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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
02 August 2018 (02.08.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/140294 A1

(51) International Patent Classification:
G06N 3/063 (2006.01) *G06N 3/04* (2006.01)
G06N 3/08 (2006.01)

(21) International Application Number:
PCT/US2018/014303

(22) International Filing Date:
19 January 2018 (19.01.2018)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
201710061333.9 25 January 2017 (25.01.2017) CN

(71) Applicant: **MICROSOFT TECHNOLOGY LICENSING, LLC** [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: **XU, Ningyi**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, WA 98052-6399 (US). **ZHOU, Hucheng**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, WA 98052-6399 (US). **WANG, Wenqiang**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, WA 98052-6399 (US). **CHEN, Xi**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, WA 98052-6399 (US).

(74) Agent: **MINHAS, Sandip, S.** et al.; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, WA 98052-6399 (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, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,

(54) Title: NEURAL NETWORK BASED ON FIXED-POINT OPERATIONS

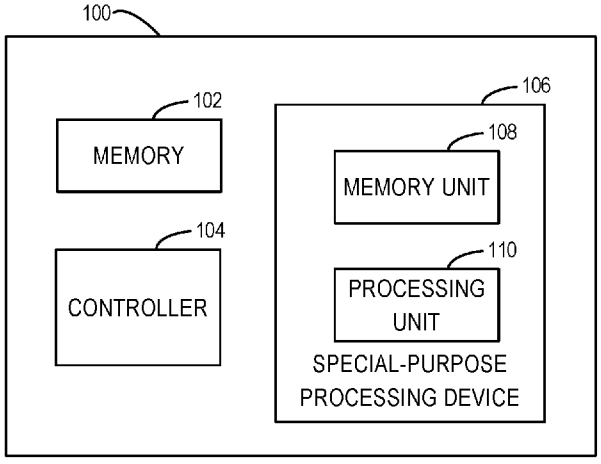

FIG. 1

(57) Abstract: Implementations of the subject matter described herein propose a solution for training a convolutional neural network. In this solution, parameters of the neural network are stored in a fixed-point format, such as weights and biases. The parameters in the first fixed-point format have a predefined bit-width and may be stored in a memory unit of the special-purpose processing device. The special-purpose processing device, when executing the solution, receives an input to a convolutional layer, reads parameters of the convolutional layer from the memory unit, and computes an output of the convolutional layer based on the input of the convolutional layer and the read parameters. In this way, the demands on the storage space and computing resources of the special-purpose processing device are lowered.

WO 2018/140294 A1

[Continued on next page]

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).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

NEURAL NETWORK BASED ON FIXED-POINT OPERATIONS

BACKGROUND

[0001] Neural networks have been widely and deeply applied in computer vision, natural language processing, and speech recognition. Convolutional neural network is a special neural network and includes a large amount of learnable parameters. Most of the current convolutional neural networks, even if deployed on one or more fast and power-hungry Graphics Processing Units (GPUs), take a great amount of time to train. Various solutions have been proposed to improve the computing speed of neural networks. However, the current solutions still have a number of problems to be solved in memory consumption and/or computation complexity.

SUMMARY

[0002] In accordance with implementations of the subject matter described herein, there is provided a solution for training a neural network. In the solution, a fixed-point format is used to store parameters of the neural networks, such as weights and biases. The parameters are also known as primal parameters to be updated for each iteration. Parameters in the fixed-point format have a predefined bit-width and can be stored in a memory unit of a special-purpose processing device. The special-purpose processing device, when executing the solution, receives an input to a layer of a neural network, reads parameters of the layer from the memory unit, and computes an output of the layer based on the input of the layer and the read parameters. In this way, the requirements for the memory and computing resources of the special-purpose processing device can be reduced.

[0003] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Fig. 1 illustrates a block diagram of a computing environment in which implementations of the subject matter described herein can be implemented;

[0005] Fig. 2 illustrates a block diagram of a neural network in accordance with an implementation of the subject matter described herein;

[0006] Fig. 3 illustrates an internal architecture for a forward pass of a convolutional layer of the neural network in accordance with an implementation of the subject matter described herein;

[0007] Fig. 4 illustrates an internal architecture for a backward pass of a layer of the neural network in accordance with an implementation of the subject matter described herein;

[0008] Fig. 5 illustrates a flowchart of a method for training a neural network in accordance with an implementation of the subject matter described herein;

[0009] Fig. 6 illustrates a block diagram of a device for training a neural network in accordance with an implementation of the subject matter described herein;

[0010] Fig. 7 illustrates a block diagram of a forward pass of the neural network in accordance with one implementation of the subject matter described herein; and

[0011] Fig. 8 illustrates a block diagram of a backward pass of the neural network in accordance with one implementation of the subject matter described herein.

[0012] Throughout the drawings, the same or similar reference symbols refer to the same or similar elements.

DETAILED DESCRIPTION OF EMBODIMENTS

[0013] The subject matter described herein will now be discussed with reference to several example implementations. It would be appreciated these implementations are discussed only for the purpose of enabling those skilled persons in the art to better understand and thus implement the subject matter described herein, rather than suggesting any limitations on the scope of the subject matter.

[0014] As used herein, the term "includes" and its variants are to be read as open terms that mean "includes, but is not limited to." The term "based on" is to be read as "based at least in part on." The term "one implementation" and "an implementation" are to be read as "at least one implementation." The term "another implementation" is to be read as "at least one other implementation." The terms "first," "second," and the like may refer to different or same objects. Other definitions, explicit and implicit, may be included below.

[0015] Recently, extensive studies have been focused on speeding up model training and inference using special purpose processing hardware, such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). Among these methods, model quantization has been considered to be one of the most promising approaches, because it not only significantly accelerates the speed and increases the power-efficiency, but also achieves comparable accuracy. Model quantization is intended to quantize the model parameters (as well as activations and gradients) to low bit-width values, while model binarization further pushes the limit of the quantization by extremely quantizing the parameters to be a binary value (a single bit, -1 and 1). As a result, during inference, memory footprint and memory accesses can be drastically reduced, and most

arithmetic operations can be implemented with bit-wise operations, i.e., binary convolution kernel. However, it is required that the quantization solution be further improved so as to reduce memory footprint, to lower computation complexity, and so on.

[0016] Basic principles and various example implementations of the subject matter will now be described with reference to the drawings. It would be appreciated that, for the sake of clarity, the implementations of the subject matter described herein will be described with reference mainly to a convolutional neural network. In this way, a convolutional layer is described as an example of a representative layer of the neural network. However, it would be appreciated that this is not intended to limit the scope of the subject matter described herein. The idea and principles described herein are suitable for any suitable neural network system currently known or to be developed in the future.

Example Environment

[0017] Fig. 1 illustrates a block diagram of a computing device 100 in which implementations of the subject matter described herein can be implemented. It would be appreciated that the computing device 100 shown in Fig. 1 is merely illustration but not limiting the function and scope of the implementations of the subject matter described herein in any way. As illustrated in Fig. 1, the computing device 100 may include a memory 102, a controller 104, and a special-purpose processing device 106.

[0018] In some implementations, the computing device 100 can be implemented as various user terminals or service terminals with computing capability. The service terminals may be servers, large-scale computer devices, and other devices provided by various service providers. The user terminals, for example, are any type of mobile terminals, fixed terminals, or portable terminals, including mobile phones, stations, units, devices, multimedia computers, multimedia tablets, Internet nodes, communicators, desktop computers, laptop computers, notebook computers, netbook computers, tablet computers, Personal Communication System (PCS) devices, personal navigation devices, Personal Digital Assistants (PDAs), audio/video players, digital camera/camcorders, positioning devices, television receivers, radio broadcast receivers, electronic book devices, game devices, or any combination thereof, including the accessories and peripherals of these devices or any combination thereof. It is also contemplated that the computing device 100 can support any type of interface to the user (such as "wearable" circuitry and the like).

[0019] The special-purpose processing device 106 may further include a memory unit 108 and a processing unit 110. For example, the special-purpose processing device 106 may be a Field Programmable Gate Array (FPGA), an Application Specific Integrated Circuit

(ASIC), a processor or a Central Processing Unit (CPU) with a customized processing unit, or a Graphics Processing Unit (GPU). Therefore, the memory unit 108 may be referred to as a memory-on-chip and the memory 102 may be referred to as a memory-off-chip accordingly. In some implementations, the processing unit 110 can control the overall operations of the special-purpose processing device 106 and perform various computations.

[0020] The memory 102 may be implemented by various storage media, including but not limited to volatile and non-volatile medium, and removable and non-removable medium. The memory 102 can be a volatile memory (such as a register, cache, Random Access Memory (RAM)), a non-volatile memory (such as, a Read-Only Memory (ROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), flash memory)), or any combinations thereof. The memory 102 may be removable and non-removable medium, and may include a machine-readable medium, such as a memory, flash drive, magnetic disk or any other media that can be used to store information and/or data and can be accessed by the computing device 100.

[0021] The controller 104 can control the start and end of the computing process and may further provide inputs required for forward pass of the convolutional neural network. In addition, the controller 104 can also provide the weight data for the neural network. The controller 104 communicates with the special-purpose processing device 106 via a standard interface such as a PCIe bus. The controller 104 assigns the computing tasks to the processing unit 110 on the special-purpose processing device 106. The processing unit 110 begins the computing process after receiving the start signal from the controller 104. The controller 104 provides the inputs and weights to the processing unit 110 for computation. The memory unit 108 of the special-purpose processing device 106 may be used to store parameters, such as convolution kernel weights, while the memory 102 may store input and output feature maps and intermediate data generated during computation. The special-purpose processing device 106 completes the computation of the forward pass of the neural network and then returns the output result obtained from the previous layer of the convolutional neural network to the controller 104. However, it would be appreciated that the above control process is merely exemplary. Those skilled in the art may change the control process after understanding the implementations of the subject matter described herein.

[0022] The computing device 100 or the special-purpose processing device 106 can perform the training of the neural networks in the implementations of the subject matter described herein. During the training of the neural network, model parameters, also known

as primal parameters, are defined to be stored weights and biases. The parameters are updated during each iteration. In the prior art, the parameters are stored in high-resolution format. These parameters will be quantized or binarized every time before forward pass, and the associated gradient accumulation is still performed in a floating-point domain. Thus, the special-purpose processing device, such as the FPGA and the ASIC, still need to implement expensive floating-point multiplication-accumulation to handle parameter updates, and even much more expensive nonlinear quantization method.

[0023] In accordance with some implementations of the subject matter described herein, the limits of quantization are further pushed by representing the parameters in a fixed-point format, which can decrease the bit-width of the parameters, so as to dramatically reduce the total memory. For example, 8-bit fixed-point number can reduce the total memory space to a quarter compared with the 32-bit floating-point number. This makes it possible to store the parameters on the memory-on-chip of the special-purpose processing device rather than memory-off-chip. In the case of 45nm CMOS process node, it means 100 times energy efficiency. Moreover, fixed point arithmetic operations with low resolution in the special-purpose processing device are much faster and energy efficient than floating-point representation. Furthermore, fixed-point operations generally will dramatically reduce logic usage and power consumption, combined with higher clock frequency, shorter pipelines, and increased throughput capabilities.

Convolutional Neural Network

[0024] Convolutional neural network is a particular neural network, which usually includes a plurality of layers with each layer including one or more neurons. Each neuron obtains input data from the input of the neural network or the previous layer to perform respective operations, and outputs the results to the next layer or the output of the neural network model. The input of the neural network may be, for example, images, e.g., RGB images of particular pixels. In the classification problem, the output of the neural network is a score or a probability of a different class. The last layer (usually the fully-connected layer) of the neural network may be provided with a loss function, which can be a cross entropy loss function. During training of the neural network, it is generally required to minimize the loss function.

[0025] Structure of the convolutional neural network is specially designed for the situation with images as the input data. In this case, therefore, the convolutional neural network is highly efficient and the number of parameters required in the neural network is greatly reduced.

[0026] In the convolutional neural network, each layer is arranged in three dimensions: width, height, and depth. Each layer of the convolutional neural network converts the three-dimensional input data to a three-dimensional activation data and outputs the converted three-dimensional activation data. The convolutional neural network includes various layers arranged in a sequence and each layer sends the activation data from one layer to another. The convolutional neural network mainly includes three types of layers: a convolutional layer, a pooling layer, and a fully-connected layer. By stacking the layers over each other, a complete convolutional neural network may be constructed.

[0027] Fig. 2 illustrates example architecture of a convolutional neural network (CNN) 200 in accordance with some implementations of the subject matter described herein. It should be understood that the structure and functions of the convolutional neural network 200 are described for illustration, and do not limit the scope of the subject matter described herein. The subject matter described herein can be implemented by different structures and/or functions.

[0028] As shown in Fig. 2, the CNN 200 includes an input layer 202, convolutional layers 204 and 208, pooling layers 206 and 210, and an output layer 212. Generally, the convolutional layer and the pooling layer are arranged alternately. For example, as shown in Fig. 2, the convolutional layer 204 is followed by the adjacent pooling layer 206 and the convolutional layer 208 is followed by the adjacent pooling layer 206. However, it would be appreciated that the convolutional layer may not be followed by a pooling layer. In some implementations, the CNN 200 only includes one of the pooling layers 206 and 210. In some implementations, there are even no pooling layers.

[0029] As described above, each of the input layer 202, the convolutional layers 204 and 208, the pooling layers 206 and 210, and the output layer 212 includes one or more planes, also known as feature maps or channels. The planes are arranged along the depth dimension and each plane may include two space dimensions, i.e., width and height, also known as space domain.

[0030] To help understand the ideas and principles of the subject matter described herein, the principle of the CNN 200 will now be described with reference to an example application of the image classification. It would be appreciated that the CNN 200 can be easily extended to any other suitable applications. Moreover, the input layer 202 may be represented by the input images, such as 32*32 RGB images. In this case, the dimension for the input layer 202 is 32*32*3. In other words, the width and height for the image is 32 and there are three color channels.

[0031] Feature map of each of the convolutional layers 204 and 208 may be obtained by applying convolutional operations on the feature map of the previous layer. By convolutional operations, each neuron in the feature map of the convolutional layers is only connected to a part of neurons of the previous layer. Therefore, applying convolutional operations on the convolutional layers indicates the presence of sparse connection between these two layers. After applying convolutional operations on the convolutional layers, the result may be applied with an activation function to determine the output of the convolutional layers.

[0032] For example, in the convolutional layer 204, each neuron is connected to a local area in the input layer 202 and computes the inner product of the local area and its weights. The convolutional layer 204 may compute the output of all neurons and, if 12 filters (also known as convolution kernel) are used, the obtained output data will have a dimension of [32*32*12]. In addition, activation operations may be performed on each output data in the convolutional layer 204 and the common activation functions include Sigmoid, tanh, ReLU, and so on.

[0033] The pooling layers 206 and 210 down sample the output of the previous layer in space dimension (width and height), so as to reduce data size in space dimension. The output layer 212 is usually a fully-connected layer, in which each neuron is connected to all neurons of the previous layer. The output layer 212 computes classification scores and converts data size to one-dimensional vectors, each element of the one-dimensional vectors corresponding to a respective category. For instance, regarding the convolutional network of the images in CIFAR-10 for classification, the last output layer has a dimension of 1*1*10 because the convolutional neural network will finally compress the images into one vector consisting of classification scores, where the vector is arranged along the direction of depth.

[0034] It can be seen that the convolutional neural network converts the images one by one from original pixel values to final classification score values. For example, when the convolutional layers and the fully-connected layer operate on the corresponding inputs, both the activation function and the learning parameters can be used, where parameters in the convolutional layers and the fully-connected layer can be optimized based on different optimization solutions. Examples of the optimization solutions include but not limited to stochastic gradient descent algorithm, adaptive momentum estimation (ADAM) method and the like. Therefore, the errors between the classification scores obtained by the convolutional neural network and labels of each image can be lowered as much as possible for the data in the training data set.

[0035] Training of the neural network can be further implemented by a backward pass. In this method, the training set is input into the input layer of the neural network, e.g., input the training set into the input layer of the neural network in batches and iteratively update the parameters of the neural network by batch. Samples of each batch can be regarded as a mini-batch. After multiple iterations, all samples in the training set have been trained once, which is called as an epoch.

[0036] In each iteration, a plurality of inputs is grouped into a mini-batch, which is provided to the input layer. By the forward pass, the inputs are propagated layer by layer to the output layer of the neural network, so as to determine the output of the neural network, such as classification scores. The classification scores are then compared with the labels in the training set to compute prediction errors by the loss function, for example. The output layer discovers that the output is inconsistent with the right label. At this time, the parameters of the last layer in the neural network may be adjusted and then parameters of the last second layer connected to the last layer may be adjusted. Accordingly, the layers are adjusted layer by layer in a backward direction. When the adjustment for all parameters in the neural network is completed, the same process is performed on the next mini-batch. In this way, the process is performed iteratively until the predefined termination condition is satisfied.

Binary Neural Network

[0037] The following description introduces a binary neural network (BNN) into which the implementations of the subject matter described herein are applied. In BNN, weights and activations can be binarized to significantly speed up the performance by using the bit convolution kernels. In some implementations, the floating-point value is converted into a single bit by the stochastic method. Although the stochastic binarization can get better performance, the computation complexity of the solution is higher since it requires the hardware resources to generate random bits when quantizing. In some implementations, a deterministic method is adopted to convert the floating-point value into a single bit and the deterministic solution is lower in computation complexity. For example, a simple sign function sign $(\cdot)$ is used to binarize the floating-point value, as shown in equation (1).

$$w^b = \text{sign}(w) = \begin{cases} +1 & w \geq 0, \\ -1 & w < 0 \end{cases} \quad (1)$$

[0038] As indicated in the equation (1), when the weigh w is great than or equal to zero, it is converted to +1, and when the weight w is less than zero, it is converted to -1, such that the obtained value w^b is a one-bit binary number. Such binary conversion drastically

reduces computation complexity and memory consumption in forward pass. However, the derivative of the sign function is 0 almost everywhere, which makes the gradients of the loss function c cannot be propagated in backward pass. To address this problem, a "straight-through estimator" STE method is employed, as shown in equation (2):

$$\begin{aligned} &\text{Forward:}\ \ r_o = \text{sign}(r_i) \\ &\text{Backward:}\ \ \frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}\mathbf{1}_{|r_i|\leq 1} \end{aligned} \tag{2}$$

[0039] In equation (2), $\mathbf{1}_{|r_i|\leq 1}$ represents an indicator function, where when the input r_i satisfies the condition that $|r_i|\leq 1$, the value of the indicator function is 1; when the input r_i satisfies the condition that $|r_i|\leq 1$, the value of the indicator function is 0. Accordingly, the STE method preserves the gradient information and cancels the gradient information when r_i is too large. If the gradient information is not cancelled when r_i is too large, it may cause a significant drop in the performance of the model.

[0040] From another perspective, STE can also be regarded as applying hard-tanh activation function HT to r_i, where HT is defined as:

$$HT(r_i) = \begin{cases} +1 & r_i > 1, \\ r_i & r_i \in [-1, 1], \\ -1 & r_i < -1. \end{cases} \tag{3}$$

[0041] Correspondingly, the derivative of HT is defined as:

$$\frac{\partial HT(r_i)}{\partial r_i} = \begin{cases} 0 & r_i > 1, \\ 1 & r_i \in [-1, 1], \\ 0 & r_i < -1. \end{cases} \tag{4}$$

[0042] It can be seen that the STEs defined in equations (4) and (2) are exactly the same. With equation (3) and (4), the neural network can binarize both activations and weights during forward pass, while still reserving real-valued gradients to guarantee stochastic gradient descent to properly operate.

Fixed-Point Format

[0043] In accordance with implementations of the subject matter described herein, weights and gradients can be stored in a fixed-point format, e.g., weights can be stored in the memory unit 108 of the special-purpose processing device 106 in a fixed-point format. The fixed-point format includes a *l*-bit signed integer mantissa and a global scaling factor (e.g., 2^{-n}) shared with the fixed-point values, as shown in equation (5):

$$\mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_K \end{pmatrix} = \begin{pmatrix} m_1 \\ \vdots \\ m_K \end{pmatrix} \cdot 2^{-n} \tag{5}$$

where n and mantissa m_1~m_k are integers.

[0044] It can be seen that the vector **v** includes K elements v_1~v_k, which share one scaling factor 2^{-n}. Here the integer n actually indicates the radix point position of the *l*-bit fixed-point number. In other words, the scaling factor in fact refers to the position of the radix point. Because the scaling factor is usually fixed, i.e., fixed radix point, this data format is called as fixed-point number. Lowering the scaling factor will reduce the range of the fixed-point format but increases the accuracy of the fixed-point format. The scaling factor is usually set to be a power of 2, since the multiplication can be replaced by bit shift to reduce the complexity of computation.

[0045] In some implementations, the following equation (6) may be used to convert data x (such as, a floating-point number) into an *l*-bit fixed-point number with the scaling factor 2^{-n},

$$FXP(x, l, n) = Clip(\lfloor \frac{x}{2^{-n}} + 0.5 \rfloor \cdot 2^{-n}, MIN, MAX) \quad (6)$$

where $\lfloor \cdot \rfloor$ means rounding down, and MIN and MAX respectively denote the minimum value and the maximum value of the *l*-bit fixed-point number with the scaling factor 2^{-n}. In some implementations, to make the summing circuit and multiplication circuit simpler by taking full advantage of all ordinality 2^l, MIN and MAX are defined as follows:

$$\begin{cases} MIN = -2^{l-1} * 2^{-n} \\ MAX = (2^{l-1} - 1) * 2^{-n} \end{cases} \quad (7)$$

[0046] It is observed that equation (6) also defines the rounding behavior, i.e., indicated by the rounding down operation $\lfloor \cdot \rfloor$. In addition, equation (6) also defines the saturating behavior represented by Clip. In other words, when $\lfloor \frac{x}{2^{-n}} + 0.5 \rfloor \cdot 2^{-n}$ is greater than MAX, the value of the converted fixed-point number is MAX; when $\lfloor \frac{x}{2^{-n}} + 0.5 \rfloor \cdot 2^{-n}$ is less than MIN, the value of the converted fixed-point number is MIN.

[0047] In the following, the operations for converting data into fixed-point format may be implemented by equations (6) and (7), unless indicated otherwise. Of course, any other suitable conversion operations may also be adopted.

Quantization

[0048] It is known that magnitudes of weight, activation, and gradient will fluctuate during training, where the gradient fluctuation is most apparent. To match the fluctuations, different bit-widths and scaling factors are assigned to the parameters, activations, and gradients in different layers and the scaling factors of the parameters are updated accordingly during iteration. Moreover, different scaling factors can also be assigned to

weights and biases among the parameters.

[0049] In some implementations of the subject matter described herein, the scaling factor may be updated based on the data range. Specifically, it may be determined, based on overflow of the data (e.g., overflow rate and/or overflow amount), whether to update the scaling factor and how to update the scaling factor. The method for updating the scaling factor will now be explained with reference to weights. However, it would be appreciated that the method can also be applied for other parameters.

[0050] In the case of the current scaling factor, it may be determined whether the overflow rate of the weights exceeds the predefined threshold. If the overflow rate exceeds the predefined threshold, the range of the fixed-point number is too small and the scaling factor should be increased accordingly. For example, the scaling factor may be multiplied with the cardinal number (e.g., 2). For example, the radix point may be shifted right by one bit. If the overflow rate does not exceed the predefined threshold and, after the weight is multiplied with 2, the overflow rate is still below the predefined threshold, the range of the fixed-point number is too large. Therefore, the scaling factor may be reduced, for example, by dividing the scaling factor by the cardinal number (e.g., 2). For example, the radix point may be shifted left by one bit.

[0051] Compared with binary weights and activations, gradients usually require higher accuracy. Therefore, quantization of the gradients should be carefully reviewed. Because the linear quantization approach does not converge well, a non-linear quantization function is employed instead to quantize the gradients. However, these non-linear quantization solutions will inevitably increase the computation overhead, which is undesirable. Hence, in accordance with implementations of the subject matter described herein, a linear quantization solution is adopted to lower the computation complexity. As described above, if the linear quantization function is simply used in the training of the neural network, it will introduce too strong regularization and impede convergence of the neural network model. However, in the case of using a solution for updating adaptive scaling factors, the linear quantization approach may be employed without causing non-convergence or dramatic decrease in model performance.

Forward Pass

[0052] Fig. 3 illustrates an internal architecture for a forward pass of a convolutional layer 300 of the convolutional neural network in accordance with an implementation of the subject matter described herein. The convolutional layer 300 may be a k-th layer of the neural network. For example, the convolutional layer 300 may be a convolutional layer

204 or 208 in the convolutional neural network as shown in Fig. 2. In Fig. 3, legend 10 represents binary numbers and legend 20 represents fixed-point numbers. It would be appreciated that, although Fig. 3 illustrates a plurality of modules or sub-layers, in specific implementations one or more sub-layers may be omitted or modified for different purposes.

[0053] As shown in Fig. 3, parameters of the convolutional layer 300 includes weights 302 and biases 304, respectively denoted as W_k^{fxp} and b_k^{fxp}, i.e., weights and biases of the k-th layer. In some implementations, parameters of the convolutional layer 300 may be represented and stored in fixed-point format instead of floating-point format. The parameters in fixed-point format may be stored in the memory unit 108 of the special-purpose processing device 106 and may be read from the memory unit 108 during operation.

[0054] In the forward pass, the weights 302 in fixed-point format are converted by a binary sub-layer 308 to binary weights 310, which may be represented by W_k^b. For example, the binary sub-layer 308 may convert the fixed-point weights 302 into binary weights 310 by a sign function, as shown in equation (1). Moreover, the convolutional layer 300 further receives an input 306, which may be represented by X_k^b. For example, when the convolutional layer 300 is an input layer of the neural network (i.e., k=1), the input 306 can be input images of the neural network. In this case, the input 306 can be regarded as an 8-bit integer vector (0-225). In another case, when the convolutional layer 300 is a hidden layer or an output layer of the neural network, for example, the input 306 may be an output of the previous layer, which may be a binary vector (+1 or -1). In both cases, convolutional operation only includes integer multiplication and accumulation and may be computed by bit convolution kernels. In some implementations, if the convolutional layer 300 is the first layer, it may be processed according to equation (8),

$$\begin{aligned} s &= x \cdot w^b; \\ s &= \sum_{n=1}^{8} 2^{n-1}(x^n \cdot w^b). \end{aligned} \tag{8}$$

where x represents an input 306 in an 8-bit fixed-point format, w^b represents a binary weight and x^n represents the mantissa of the n-th element of vector x.

[0055] A normalization sub-layer 316 represents integer batch normalization (IBN) sub-layer, which normalizes input tensor within a mini-batch with mean and variance. Different from conventional batch normalization performed in floating-point domain, all intermediate results involved in the sub-layer 316 are either 32-bit integers or low resolution fixed-point values. Since integer is a special fixed-point number, the IBN sub-layer 316

only includes corresponding fixed-point operations. Subsequently, the quantization sub-layer 318 converts the output of the IBN sub-layer 316 to a predefined fixed-point format.

[0056] Specifically, for the IBN sub-layer 316, the input may be fixed-point input in a mini-batch $\mathbb{X}_{\text{in}} = \{x_1, \cdots, x_N\}$, including N elements. To obtain normalized output $\mathbb{X}_{\text{out}} = \{y_1^{fxp}, \cdots, y_N^{fxp}\}$, the sum $\text{sum1} \leftarrow \sum_{i=1}^{N} x_i$ and the sum of squares $\text{sum2} \leftarrow \sum_{i=1}^{N} x_i^2$ of all inputs can be determined. Then, the mean value $\text{mean} \leftarrow \text{Round}(\text{sum1}/N)$ and the variance $\text{var} \leftarrow \text{Round}(\text{sum2}/N) - \text{mean}^2$ of the input are computed based on sum1 and sum 2, wherein Round $(\cdot)$ means rounding to the nearest 32-bit integer. Then, the normalized output $y_i \leftarrow (x_i - \text{mean})/\text{Round}(\sqrt{\text{var}})$ is determined based on the mean and the variance. The normalized output can be converted to $y_i^{fxp} \leftarrow \text{FXP}(y_i)$ in a predefined fixed-point format via the sub-layer 318.

[0057] For the output of the IBN sub-layer 316, the method for updating scaling factors described in the Quantization section above can be used to update the scaling factors. For example, it may be first determined whether the overflow rate of the IBN output exceeds the predefined threshold. If the overflow rate is greater than the predefined threshold, the range of the IBN output is extended, that is, increasing the scaling factor or right shifting the radix point in fixed-point format when the cardinal number is 2. This will not be repeated because it is substantially the same as the method for updating scaling factors described with reference to quantization.

[0058] In some implementations, a summing sub-layer 320 adds the output of the IBN sub-layer 136 with the bias 304 to provide an output s_k. The bias 304 may be read from the memory unit 108 of the special-purpose processing device 106. The activation sub-layer 322 represents an activation function, which is usually implemented by a non-linear activation function, e.g., hard-tanh function HT. The output of the activation sub-layer 322 is converted via the quantization sub-layer 324 to an output 326 in a fixed-point format, which is denoted by X_{k+1}^b, to be provided to the next layer (k+1 layer) of the neural network. Moreover, the last layer of the neural network may not include the activation sub-layer 322 and binary layer 324, i.e., the loss function layer is computed in a floating-point domain.

[0059] In some implementations, a pooling layer is located after the convolutional layer 300. For example, as shown in Fig. 2, both of the convolutional layers 204 and 208 are followed by a pooling layer 206 in the convolutional neural network 200. In this case, the pooling layer may be incorporated into the convolutional layer 300 to further reduce computation complexity. For example, the pooling layer 206 is incorporated into the

convolutional layer 204 in the convolutional neural network 200. As shown in Fig. 3, the pooling sub-layer 314 indicated by the dotted line may be incorporated into the convolutional layer 300 and arranged between the convolutional sub-layer 321 and the IBN sub-layer 316.

[0060] The above description introduces the forward pass with reference to a convolutional layer 300. It would be appreciated that the forward pass of the entire neural network may be stacked by a plurality of similar processes. For example, the output of the k-th layer is provided to the k+1 layer to serve as an input of the k+1 layer; and the process continues layer by layer. In the convolutional neural network 200 of Fig. 2, the output of the convolutional layer 204 is determined from the architecture of the convolutional layer 300 (without the sub-layer 314). If the pooling layer 206 is incorporated into the convolutional layer 204, the output of the pooling layer 206 may be determined by the architecture of the convolutional layer 300 (including the sub-layer 314). Then, the output is provided to the convolutional layer 208 and the classification category is provided at the output layer 212.

Backward Pass

[0061] Fig. 4 illustrates an internal architecture for a backward pass of a convolutional layer 400 of the convolutional neural network in accordance with an implementation of the subject matter described herein. The backward pass process is shown in Fig. 4 from right to left. In Fig. 4, legend 30 represents floating-point number and legend 20 represents fixed-point number. It would be appreciated that, although the forward pass and backward pass process of the convolutional layer is respectively indicated by the signs 300 and 400, the convolutional layers 300 and 400 may refer to the same layer in the neural network. For example, the convolutional layers 300 and 400 may be the architecture for implementing the forward pass and backward pass of the convolutional layer 204 or 208 in the convolutional neural network 200. It would be further appreciated that, although Fig. 4 illustrates a plurality of modules or sub-layers, each sub-layer can be omitted or modified in specific implementations for different purposes in view of different situations.

[0062] As shown in Fig. 4, during backward pass, the convolutional layer 400 receives a backward input 426 from a next layer of the neural network, e.g., if the convolutional layer 400 is a k-th layer, the convolutional layer 400 receives a backward input 426 from the k+1-th layer. The backward input 426 may be a gradient of the loss function with respect to the forward output 326 of the convolutional layer 300. The gradient may be in floating-point

format and may be represented as $g_{x_{k+1}^b}$.

[0063] The backward input 426 is converted to a fixed-point value 430 (denoted by $g_{X_{k+1}^b}^{\text{fxp}}$) by the quantization sub-layer 424. The activation sub-layer 422 computes its output based on the fixed-point value 430, i.e., the gradient of the loss function with respect to the input s_k of the activation sub-layer 322, denoted by $g_{s_k}^{fxp}$.

[0064] It would be appreciated that most of the sub-layers in Fig. 4 corresponds to the sub-layers shown in Fig. 3. For example, the activation sub-layer 322 in Fig. 3 corresponds to the activation sub-layer 422 in Fig. 4, which serves as a backward gradient operation for the activation sub-layer 322. If the input of the activation sub-layer 322 is x, the output thereof is y, the backward input of the corresponding activation sub-layer 422 is a gradient of the loss function with respect to the output y and the backward output is a gradient of the loss function with respect to the input x. In Fig. 3, if the hard-tanh function serves as the activation function, the operations performed by the activation sub-layer 322 are shown in equation (3). Accordingly, the operations performed by the activation sub-layer 422 are shown in equation (4). Therefore, in the context of the subject matter described herein, the names for the two types of sub-layers are not distinguished from each other.

[0065] The backward output of the activation sub-layer 422 is provided to the summing sub-layer 420, which corresponds to the summing sub-layer 320, and the gradients of the loss function with respect to two inputs of the summing sub-layer 320 may be determined. Because an input of the sub-layer 320 is the bias, the gradient of the loss function with respect to the bias may be determined and the gradient is provided to the quantization sub-layer 428. Subsequently, the gradient is converted to a fixed-point gradient by the quantization sub-layer 428 for updating the bias 404 (represented by b_k^{fxp}). The fixed-point format has a specific scaling factor, which may be updated in accordance with the method for updating scaling factors as described in the Quantization section above.

[0066] Another backward output of the summing sub-layer 420 is propagated to the IBN sub-layer 418. In forward pass, a fixed-point format is used to compute the IBN sub-layer 418. However, if the same method is applied in the backward pass and the backward propagation of IBN is restricted in fixed-point representation, non-negligible accuracy degradation will occur. In some implementations, therefore, the IBN sub-layer 418 is returned to the floating-point domain for operations, so as to provide an intermediate gradient output. As shown in Fig. 4, the intermediate gradient output is a gradient of the

loss function with respect to the convolution of the input and parameters. Hence, an additional quantization sub-layer 416 is utilized after the IBN sub-layer 418 for converting the floating-point format into a fixed-point format. The quantization sub-layer 416 converts the intermediate gradient output to a fixed-point format having a specific scaling factor, which may be updated according to the method for updating scaling factors as described in the Quantization section above.

[0067] The convolutional sub-layer 412 further propagates a gradient $g_{w_k^b}$ of the loss function with respect to the weight W_k^b and a gradient $g_{x_k^b}$ of the loss function with respect to an output X_k^b of the convolutional layer. Because the input X_k^b is either an 8-bit integer vector (for the first layer, i.e., k=1) or a binary vector (for other layers, i.e., k≠1) and the weight W_k^b is a binary vector, the convolutional sub-layer 412 only contains fixed-point multiplication and accumulation, thereby resulting in a very low computation complexity.

[0068] The backward output $g_{x_k^b}$ of the convolutional sub-layer 412 provides a backward output 406 of the convolutional layer 400 to a previous layer. The backward output $g_{w_k^b}$ of the convolutional sub-layer 412 is converted to a fixed-point format via the quantization layer 408 to update the weight 402 (represented by W_k^{fxp}). The fixed-point format has a specific scaling factor, which may be updated according to the method for updating scaling factors as described in the Quantization section above.

[0069] After determining the gradient of the loss function with respect to the parameters by the backward pass, the parameters may be updated. As described above, the parameter may be updated by various updating rules, e.g., stochastic gradient descent, Adaptive Momentum Estimation (ADAM), or the like. In some implementations, the updating rules are performed in the fixed-point domain to further reduce floating-point computation. It would be appreciated that, although reference is made to the ADAM optimization method, any other suitable methods currently known or to be developed in the further may also be implemented.

[0070] ADAM method dynamically adjusts the learning rate for each parameter based on a first moment estimate and a second moment estimate of the gradient of the loss function with respect to each parameter. Fixed-point ADAM optimization method differs from the standard ADAM optimization method in that the fixed-point ADAM method operates entirely within the fixed-point domain. In other words, its intermediate variables (e.g., first

moment estimate and second moment estimate) are represented by fixed-point numbers. To be specific, one fixed-point ADAM learning rule is represented by the following equation (9), which converts the standard ADAM updating rules to a fixed-point format.

$$\begin{aligned} &1:\ m_t^{\text{fxp}} \leftarrow \text{FXP}(\beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t, l_2, n_2) \\ &2:\ v_t^{\text{fxp}} \leftarrow \text{FXP}(\beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2, 2l_2, 2n_2) \\ &3:\ u_t^{\text{fxp}} \leftarrow \text{FXP}(\sqrt{v_t^{\text{fxp}}} + \epsilon, l_2, n_2) \\ &4:\ \theta_t \leftarrow \text{FXP}(\theta_{t-1} - \eta_t \cdot \sqrt{1-\beta_2}/(1-\beta_1)\cdot m_t^{\text{fxp}}/u_t^{\text{fxp}}, l_1, n_1) \end{aligned} \tag{9}$$

In equation (9), g_t^2 denotes element-by-element square $g_t \circ g_t$. For the sake of simplicity, $1-\beta_1^t$ and $1-\beta_2^t$ are respectively fixed to be $1-\beta_1$ and $1-\beta_2$. FXP(·) represents a function of equation (6). The default settings are $1-\beta_1 = 2^{-3}$, $1-\beta_2 = 2^{-8}$, and $\epsilon = 2^{-20}$. The parameter θ_{t-1} represents the current fixed-point parameter value with a fixed-point format l_1, n_1, and θ_t represents the updated fixed-point parameter value. The fixed-point format for the gradient g_t is l_2 and n_2, and η_t is the learning rate. It can be seen that the ADAM method computes the updated parameters by calculating the intermediate variables m_t, v_t, and u_t, and only includes respective fixed-point operations.

[0071] By the fixed-point ADAM method, the updated weight W_k^{fxp} and bias b_k^{fxp} can be computed. As described above, these parameters can be stored in a memory unit 108 of the special-purpose processing device 106 in a fixed-point format. In addition, the scaling factors for the fixed-point format of the parameters may also be updated as described above. The scaling factors may be updated according to the method for updating scaling factors as described in the Quantization section above.

[0072] Additionally, if the pooling layer is incorporated into the convolutional layer 300 to serve as its pooling sub-layer 314 in the forward pass, a corresponding pooling layer should be incorporated into the convolutional layer 400 to serve as its pooling sub-layer 414 in the backward pass.

[0073] It can be seen that in the architecture shown in Figs. 3 and 4, at most two portions are implemented by floating-point numbers. The first portion is the loss function and the second portion is a backward pass of the gradient in the IBN sub-layer 418. Therefore, the floating-point computations are avoided as much as possible to lower computation complexity and reduce memory space.

[0074] Additionally, in the architecture shown in Figs. 3 and 4, the quantization sub-layer may be implemented by a linear quantization method, and an adaptive updating method for scaling factors of the fixed-point parameters corresponding to the quantization sub-layer may be used to ensure that no significant drop will occur in accuracy. The linear

quantization method can greatly lower computation complexity, which can further facilitate the deployment of the convolutional neural network on the special-purpose processing device.

[0075] The backward pass process has been introduced above with reference to a convolutional layer 400. It would be appreciated that the backward pass of the entire neural network can be stacked by a plurality of similar processes. For example, the backward output of the k+1-th layer is provided to the k-th layer to serve as a backward input of the k-th layer; and the parameter of each layer is updated layer by layer. In the convolutional neural network 200 of Fig. 2, if the convolutional layer 204 and the pooling layer 206 are combined for implementation, the backward output of the convolutional layer 204 can be determined by the architecture of the convolutional layer 300 (including a sub-layer 314). Then, the backward output is provided to the input layer 202, to finally finish updating all parameters of the neural network 200, thereby completing an iteration of a mini-batch. Iteratively completing iterations of all mini-batches in the training set may be referred to as finishing a full iteration of the data set, which is also known as epoch. After a plurality of epochs, if the training result satisfies the predefined threshold condition, the training is complete. For example, the threshold condition can be a predefined number of epochs or a predefined accuracy.

[0076] Additionally, it would be appreciated that it is not necessary to apply the adaptive updating method in each iteration. For example, the adaptive updating method may be performed once after a plurality of iterations. Moreover, the frequency for applying the adaptive updating method may vary for different quantities. For example, the adaptive updating method may be applied more frequently for the gradients, because the gradients tend to fluctuation more extensively.

Model Training

[0077] Fig. 5 illustrates a flowchart of a method 500 for a convolutional neural network in accordance with implementations of the subject matter described herein. The method 500 may be performed on the special-purpose processing device 106 as shown in Fig. 1. As described above, the special-purpose processing device 106 may be an FPGA or ASIC, for example.

[0078] At 502, an input to a convolutional layer of the neural network is received. As described above, the input may be received from the previous layer, or may be an input image for the neural network. The input may correspond to samples of a mini-batch in the training set.

[0079] At 504, parameters of the convolutional layer are read from a memory unit 108 of the special-purpose processing device 106, where the parameters are stored in the memory unit 108 of the special-purpose processing device 106 in a first fixed-point format and have a predefined bit-width. The parameters may represent either weight parameters or bias parameters of the convolutional layer, or may represent both the weight parameters and the bias parameters. Generally, the bit-width of the first fixed-point format is smaller than the floating-point number to reduce the memory space of the memory unit 108.

[0080] At 506, the output of the convolutional layer is computed by fixed-point operations based on the input of the convolutional layer and the read parameters. In some implementations, the convolutional operations may be performed on the input and the parameters of the convolutional layer to obtain an intermediate output, which is normalized to obtain a normalized output. The normalization only includes respective fixed-point operations. For example, the normalization may be implemented by the IBN layer 316 as shown in Fig. 3.

[0081] In some implementations, in order to reduce the bit-width of the first fixed-point format while maintaining the model accuracy, the scaling factors of the parameters above are adaptively updated. For example, a backward input to the convolutional layer is received at the output of the convolutional layer, where the backward input is a gradient of the loss function of the neural network with respect to the output of the convolutional layer. Based on the backward input, the gradient of the loss function of the neural network with respect to parameters of the convolutional layer is computed. The parameters in the first fixed-point format may be updated based on the gradient of the loss function of the neural network with respect to parameters. The scaling factors of the first fixed-point format may be updated based on the updated parameter range. For example, the fixed-point format of the parameters may be updated by the method described above with reference to quantization.

[0082] The updated parameters may be stored in the memory unit 108 of the special-purpose processing device 106 to be read at the next iteration. In addition, it is not necessary to update the parameter format in each iteration. Instead, the fixed-point format of the parameters may be updated at a certain frequency. In some implementations, updating parameters only include respective fixed-point operations, which may be implemented by a fixed-point ADAM optimization method, for example.

[0083] In some implementations, the gradient of the loss function with respect to the parameters may be first converted to a second fixed-point format for updating parameters

in the first fixed-point form. The first fixed-point format may be identical to or different from the second fixed-point format. The conversion method can be carried out by a linear quantization method. In other words, the gradient of the loss function of the neural network with respect to parameters may be converted to the second fixed-point format by a linear quantization method. Then, the parameters in the first fixed-point format may be updated based on the gradient in the second fixed-point format. In some implementations, the scaling factors of the second fixed-point format may be updated based on the range of the gradient of the loss function with respect to the parameters. As described above, the linear quantization method has a lower computation complexity and the performance will not be substantially degraded, because the scaling factor updating method is employed in the implementations of the subject matter described herein.

[0084] In some implementations, computing the output of the convolutional layer further comprises: converting the normalized output to a normalized output in a third fixed-point format, where the scaling factors of the third fixed-point format may be updated based on the range of the normalized output in the third fixed-point format. As shown in Fig. 3, the output of the IBN sub-layer 316 may be provided to the quantization layer 318, which can convert the normalized output of the IBN sub-layer 316 to a normalized output in a second fixed-point format. The scaling factors of the second fixed-point format can be updated depending on various factors. For example, the updating method may be configured to be carried out after a given number of iterations, which updating method may be the method described in the Quantization section above.

[0085] In some implementations, the method further comprises: receiving a backward input to the convolutional layer at the output of the convolutional layer, which backward input is a gradient of the loss function of the neural network with respect to the output of the convolutional layer. Then, the intermediate backward output is obtained based on the normalized backward gradient operations. In other words, the gradient of the loss function with respect to the convolution above is computed based on the backward input. For example, the backward gradient operations of the IBN gradient sub-layer 416 corresponds to normalization of the IBN sub-layer 416 as shown in Fig. 4. The backward gradient operations can be performed on the IBN gradient sub-layer 416 to get an intermediate backward output. Subsequently, the intermediate backward output is converted to a fourth fixed-point format and the scaling factors of the fourth fixed-point format can be updated based on the range of the intermediate backward output. For example, the scaling factors of the fourth fixed-point format may be updated according to the updating method described

above with reference to quantization.

[0086] It would be appreciated that, although the method 500 describes one convolutional layer, the training process of the entire neural network may be stacked by the process of method 500 as described above with reference to Figs. 3 and 4.

Another Example Implementations of Special-Purpose Processing Device

[0087] Fig. 1 illustrates an example implementation of the special-purpose processing device 106. In the example of Fig. 1, the special-purpose processing device 106 includes a memory unit 108 for storing parameters of the neural network, and a processing unit 110 for reading the stored parameters from the memory unit 108 and using the parameters to process the input.

[0088] Fig. 6 illustrates a block diagram of a further example implementation of the special-purpose processing device 106. As described above, the special-purpose processing device 106 may be an FPGA or ASIC, for example.

[0089] In this example, the special-purpose processing device 106 includes a memory module 602 configured to store parameters of the convolutional layer of the neural network in a first fixed-point format, where the parameters in the first fixed-point format have a predefined bit-width. It would be appreciated that the memory module 602 is similar to the memory unit 108 of Fig. 1 in terms of functionality and both of them may be implemented using the same or different techniques or processes. Generally, the bit-width of the first fixed-point format is smaller than the floating-point numbers to reduce memory space of the memory module 602.

[0090] The special-purpose processing device 106 further includes an interface module 604 configured to receive an input to the convolutional layer. In some implementations, the interface module 604 may be used for processing various inputs and outputs between various layers of the neural network. The special-purpose processing device 106 further includes a data access module 606 configured to read parameters of the convolutional layer from the memory module 602. In some implementations, the data access module 606 may interact with the memory module 602 to process the access to the parameters of the neural network. The special-purpose processing device 106 may further include a computing module 608 configured to compute, based on the input of the convolutional layer and the read parameters, the output of the convolutional layer by a fixed-point operation.

[0091] In some implementations, the interface module 604 is further configured to receive a backward input to the convolutional layer at the output of the convolutional layer, where the backward input is a gradient of the loss function of the neural network with respect to

the output of the convolutional layer. In addition, the computing module 608 is further configured to compute a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer based on the backward input; and update parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters, where the scaling factors of the first fixed-point format can be updated based on the range of the updated parameters.

[0092] In some implementations, updating parameters only includes respective fixed-point operations.

[0093] In some implementations, the computing module 608 is further configured to convert the gradient of the loss function of the neural network with respect to the parameters to a second fixed-point format by a linear quantization method, where the scaling factors of the second fixed-point format can be updated based on the gradient of the loss function with respect to the parameters; and update the parameters based on the gradient in the second fixed-point format.

[0094] In some implementations, the computing module 608 is further configured to normalize a convolution of the input of the convolutional layer and the parameters to obtain a normalized output, where the normalization only includes respective fixed-point operations.

[0095] In some implementations, the computing module 608 is further configured to convert the normalized output to a normalized output in a third fixed-point format, where the scaling factors of the third fixed-point format can be updated based on the range of the normalized output in the third fixed-point format.

[0096] In some implementations, the interface module 604 is further configured to obtain a backward input to the convolutional layer at the output of the convolutional layer, where the backward input is a gradient of the loss function of the neural network with respect to the output of the convolutional layer. Additionally, the computing module 608 may be configured to compute the gradient of the loss function with respect to the convolution based on the backward input; and convert the gradient of the loss function with respect to the convolution to a fourth fixed-point format, where the scaling factor of the fourth fixed-point format can be updated based on the range of the gradient of the loss function with respect to the convolution.

Testing and Performance

[0097] The following section will introduce the important factors that affect the final prediction accuracy of the training model of the neural network in accordance with

implementations of the subject matter described herein. The factors comprise the batch normalization (BN) scheme, bit-width of the primal parameters, and bit-width of gradients. The effects of the factors are evaluated in turn by applying them separately on the binary neural network (BNN). Finally, all these factors are combined to obtain a neural network model.

[0098] In the following test, a data set CIRFA-30 is used, where the data set CIRFA-30 is an image classification benchmark with 60K 32×32 RGB tiny images. It consists of 10 classes object, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class has 5K training images and 1K test images. To evaluate the model fitting capability and training efficiency, three networks with different size are designed by stacking basic structural modules of the neural network shown in Figs. 3 and 4, including Model-S (Small), Model-M (Medium) and Model-L (Large). The overall network structure is illustrated in Figs. 7 and 8.

[0099] Fig. 7 illustrates a block diagram of a forward pass of the convolutional neural network 700 in accordance with an implementation of the subject matter described herein and Fig. 8 illustrates a block diagram of a backward pass of the convolutional neural network 800 in accordance with an implementation of the subject matter described herein.

[00100] In the convolutional neural networks 700 and 800, all convolution kernels are 3×3, and the number of output channels in the first convolutional layer is 32, 64, and 128. Table 1 lists the number of parameters and the number of multiply-accumulate operations (MACs) in the three models. In Figs. 7 and 8, "×2 (4 or 8)" in layer C21 means the number of output channels in layer C21 is two times (4X or 8X) as much as the number in layers C11 and C12. Additionally, S represents same padding, V represents valid padding, MP indicates maximum pooling, C indicates convolutional layer, and FC indicates the fully-connected layer. The specific structure of each layer in Figs. 7 and 8 is not shown, which can be inspected from Figs. 3 and 4. It is noted that the loss function layer is computed in the floating-point domain in both forward pass and backward pass.

Table 1

Model	Parameter number	MAC number
Model-S	0.58M	39.82M
Model-M	2.32M	156.60M
Model-L	9.29M	623.74M

[00101] In all of the experiments, 50K training images and a mini-batch size of 200 are

given. Additionally, there are 37,500 iterations and 150 epochs in total. Because each epoch means one training that uses all samples in the training set and each iteration uses samples of one batch for training, each epoch has 250 iterations accordingly. Furthermore, in the experiments, by using the fixed-point ADAM optimization method or standard ADAM optimization method and setting the initial learning rate as 2^{-6}, the learning rate will be decreased by a factor of 2^{-4} every 50 epochs.

[00102] Now the effect of different normalization schemes on prediction accuracy is evaluated, including standard floating-point BN and IBN output of different bit-widths. Here the primal parameters and all gradients are kept in the floating-point format and the standard ADAM algorithm is used to optimize the network. Note that the scaling factor updating algorithm described above will be performed on the IBN output every 1,125 iterations (3% of total iterations), and the threshold of the scaling factor updating algorithm is set to be 0.01%.

[00103] After testing, the accuracy loss of the neural network is quite stable with respect to the bit-width of the IBN output, as low as 6 bits. If the bit-width of the IBN output continues to decrease, the accuracy will suffer a cliff-off drop.

[00104] To evaluate the effects resulted from bit-width of storage parameters, experiments are conducted with floating-point gradients. In this case, the standard ADAM algorithm is used to update the parameters and the updated parameters are stored in a fixed-point format. The testing shows that 8-bit parameters are sufficient for maintaining performance and the bit-width lower than 8-bit will bring significant accuracy loss. Furthermore, the scaling factors are updated to maintain the values within a normal range. On the contrary, static scaling factor imposes too strong regularization on model parameters and fails to converge when the bit-width is lower than 8-bit.

[00105] Furthermore, the effect of the gradient bit-width is also evaluated. The gradients are more unstable than the parameters, which shows that the scaling factors of the gradients should be updated more frequently. In some implementations, the update occurs every 375 iterations (1% of total iterations) and the fixed-point ADAM method is employed. In the testing, the primal parameters are set with floating-point values. It can be seen from the testing that the prediction accuracy decreases very slowly when the bit-width of the gradient is reduced. The prediction accuracy also suffers a cliff-off drop when the bit-width of the gradient is lower than 12 bits, which is similar to the effect of the IBN output and the parameter bit-width. Therefore, a cliff-off drop will occur when the IBN output, parameter bit-width, and the gradient bit-width is lower than the threshold.

[00106] The test is performed by combining the three effects, i.e., the neural network is implemented to substantially involve fixed-point computations only. In this way, the result in Table 2 can be obtained.

Table 2

Method	Weight (bit)	Running Weight (bit)	Activation (bit)	Gradient (bit)	Parameter Number	Relative Storage	CIFAR-10 error rate
Model-L	24	1	1	24	9.29 M	2.0	10.30%
	16	1	1	16	9.29 M	1.3	10.51%
	12	1	1	12	9.29 M	1.0	11.48%

[00107] Because the parameters are stored in a memory-on-chip (e.g., memory unit 108) of the special-purpose processing device 106, the relative storage is characterized by a product of the parameter number and the bits of the primal weight. It can be seen from Table 2 that a comparable accuracy with a larger bit-width can be obtained when the bit-width of the primal weight is 12 and the bit-width of the gradient is also 12. As the weight bit-width decreases, the storage will be substantially decreased. Therefore, the training solution for the neural network according to implementations of the subject matter described herein can lower the storage while maintaining computation accuracy.

[00108] As shown in Table 2, the method can achieve comparable result with the state-of-art works (not shown) when the bit-width of each of the primal weight and the gradient is 12. However, compared with the prior art, the method dramatically reduces the storage and significantly improves system performance.

Example Implementations

[00109] Several example implementations of the subject matter described herein are listed below.

[00110] In accordance with implementations of the subject matter described herein, there is provided a special-purpose processing device. The special-purpose processing device comprises a memory unit configured to store parameters of a layer of a neural network in a

first fixed-point format, the parameters in the first fixed-point format having a predefined bit-width; a processing unit coupled to the memory unit and configured to perform acts including: receiving an input to the layer; reading the parameters of the layer from the memory unit; and computing, based on the input of the layer and the read parameters, an output of the layer through a fixed-point operation.

[00111] In some implementations, the layer of the neural network includes a convolutional layer.

[00112] In some implementations, the acts further include: receiving a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of a loss function of the neural network with respect to the output of the convolutional layer; computing, based on the backward input, a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer; and updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, a scaling factor of the first fixed-point format being updatable based on a range of the updated parameters.

[00113] In some implementations, updating the parameters only include a respective fixed-point operation.

[00114] In some implementations, updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer comprises: converting the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer into a second fixed-point format by a linear quantization method, the scaling factor of the second fixed-point format being updatable based on a range of the gradient of the loss function with respect to the parameters of the convolutional layer; and updating the parameters in the first fixed-point format based on the gradient in the second fixed-point format.

[00115] In some implementations, computing the output of the layer comprises: normalizing a convolution of the input of the convolutional layer and the parameters in the first fixed-point format to obtain a normalized output, the normalizing only including a respective fixed-point operation.

[00116] In some implementations, computing the output of the convolutional layer further comprises: converting the normalized output into the normalized output in a third fixed-point format, a scaling factor of the third fixed-point format being updatable based on a range of the normalized output in the third fixed-point format.

[00117] In some implementations, the acts further include: obtaining a backward input to

the convolutional layer at an output of the convolutional layer, the backward input being a gradient of the loss function of the neural network with respect to the output of the convolutional layer; computing, based on the backward input, a gradient of the loss function with respect to the convolution; and converting the gradient of the loss function with respect to a convolution into a fourth fixed-point format, a scaling factor of the fourth fixed-point format being updatable based on a range of the gradient of the loss function with respect to the convolution.

[00118] In some implementations, the special-purpose processing device is a field programmable gate array (FPGA), an application-specific integrated circuit (ASIC), a processor having a customized processing unit, or a graphics processing unit (GPU).

[00119] In accordance with implementations of the subject matter described herein, there is provided a method executed by a special-purpose processing device including a memory unit and a processing unit. The method comprises receiving an input to a layer of a neural network; reading parameters of the layer from the memory unit of the special-purpose processing device, the parameters being stored in the memory unit in a first fixed-point format and having a predefined bit-width; and computing, by the processing unit and based on the input of the layer and the read parameters, an output of the layer through a fixed-point operation.

[00120] In some implementations, the layer of the neural network includes a convolutional layer.

[00121] In some implementations, the method further comprises: receiving a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of a loss function of the neural network with respect to the output of the convolutional layer; computing, based on the backward input, a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer; and updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, a scaling factor of the first fixed-point format being updatable based on a range of the updated parameters.

[00122] In some implementations, updating the parameters only include a respective fixed-point operation.

[00123] In some implementations, updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer comprises: converting the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer into a second

fixed-point format by a linear quantization method, the scaling factor of the second fixed-point format being updatable based on a range of the gradient of the loss function with respect to the parameters of the convolutional layer; and updating the parameters in the first fixed-point format based on the gradient in the second fixed-point format.

[00124] In some implementations, computing the output of the layer comprises: normalizing a convolution of the input of the convolutional layer and the parameters in the first fixed-point format to obtain a normalized output, the normalizing only including a respective fixed-point operation.

[00125] In some implementations, computing the output of the convolutional layer further comprises: converting the normalized output into the normalized output in a third fixed-point format, a scaling factor of the third fixed-point format being updatable based on a range of the normalized output in the third fixed-point format.

[00126] In some implementations, the method further comprises: obtaining a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of the loss function of the neural network with respect to the output of the convolutional layer; computing, based on the backward input, a gradient of the loss function with respect to the convolution; and converting the gradient of the loss function with respect to a convolution into a fourth fixed-point format, a scaling factor of the fourth fixed-point format being updatable based on a range of the gradient of the loss function with respect to the convolution.

[00127] In some implementations, the special-purpose processing device is a field programmable gate array (FPGA), an application-specific integrated circuit (ASIC), a processor having customized processing units or a graphics processing unit (GPU).

[00128] In accordance with implementations of the subject matter described herein, there is provided a special-purpose processing device. The special-purpose processing device comprises: a memory module configured to store parameters of a layer of a neural network in a first fixed-point format, the parameters in the first fixed-point format having a predefined bit-width; an interface module configured to receive an input to the layer; a data access module configured to read the parameters of the layer from the memory module; and a computing module configured to compute, based on the input of the layer and the read parameters, an output of the layer through a fixed-point operation.

[00129] In some implementations, the layer of the neural network includes a convolutional layer.

[00130] In some implementations, the interface module is further configured to receive a

backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of a loss function of the neural network with respect to the output of the convolutional layer; the computing module is further configured to: compute, based on the backward input, a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, and update the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, a scaling factor of the first fixed-point format being updatable based on a range of the updated parameters.

[00131] In some implementations, updating the parameters only include a respective fixed-point operation.

[00132] In some implementations, the computing module is further configured to: convert the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer into a second fixed-point format by a linear quantization method, the scaling factor of the second fixed-point format being updatable based on a range of the gradient of the loss function with respect to the parameters of the convolutional layer; and update the parameters in the first fixed-point format based on the gradient in the second fixed-point format.

[00133] In some implementations, the computing module is further configured to: normalize a convolution of the input of the convolutional layer and the parameters in the first fixed-point format to obtain a normalized output, the normalizing only including a respective fixed-point operation.

[00134] In some implementations, the computing module is further configured to: convert the normalized output into the normalized output in a third fixed-point format, a scaling factor of the third fixed-point format being updatable based on a range of the normalized output in the third fixed-point format.

[00135] In some implementations, the interface module is further configured to: obtain a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of the loss function of the neural network with respect to the output of the convolutional layer; compute, based on the backward input, a gradient of the loss function with respect to the convolution; and convert the gradient of the loss function with respect to a convolution into a fourth fixed-point format, a scaling factor of the fourth fixed-point format being updatable based on a range of the gradient of the loss function with respect to the convolution.

[00136] In some implementations, the special-purpose processing device is a field

programmable gate array (FPGA), an application-specific integrated circuit (ASIC), a processor having customized processing units or a graphics processing unit (GPU).

[00137] The above described functions in the text can be performed, at least in part, by one or more hardware logic components. For example, and without limitation, examples types of hardware logic components that can be used include Field-Programmable Gate Arrays (FPGAs), Application-specific Integrated Circuits (ASICs), Application-specific Standard Product (ASSP), System-on-a-chip systems (SOC), Complex Programmable Logic Devices (CPLD), and the like.

[00138] Further, while operations are depicted in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Likewise, while several specific implementation details are contained in the above discussions, these should not be construed as limitations on the scope of the subject matter described herein, but rather as descriptions of features that may be specific to particular implementations. Certain features that are described in the context of separate implementations may also be implemented in combination in a single implementation. Conversely, various features that are described in the context of a single implementation may also be implemented in multiple implementations separately or in any suitable sub-combination.

[00139] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter specified in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims.

CLAIMS

1. A special-purpose processing device, comprising:

a memory unit configured to store parameters of a layer of a neural network in a first fixed-point format, the parameters in the first fixed-point format having a predefined bit-width;

a processing unit coupled to the memory unit and configured to perform acts including:

receiving an input to the layer;

reading the parameters of the layer from the memory unit; and

computing, based on the input of the layer and the read parameters, an output of the layer through a fixed-point operation.

2. The special-purpose processing device of claim 1, wherein the layer includes a convolutional layer, and wherein the acts further include:

receiving a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of a loss function of the neural network with respect to the output of the convolutional layer;

computing, based on the backward input, a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer; and

updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, a scaling factor of the first fixed-point format being updatable based on a range of the updated parameters.

3. The special-purpose processing device of claim 2, wherein updating the parameters only include a respective fixed-point operation.

4. The special-purpose processing device of claim 2, wherein updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer comprises:

converting the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer into a second fixed-point format by a linear quantization method, the scaling factor of the second fixed-point format being updatable based on a range of the gradient of the loss function with respect to the parameters of the convolutional layer; and

updating the parameters in the first fixed-point format based on the gradient in the second fixed-point format.

5. The special-purpose processing device of claim 1, wherein the layer includes a convolutional layer and computing the output of the layer comprises:

normalizing a convolution of the input of the convolutional layer and the parameters in the first fixed-point format to obtain a normalized output, the normalizing only including a respective fixed-point operation.

6. The special-purpose processing device of claim 5, wherein computing the output of the convolutional layer further comprises:

converting the normalized output into the normalized output in a third fixed-point format, a scaling factor of the third fixed-point format being updatable based on a range of the normalized output in the third fixed-point format.

7. The special-purpose processing device of claim 5, wherein the acts further include:

obtaining a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of the loss function of the neural network with respect to the output of the convolutional layer;

computing, based on the backward input, a gradient of the loss function with respect to the convolution; and

converting the gradient of the loss function with respect to a convolution into a fourth fixed-point format, a scaling factor of the fourth fixed-point format being updatable based on a range of the gradient of the loss function with respect to the convolution.

8. The special-purpose processing device of claim 1, wherein the special-purpose processing device is a field programmable gate array (FPGA), an application-specific integrated circuit (ASIC), a processor having a customized processing unit, or a graphics processing unit (GPU).

9. A method executed by a special-purpose processing device including a memory unit and a processing unit, the method comprising:

receiving an input to a layer of a neural network;

reading parameters of the layer from the memory unit of the special-purpose processing device, the parameters being stored in the memory unit in a first fixed-point format and having a predefined bit-width; and

computing, by the processing unit and based on the input of the layer and the read parameters, an output of the layer through a fixed-point operation.

10. The method of claim 9, wherein the layer includes a convolutional layer, and the method further comprises:

receiving a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of a loss function of the neural network with respect to the output of the convolutional layer;

computing, based on the backward input, a gradient of the loss function of the neural network with respect to the parameters of the convolutional layer; and

updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer, a scaling factor of the first fixed-point format being updatable based on a range of the updated parameters.

11. The method of claim 10, wherein updating the parameters only include a respective fixed-point operation.

12. The method of claim 10, wherein updating the parameters in the first fixed-point format based on the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer comprises:

converting the gradient of the loss function of the neural network with respect to the parameters of the convolutional layer into a second fixed-point format by a linear quantization method, the scaling factor of the second fixed-point format being updatable based on a range of the gradient of the loss function with respect to the parameters of the convolutional layer; and

updating the parameters in the first fixed-point format based on the gradient in the second fixed-point format.

13. The method of claim 9, wherein the layer includes a convolutional layer and computing the output of the layer comprises:

normalizing a convolution of the input of the convolutional layer and the parameters in the first fixed-point format to obtain a normalized output, the normalizing only including a respective fixed-point operation.

14. The method of claim 13, wherein computing the output of the convolutional layer further comprises:

converting the normalized output into the normalized output in a third fixed-point format, a scaling factor of the third fixed-point format being updatable based on a range of the normalized output in the third fixed-point format.

15. The method of claim 13, further comprising:

obtaining a backward input to the convolutional layer at an output of the convolutional layer, the backward input being a gradient of the loss function of the neural

network with respect to the output of the convolutional layer;

computing, based on the backward input, a gradient of the loss function with respect to the convolution; and

converting the gradient of the loss function with respect to a convolution into a fourth fixed-point format, a scaling factor of the fourth fixed-point format being updatable based on a range of the gradient of the loss function with respect to the convolution.

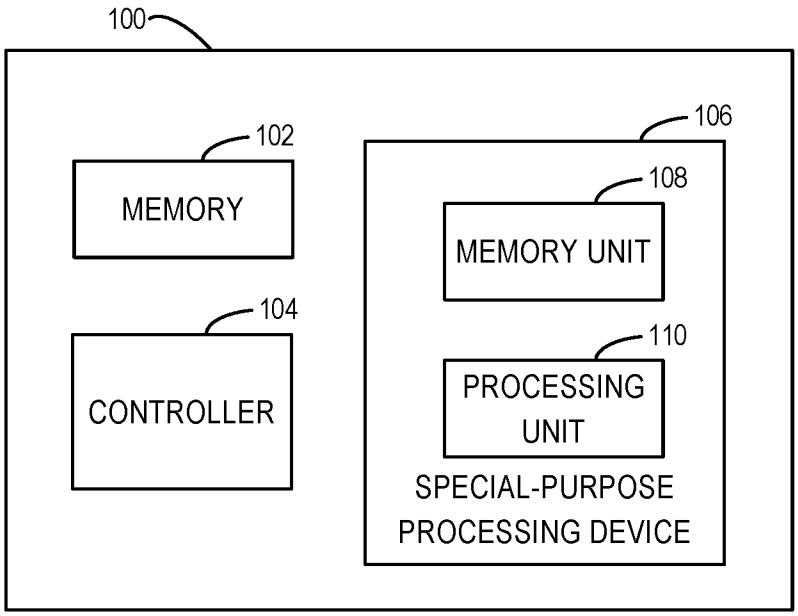
100
102
MEMORY
104
CONTROLLER
106
108
MEMORY UNIT
110
PROCESSING UNIT
SPECIAL-PURPOSE PROCESSING DEVICE

FIG. 1

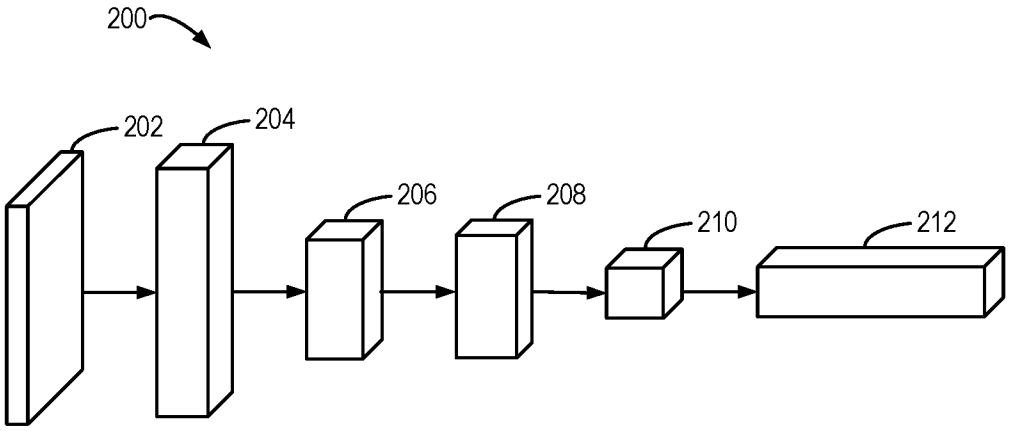
200
202
204
206
208
210
212

FIG. 2

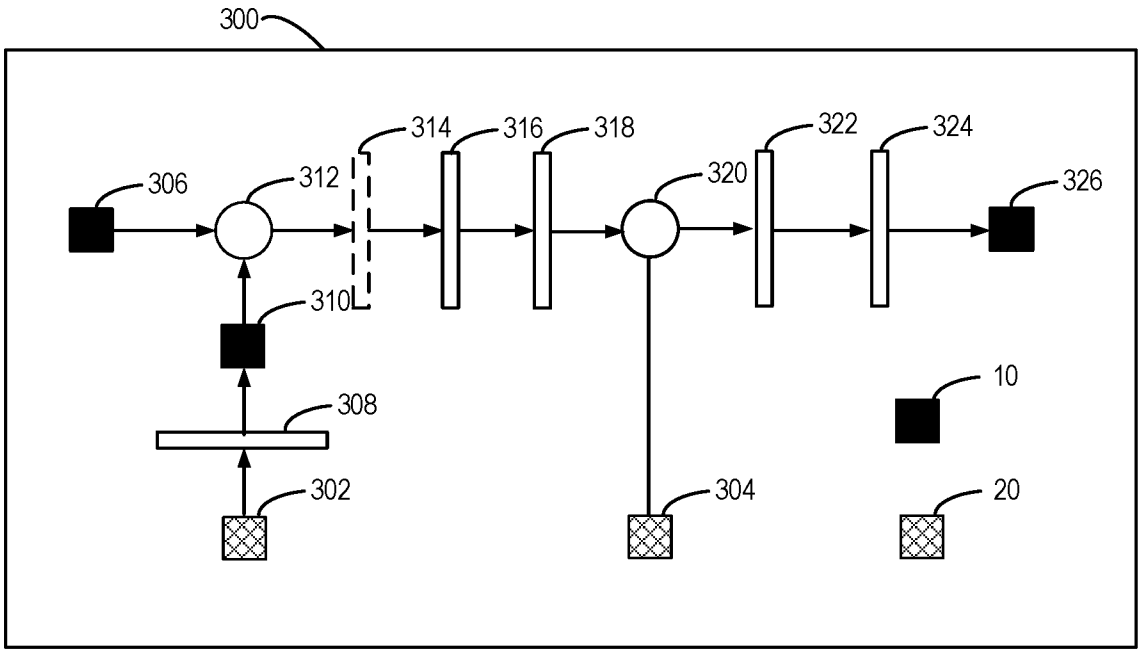
300
306
312
314
316
318
320
322
324
326
310
308
302
304
10
20

FIG. 3

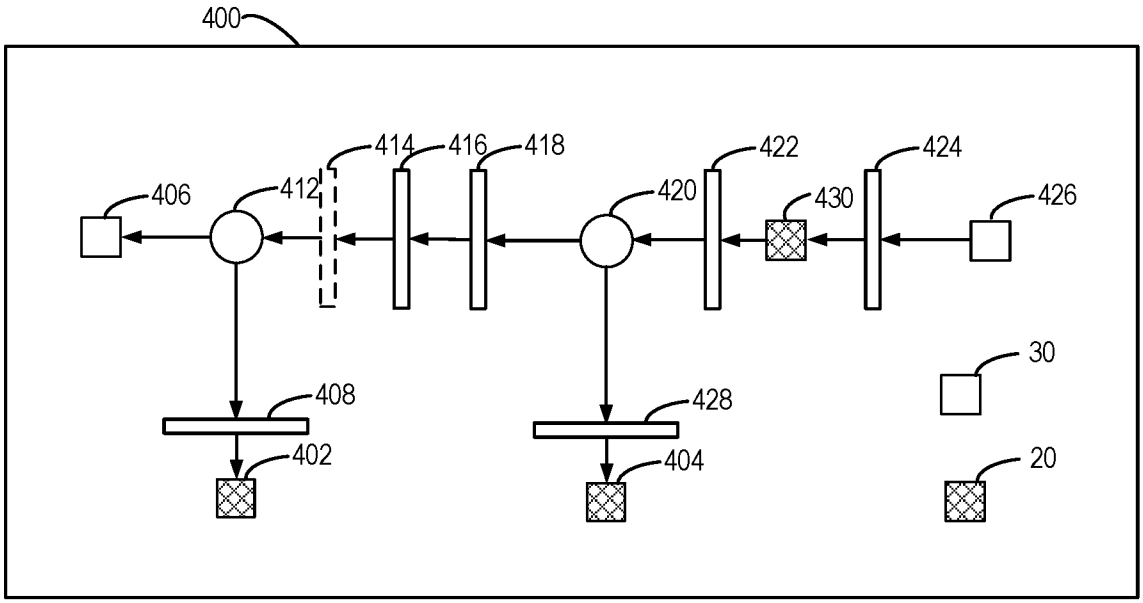
400
406
412
414
416
418
420
422
424
426
430
408
402
428
404
30
20

FIG. 4

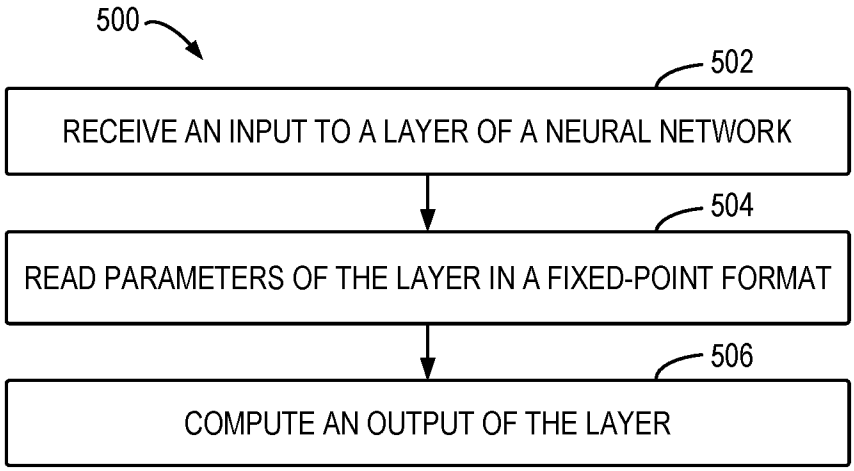
500
502
RECEIVE AN INPUT TO A LAYER OF A NEURAL NETWORK
504
READ PARAMETERS OF THE LAYER IN A FIXED-POINT FORMAT
506
COMPUTE AN OUTPUT OF THE LAYER

FIG. 5

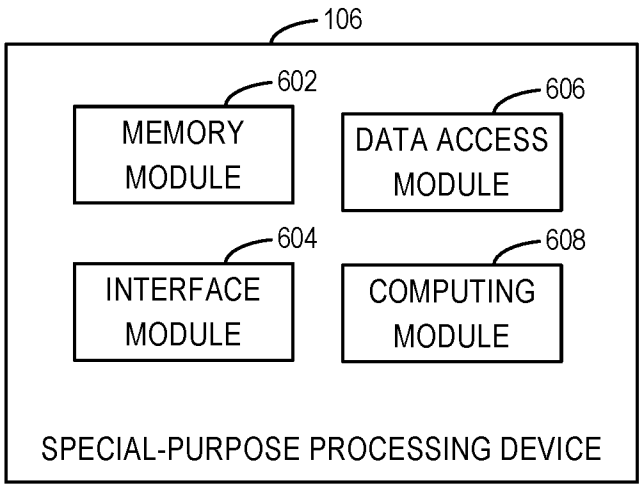
106
602
MEMORY MODULE
606
DATA ACCESS MODULE
604
INTERFACE MODULE
608
COMPUTING MODULE
SPECIAL-PURPOSE PROCESSING DEVICE

FIG. 6

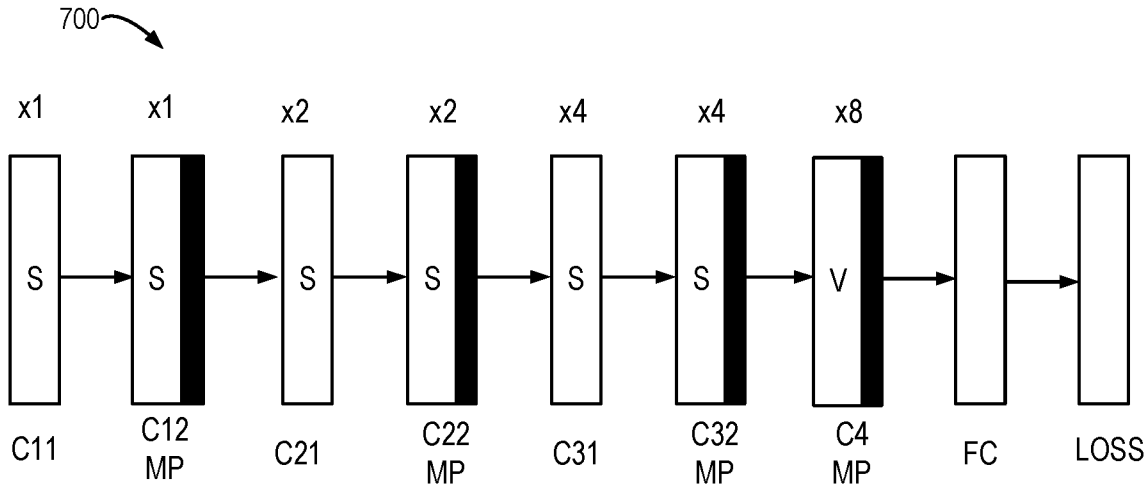
700
x1
x1
x2
x2
x4
x4
x8
S
S
S
S
S
S
V
C11
C12
MP
C21
C22
MP
C31
C32
MP
C4
MP
FC
LOSS

FIG. 7

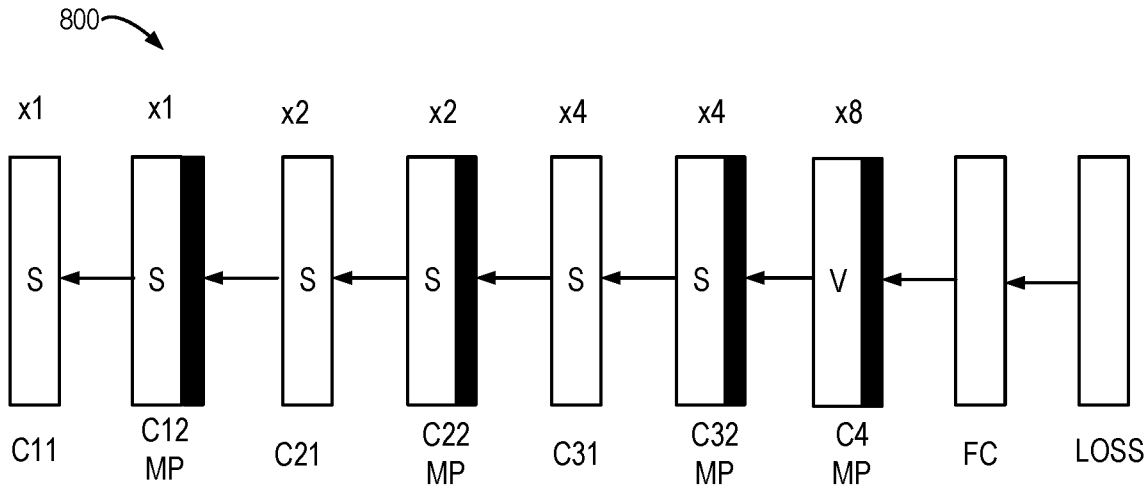
800
x1
x1
x2
x2
x4
x4
x8
S
S
S
S
S
S
V
C11
C12
MP
C21
C22
MP
C31
C32
MP
C4
MP
FC
LOSS

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2018/014303

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06N3/063 G06N3/08
ADD. G06N3/04

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	US 2016/328647 A1 (LIN DEXU [US] ET AL) 10 November 2016 (2016-11-10) abstract; claims 1-29; figures 4,5 paragraph [0010] - paragraph [0088] -----	1-15
X	SUYOG GUPTA ET AL: "Deep Learning with Limited Numerical Precision", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 9 February 2015 (2015-02-09), XP080677454, A page 1 - page 11, right-hand column, paragraph 1 -----	1,8 2-7,9-15
	-/--	

[X] Further documents are listed in the continuation of Box C.

[X] 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
17 May 2018	28/05/2018

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	Volkmer, Markus

1

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

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2018/014303

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Philipp Gysel ET AL: "HARDWARE-ORIENTED APPROXIMATION OF CONVOLUTIONAL NEURAL NETWORKS", , 11 April 2016 (2016-04-11), XP055398866, Retrieved from the Internet: URL:https://arxiv.org/pdf/1604.03168v1.pdf [retrieved on 2017-08-16]	1,8
A	page 1 - page 6	2-7,9-15

X	JIANTAO QIU ET AL: "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network", PROCEEDINGS OF THE 2016 ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS, FPGA '16, 1 January 2016 (2016-01-01), pages 26-35, XP055423746, New York, New York, USA DOI: 10.1145/2847263.2847265 ISBN: 978-1-4503-3856-1	1,8
A	page 26 - page 35, left-hand column, paragraph 3	2-7,9-15

X	DE 10 2015 007943 A1 (INTEL CORP [US]) 28 January 2016 (2016-01-28)	1,8
A	abstract; claims 1-22; figures 12-14 paragraph [0004] - paragraph [0125]	2-7,9-15

X,P	CHEN XI ET AL: "FxpNet: Training a deep convolutional neural network in fixed-point representation", 2017 INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), IEEE, 14 May 2017 (2017-05-14), pages 2494-2501, XP033112353, DOI: 10.1109/IJCNN.2017.7966159 [retrieved on 2017-06-30] the whole document	1-15

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2018/014303

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2016328647 A1	10-11-2016	CN 107646116 A	30-01-2018
		EP 3295382 A1	21-03-2018
		US 2016328647 A1	10-11-2016
		WO 2016182659 A1	17-11-2016
DE 102015007943 A1	28-01-2016	CN 105320495 A	10-02-2016
		DE 102015007943 A1	28-01-2016
		TW 201617977 A	16-05-2016
		TW 201734894 A	01-10-2017
		US 2016026912 A1	28-01-2016