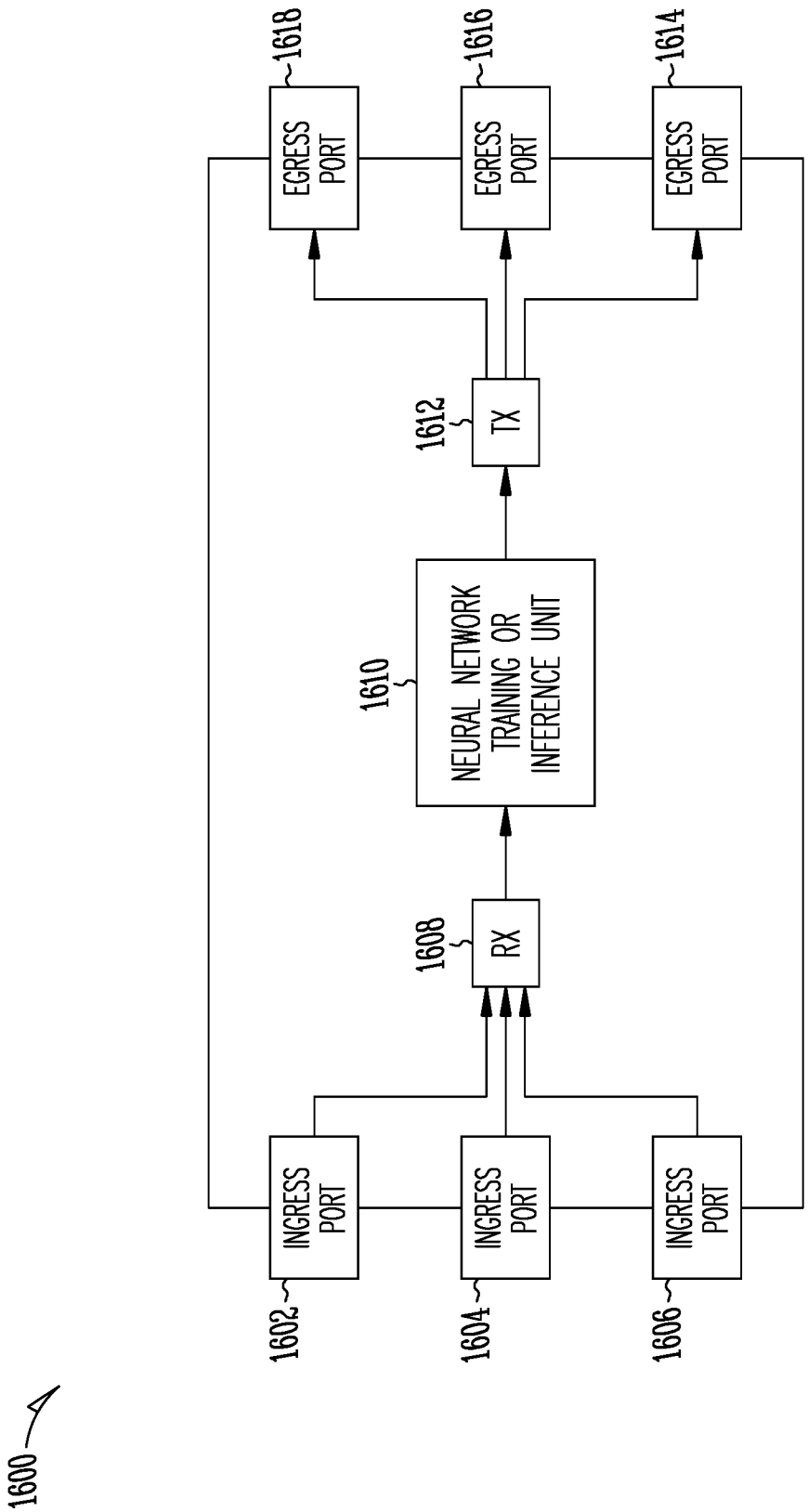
1600
1602
INGRESS PORT
1604
INGRESS PORT
1606
INGRESS PORT
1608
RX
1610
NEURAL NETWORK TRAINING OR INFERENCE UNIT
1612
TX
1614
EGRESS PORT
1616
EGRESS PORT
1618
EGRESS PORT

Fig. 16

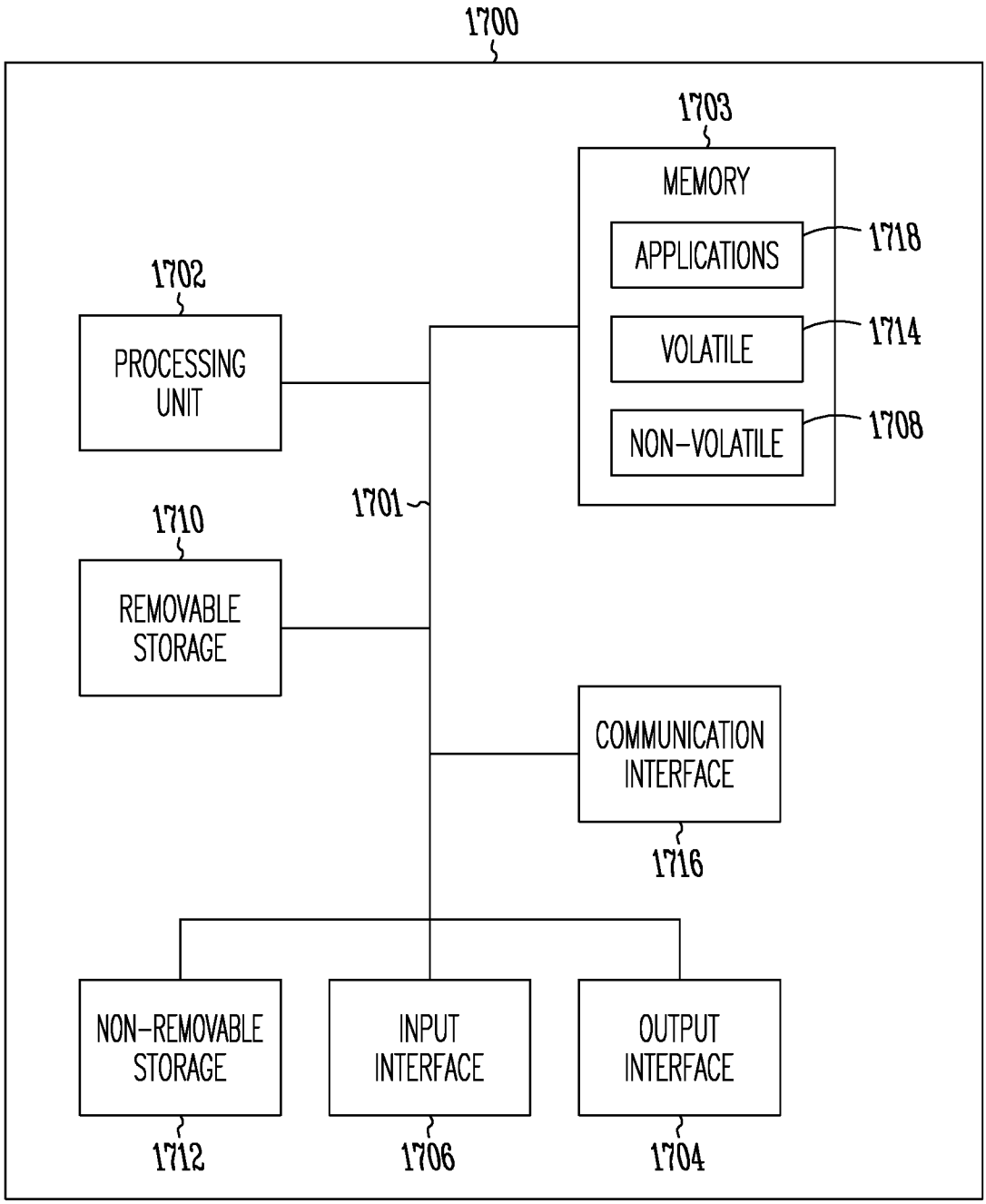
1700
1703
MEMORY
APPLICATIONS
1718
VOLATILE
1714
NON-VOLATILE
1708
1702
PROCESSING UNIT
1701
1710
REMOVABLE STORAGE
COMMUNICATION INTERFACE
1716
NON-REMOVABLE STORAGE
INPUT INTERFACE
OUTPUT INTERFACE
1712
1706
1704

Fig. 17

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2020/022753

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06N20/00
ADD.

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)
G06N

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	DHARMA TEJA VOOTURI ET AL: "Efficient Inferencing of Compressed Deep Neural Networks", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 1 November 2017 (2017-11-01), XP081284118, Section III; Section IV; figures 1, 2, 3 -----	1-20
Y	YUNHE WANG ET AL: "Beyond Filters: Compact Feature Map for Portable Deep Model", PROCEEDINGS OF THE 34TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, vol. 70, 2017, pages 3703-3711, XP055704576, Section 2 ----- -/--	1,7,13, 19

[X] 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
12 June 2020	22/06/2020

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Craciun, Paula

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	GUDOVSKIY DENIS ET AL: "DNN Feature Map Compression Using Learned Representation over GF(2)", 23 January 2019 (2019-01-23), ROBOCUP 2008: ROBOCUP 2008: ROBOT SOCCER WORLD CUP XII; [LECTURE NOTES IN COMPUTER SCIENCE; LECT.NOTES COMPUTER], SPRINGER INTERNATIONAL PUBLISHING, CHAM, PAGE(S) 502 - 516, XP047501220, ISBN: 978-3-319-10403-4 [retrieved on 2019-01-23] Section 3	1,7,13, 19

A	Song Han ET AL: "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", , 15 February 2016 (2016-02-15), XP055393078, Retrieved from the Internet: URL:https://arxiv.org/pdf/1510.00149v5.pdf [retrieved on 2017-07-21] Sections 2-4; figures 1-3	1-20

A	Feng Shi ET AL: "Sparse Winograd convolution neural networks on small-scale systolic arrays", Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 3 August 2018 (2018-08-03), XP055704583, DOI: 10.1145/nnnnnnn.nnnnnnn Retrieved from the Internet: URL:https://arxiv.org/pdf/1810.01973.pdf [retrieved on 2020-06-12] Sections 3-5	1-20

1