Neurocomputing 407 (2020) 439–453

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Residue Number System-Based Solution for Reducing the Hardware Cost of a Convolutional Neural Network

N.I. Chervyakov[a], P.A. Lyakhov[a,b,*], M.A. Deryabin[a], N.N. Nagornov[a], M.V. Valueva[a], G.V. Valuev[a]

[a] Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia
[b] Department of Automation and Control Processes, Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russia

ARTICLE INFO

Article history:
Received 30 June 2019
Revised 23 September 2019
Accepted 7 April 2020
Available online 18 May 2020

Keywords:
Image recognition
Convolutional neural networks
Residue number system
Quantization noise
FPGA

ABSTRACT

Convolutional neural networks (CNNs) represent deep learning architectures that are currently used in a wide range of applications, including computer vision, speech recognition, time series analysis in finance, and many others. At the same time, CNNs are very demanding in terms of the hardware and time cost of a computing system, which considerably restricts their practical use, e.g., in embedded systems, real-time systems, and mobile volatile devices. The goal of this paper is to reduce the resources required to build and operate CNNs. To achieve this goal, a CNN architecture based on Residue Number System (RNS) and the new Chinese Remainder Theorem with fractions is proposed. The new architecture gives an efficient solution to the main problem of RNSs associated with restoring the number from its residues, which determines the main contribution to the CNN structure. In accordance with the results of hardware simulation on Kintex7 xc7k70tfbg484-2 FPGA, the use of RNS in the convolutional layer of a neural network reduces hardware cost by 32.6% compared to the traditional approach based on the binary number system. In addition, the use of the proposed hardware-software architecture reduces the average image recognition time by 37.06% compared to the software implementation.

1. Introduction

The modern development of science and technology is characterized by an ever-growing volume of data that is continuously collected, stored and processed. Recent progress in the field of artificial intelligence has opened up new possibilities for the automation of these processes, raising computer vision, speech recognition, bioinformatics, analysis of medical images and many others fields of science to a qualitatively new level of development. Processing of visual information occupies a special place among machine learning tasks, due to the large amount of processed data and the high complexity of biological visual systems that perform a similar function [1]. The range of applications requiring the use of artificial intelligence methods for image processing is constantly expanding and currently includes personal identification [2], scene recognition [3], processing of information from external sensors in unmanned land and aircraft vehicles [4], medical diagnostics [5] and others. At the same time, the constantly growing practical need for increasing the quality of visual information and the speed of its processing, combined with the high complexity of machine learning systems, requires the development of new and specialized computer architectures to satisfy the demands of science and technology in this area.

Convolutional neural networks (CNNs) represent deep learning architectures that are currently used in a wide range of applications, including computer vision [6], speech recognition [7], identification of albuminous sequences in bioinformatics [8], production control [9], time series analysis in finance [10], and many others. At the same time, CNNs are very demanding in terms of the hardware and time cost of a computing system, which considerably restricts their practical use, e.g., in embedded systems, real-time systems, and mobile volatile devices [11].

The idea of using artificial neural networks for visual information processing was proposed in [12] in order to perform the automatic recognition of handwritten numbers. The architecture proposed in that paper was called the CNN. Its main peculiarity is the combination of convolutional layers that implement the extraction of visual features and a multilayer perceptron that implements the recognition operation using the results of convolutions. The evolution of this scientific idea and the advancement of

* Corresponding author.
E-mail addresses: k-fmf-primath@stavsu.ru (N.I. Chervyakov), ljahov@mail.ru (P.A. Lyakhov), maderiabin@ncfu.ru (M.A. Deryabin), sparta1392@mail.ru (N.N. Nagornov), mriya.valueva@mail.ru (M.V. Valueva), elasgreece92@mail.ru (G.V. Valuev).

https://doi.org/10.1016/j.neucom.2020.04.018
0925-2312/

computing technology have led to the fact that currently the theory of CNNs and its applications are developing along the path of an extensive increase in the number of CNN layers, which causes a high computational complexity of the implementation of such systems. For example, consider the network architecture [13], which showed the best image recognition result for the ImageNet base in 2010. It contains about 650 000 neurons and 60 million tunable parameters, requiring 27 GBs of disk space for training. An original solution by Google that demonstrated the best image recognition result of the ImageNet base in 2014 was presented in the paper [14]. For image recognition, this CNN performs more than one and a half billion computational operations, which motivated Google to develop a special tensor processor for its optimal performance [15]. Summarizing the aforesaid, it can be concluded that modern CNN architectures are very resource intensive, which restricts their applicability in practice. Nowadays, there is no unified approach to reducing the hardware cost of CNNs in practice.

The goal of this paper is to reduce the resources required to build and operate CNNs. To achieve this goal, a new CNN architecture is proposed, in which convolution layers have a hardware implementation on FPGA using the residue number system (RNS) arithmetic with special moduli while the other layers have a software implementation. In addition, a quantization method for the coefficients of a convolutional layer of the CNN, a compression technique for an array of partial products for the hardware implementation in the RNS and a reverse RNS to Binary Number System (BNS) conversion based on the Chinese Remainder Theorem with fractions (CRTf) are proposed. The efficiency of the proposed approach is illustrated by hardware simulation on a Xilinx FPGA.

2. Related work

To solve the problem of consuming large resources, research teams and also the companies supplying such products often use remote computing resources, e.g., large data processing centers [16] and cloud computing [17]. But this solution is not suitable for all practical tasks. For example, system autonomy is needed for unmanned vehicles [18], since the loss of communication with a remote server may cause an accident. Therefore, an autonomous system for intelligent visual information analysis has to be implemented [19]. In autonomous systems, the problem of consuming large computing resources stimulates the development of new approaches to implement the intelligent analysis algorithms of visual information.

The time and hardware cost can be reduced and system autonomy can be guaranteed using the hardware implementation of the neural network components. The hardware implementations of CNNs are complex; as a rule, such networks are trained before hardware design. The hardware design of deep CNNs is accompanied by the problems of high computational complexity. A new approach for reducing the complexity of the convolution operation in CNNs was introduced in [20] and [21]. The hardware implementation of convolution was performed using energy-efficient fast convolution units composed of parallel filters. A CNN accelerator that optimizes equipment use and data throughput using a flexible and efficient architecture for achieving real-time CNN acceleration was presented in [22]. A new and faster FPGA-based CNN architecture that places all layers on a chip and simultaneously acts as a conveyor was proposed in the paper [23]. For facilitating the programming process, a developer's environment with a universal function for creating an accelerator with the proposed optimization methodology was also introduced. The hardware implementation of CNNs on a cluster of several FPGAs was suggested in [24]; the experiment was conducted for the AlexNet CNN and VGG16. With the described solution, the energy consumption was reduced in comparison with the well-known CPU and GPU implementations.

Another approach to improving the performance of CNNs is the choice of alternative non-positional number systems. In the papers [25] and [26], the technical characteristics of a CNN-based device were bettered using RNS calculations. But the proposed approach with calculations in a nested RNS requires too much memory. Also an implementation of a neural network using the RNSnet residue number system was presented in [27]. The effectiveness of the proposed architecture was compared with several well-known neural network implementations. In accordance with experimental evidence, RNSnet consumes 145.5 times less energy and gets accelerated 35.4 times compared to the NVIDIA GTX 1080 graphics processor. In addition, the results showed that RNSnet can reach 8.5 times higher performance than modern neural network accelerators. In [28], a CNN-based super-resolution system was proposed for interpolating images with increasing resolution; to reduce the resources consumed, the authors adopted RNSs. In [29], the moduli of special form $2^{\alpha} \pm 1$ were used and hardware implementation was performed with a 6-bit representation of numbers, which caused a decrease in network accuracy. Also a possible implementation of each network component in the RNS was theoretically, and a potential reduction in energy consumption was shown. A method involving Sobel filters in the convolutional CNN layer and also its hardware FPGA implementation using RNSs were proposed in [30]. As was demonstrated by the authors, the device has an increase in speed and also a reduction in hardware cost in comparison with the BNS implementation. A disadvantage of the method [30] is the fixed values of the convolutional layer coefficients, which significantly decelerates the training process of a CNN.

The well-known approaches considered above have several disadvantages. For example, they do not justify the choice of bit depth for the hardware implementation on FPGA and also require a large amount of memory to store weight coefficients.

Below, we will present an architecture of CNNs that consists of the hardware and software parts. In the hardware part, which implements the convolutional layer of a CNN, an RNS will be used.

3. Convolutional neural networks

A CNN consists of input and output layers and also of several hidden layers. The hidden layers of a CNN consist of convolutional layers, pooling layers and a fully connected classifier – a perceptron that processes the features obtained on the previous layers.

As an example Fig. 1 shows LeNet-5 [12], one of the first CNN architectures intended for recognizing handwritten numbers. It was trained on the MNIST data set [31]. The network includes 7 layers, among which the first five layers consist of alternating convolutional layers (denoted by C_1, C_3 and C_5) and pooling layers (P_2 and P_4) while the last two layers consist of fully connected neuron layers (FC_6 and FC_7). Such a network operates in the following way. At the beginning, it receives an image of size 32×32 at the input; after processing with a set of 6 filter masks of size $5 \times 5 \times 1$, layer C_1 forms an array of 6 feature maps, each of size 28×28. Next, the pooling layer P_1 divides each feature map of the previous layer into neighborhoods of size 2×2 and forms from each neighborhood one element of the feature map to be supplied to the next layer. Thus, layer P_1 reduces the size of the feature maps, and a set of 6 feature maps of size 14×14 is supplied to the input of the next layer. The subsequent layers C_3, P_4 and C_5 have a similar structure. As its output the CNN determines whether an input image belongs to one of 10 classes (denoted by numbers) or not.

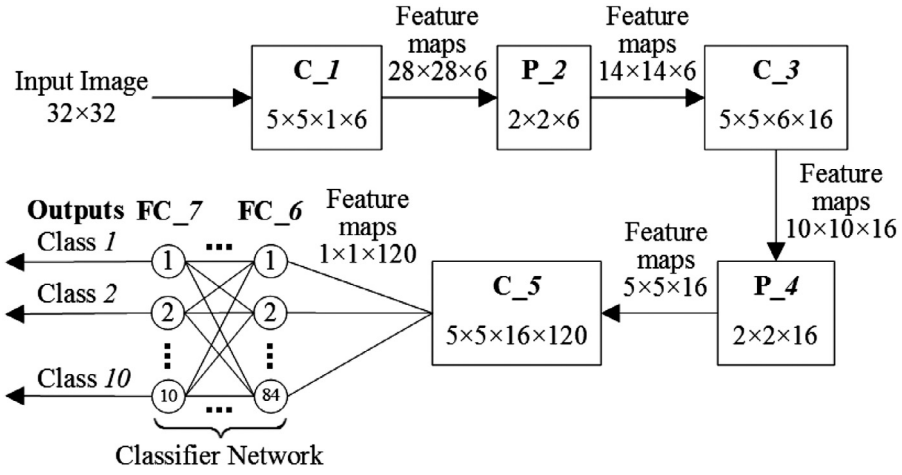

Fig. 1. Convolutional neural network: example of architecture Consider each component of the CNN in detail.

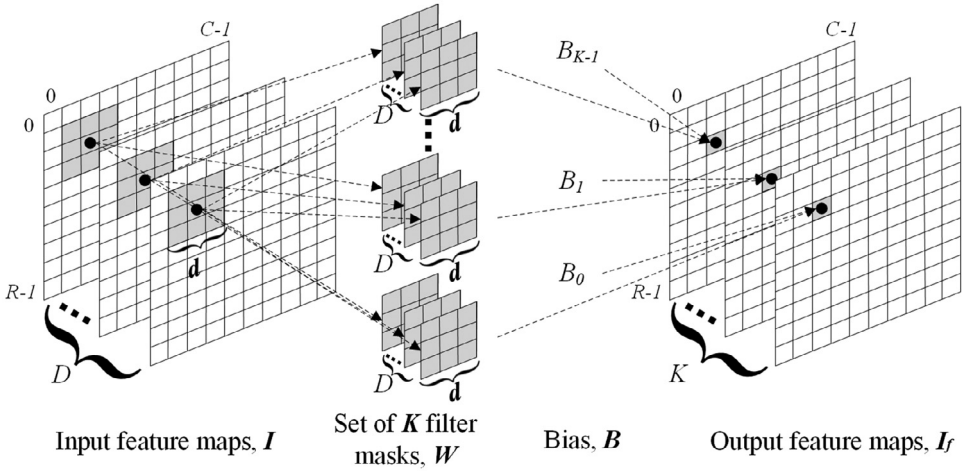

Fig. 2. Procedure for obtaining arrays of feature maps in the convolutional layer of CNN

3.1. Convolution

The purpose of the convolutional layers of the CNN is the formation of feature maps for an image.

Assume a set I of feature maps consisting of D maps with R rows and C columns is supplied to the input of a convolutional layer. The RGB or grayscale image is supplied to the input of the first convolutional layer. Hence, the input of the convolutional layer can be described as a three-dimensional function $I(x, y, z)$, where $0 \le x < R$, $0 \le y < C$ and $0 \le z < D$denote the spatial coordinates while the amplitude I is the intensity of pixels at a given point (x, y, z). The procedure for obtaining the kth feature map, $0 \le k < K$, in the convolutional layer can be represented as

$$I_f(x, y, k) = B_k + \sum_{i=-t}^{t} \sum_{j=-t}^{t} \sum_{z=0}^{D-1} W_{i+t, j+t, z, k} I(x + i, y + j, z), \quad (1)$$

with the following notations: I_f as the set of K output feature maps in the convolutional layer; W as the set of K three-dimensional filter masks of size $d \times d$ that are used for processing of D two-dimensional arrays; $t = \lfloor \frac{d}{2} \rfloor$, where the symbol $\lfloor \bullet \rfloor$ denotes rounding down; B as the bias vector of dimension K. The procedure for obtaining the feature maps is shown in Fig. 2 [30].

Thus, the convolutional layer outputs an array of feature maps that has the same dimension as the number of filter masks.

A linear activation function is used at the output of the convolutional layer. The most common function is the rectified linear unit (ReLU), which forms an array of feature maps $F(x, y, k)$ [32] as follows:

$$F(x, y, k) = \max\left(0, I_f(x, y, k)\right) \quad (2)$$

3.2. Pooling layer

CNNs usually have a large number of filters in the convolutional layer. This leads to a sharp increase in the amount of data processed by the network. To reduce them, a pooling layer is used.

There are different approaches to the implementation of this layer: average largest pooling, max pooling, the calculation of the arithmetic average of all elements. The most commonly used method is max pooling in some neighborhood. An array of feature maps consisting of Dmaps with R rows and C columns comes to the layer's input. Hence, the input of this layer can be described as a three-dimensional function $P_{in}(x_{in}, y_{in}, z)$, where $0 \le x_{in} < R$, $0 \le y_{in} < C$ and $0 \le z < D$ denote the spatial coordinates while the amplitude P_{in} is the intensity of pixels at a given point (x_{in}, y_{in}, z). The max pooling procedure in a neighborhood of size $s \times s$ with a step s for the zth feature map can be described by the formula

$$P_{out}(x_{out}, y_{out}, z) = \max\{P_{in}(x_{in}, y_{in}, z) | s \cdot x_{out} \le x_{in} \le s \cdot (x_{out} + 1), s \cdot y_{out} \le y_{in} \le s \cdot (y_{out} + 1)\}, \quad (3)$$

where $P_{out}(x_{out}, y_{out}, z)$ is the set of D feature maps outputted by the convolutional layer, $0 \le x_{out} < \frac{R}{s} - 1$, $0 \le y_{out} < \frac{C}{s} - 1$.

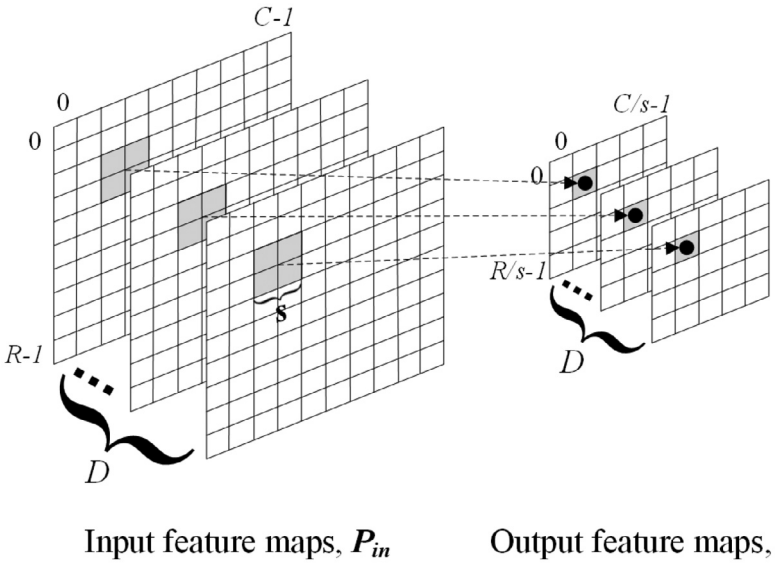

Fig. 3. Output feature maps produced by max pooling layer

The procedure for obtaining the output feature maps in the max pooling layer is demonstrated in Fig. 3 [30].

3.3. Fully connected layers of neurons

The final layers of the network are the fully connected layers of neurons acting as a classifier [33].

Denote by $\{x_i\}, i = \overline{1,m}$, an input vector for the $(p-1)$th layer, where m is the total number of inputs. Each element of the vector is multiplied by a corresponding weight w_{ji}, where $i = \overline{1,m}$, $j = \overline{1,n}$, and n gives the number of all neurons in the $(p-1)$th layer, and the resulting expressions are summed up:

$$y_j = \sum_{i=1}^{m} w_{ji} x_i \tag{4}$$

The activation function for the $(p-1)$th layer has the form

$$\phi(t) = \frac{2}{1+e^{-2t}} - 1 \tag{5}$$

The result of the $(p-1)$th layer is a vector $\{h_j\}$, $j = \overline{1,n}$. Thus, the values of the neurons in the $(p-1)$th layer are calculated by

$$h_j = \frac{2}{1+e^{-2y_j}} - 1 \tag{6}$$

The result calculated by the $(p-1)$th layer is supplied to the input of the pth layer. Multiplication by the corresponding weights and summation yield a vector $\{z_k\}$, where $k = \overline{1,l}$ and l is the number of all neurons in the pth layer. The result of the pth layer is a vector $\{g_k\}$, $k = \overline{1,l}$. The activation function for the pth layer has the form

$$\psi(t) = e^t \tag{7}$$

Thus, the value of the neurons in the last layer is calculated by

$$g_k = \frac{e^{\left[z_k - \max\limits_{k=\overline{1,l}} z_k\right]}}{\sum\limits_{k=1}^{l} e^{\left[z_k - \max\limits_{k=\overline{1,l}} z_k\right]}} \tag{8}$$

A fully connected layer outputs a vector in which the number of elements corresponds to the number of classes and each element reflects the probability that an input image of the network is a member of the associated class.

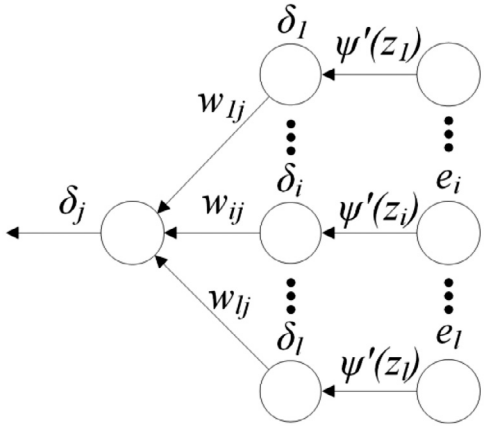

Fig. 4. Neuron training using the backpropagation algorithm.

3.4. Training

The CNN is trained by adjusting the weights. Therefore, the parameters to be trained are contained in the convolutional layers and also in the fully connected layers of neurons.

3.4.1. Training of fully connected layers of classifier

Neural networks are often trained using the backpropagation algorithm [12,33]. Consider this algorithm in detail for the above model of fully connected layers (Fig. 4).

The weights of the last layer are calculated as follows. First, find the error signal (loss function)

$$e_k = d_k - g_k, \tag{9}$$

where $\{d_k\}, k = \overline{1,l}$, is a target vector. Next, find the local gradient δ_k, $k = \overline{1,l}$, by the formula

$$\delta_k = e_k \psi'(z_k) \tag{10}$$

The weights w_{kj} of the output pth layer are adjusted using the corrections

$$\Delta w_{kj} = \eta \delta_k g_k, \tag{11}$$

where a parameter η describes the speed of training.

For the jth neuron of the $(p-1)$th layer, the local gradient is calculated by

$$\delta_j = \phi'(y_j)\sum_{k=1}^{l}\delta_k w_{kj} \tag{12}$$

The weights w_{ji} of the $(p-1)$th layer are adjusted using the corrections

$$\Delta w_{ji} = \eta\delta_j y_j \tag{13}$$

Then the error is transferred to the previous layers of the CNN.

3.4.2. Backpropagation in pooling layer

The max pooling operation is described by formula (3). This layer does not contain any parameters to be trained; hence, the error has to be transferred to the previous layers. For this purpose, calculate the gradient of the max pooling function in a given neighborhood as follows [32]:

$$\delta^{P_{in}}_{x_{in},y_{in},k} = \frac{\partial P_{out}}{\partial P_{in}(x_{in},y_{in},k)} = \begin{cases} 1, \max\{P_{in}(x_{in},y_{in},z) | s\cdot x_{out} \le x_{in} \le s\cdot(x_{out}+1), \\ \qquad\qquad s\cdot y_{out} \le y_{in} \le s\cdot(y_{out}+1)\}, \\ 0, \ otherwise. \end{cases} \tag{14}$$

Next, the error is transferred to the convolutional layer of the CNN.

3.4.3. Training of convolutional layer of CNN

The convolution operation is described by formula (1). The weights of the filter masks are trained by analogy with the training of the fully connected layers of neurons [33]. First, calculate the gradient $\{\delta^{I_f}_{x,y,k}\}$ of the loss function $L(x,y,k)$ for the output of the convolutional layer:

$$\delta^{I_f}_{x,y,k} = \frac{\partial L}{\partial I_f(x,y,k)} \tag{15}$$

The activation function is needed at the output of the convolutional layer. Hence, its local gradient for error propagation has to be calculated. The gradient of the function ReLU has the following form:

$$\delta^{R}_{x,y,k} = \frac{\partial R}{\partial I_f(x,y,k)} = \begin{cases} 0, & I_f(x,y,k) < 0, \\ 1, & I_f(x,y,k) \ge 0. \end{cases} \tag{16}$$

The mask is adjusted using the correction

$$\Delta W_{i,j,z,k} = \frac{\partial L}{\partial W_{i,j,z,k}} = \sum_{x=i}^{R-d+i}\sum_{y=i}^{C-d+j}\delta^{I_f}_{x,y,k} I(x+i,y+j,z) \tag{17}$$

The bias B is adjusted using the correction

$$\Delta B_k = \frac{\partial L}{\partial B_k} = \sum_{x=0}^{R-1}\sum_{y=0}^{C-1}\delta^{I_f}_{x,y,k} \tag{18}$$

For transferring the error to the previous layers, the CNN has to calculate the gradient of the input set I of feature maps by the formula

$$\delta^{I}_{x,y,z} = \frac{\partial L}{\partial I(x,y,z)} = \sum_{i=-t}^{t}\sum_{j=-t}^{t}\sum_{k=0}^{K-1} W_{i+t,j+t,z,k}\delta^{I_f}_{x+i,y+j,k} \tag{19}$$

4. RNS based optimization of calculations in CNN

In comparison with the other CNN layers, the convolution operation needs most of the time resources. For improving the technical characteristics of the CNN, the idea suggested in this paper is to use RNSs.

In an RNS, the numbers are represented in the basis of coprimes (the RNS moduli): $\{m_1, \ldots, m_n\}$, where $GCD(m_i, m_j) = 1$, $i \ne j$, and GCD denotes the common greatest divisor. The product of all RNS moduli is called the dynamic range of the system. Any integer can be uniquely represented in the RNS as a vector $(x_1, x_2, \ldots, x_n)$, where $x_i = |X|_{m_i} = X \bmod m_i$ in accordance with the Chinese Remainder Theorem (CRT) [34].

The dynamic range of an RNS is usually separated into two approximately equal parts, with about half of the range representing positive numbers and the rest of it negative numbers. Consequently, any integer that satisfies one of the relations

$$-\frac{M-1}{2} \le X \le \frac{M-1}{2}, \ for\ odd\ M, \tag{20}$$

$$-\frac{M}{2} \le X \le \frac{M}{2} - 1, \ for\ even\ M, \tag{21}$$

can be written in the RNS representation.

The operations of addition, subtraction and multiplication in the RNS have the form

$$A \pm B = \left(|a_1 \pm b_1|_{m_1}, \ \ldots, \ |a_k \pm b_k|_{m_n}\right), \tag{22}$$

$$A \cdot B = \left(|a_1 \cdot b_1|_{m_1}, \ \ldots, \ |a_k \cdot b_k|_{m_n}\right), \tag{23}$$

Equalities (22) and (23) show the parallel character of RNSs, which are free from local (bitwise) carry. The main advantages of RNS representations are the capabilities of expressing large numbers as the residues of small bit-width and performing addition, subtraction, and multiplication in parallel for each modulus [35]. The key drawbacks of such representations were discussed in [35].

In view of the above advantages, in this paper we will use RNSs for implementing the convolutional layers of the CNN. The proposed architecture of the CNN in which the convolutional layers involve the RNS arithmetic (C_RNS_i, where i denotes the number of CNN layer) is shown in Fig. 5. To implement the calculations in the RNS, it is necessary to add blocks for forward conversion from the BNS to the RNS and reverse conversion from the RNS to the BNS. Thus, the convolutional layer consists of a block for converting numbers from the BNS to RNS, blocks for calculating the convolution for each modulus, and a block for converting numbers from the RNS to BNS. The reverse conversion is based on the Chinese Remainder Theorem with fractions, which dramatically reduces the hardware cost of its implementation.

For implementing RNS calculations, the BNS-to-RNS and RNS-to-BNS converters have to be introduced into the architecture.

Consider RNS calculations in detail.

4.1. Choice of RNS moduli

An important task in the development of an applications-oriented system with RNS calculations is the choice of an appropriate set of moduli $\{m_1, \ldots, m_n\}$. In most modern publications dedicated to RNS applications, a modulus equal to the power of two is used, i.e., 2^α, $\alpha \in N$, where N denotes the set of natural numbers. This can be explained by the fact that mod 2^α calculations have a simplest hardware implementation in the form of conventional arithmetic devices of the binary number system with bit width α. The choice of the moduli 2^α, in combination with the coprimeness requirement for each pair of RNS moduli, dictate that all other moduli of the system must be odd, i.e., have the form $2^\alpha \pm k$, where $\alpha \in N$ and $k = 1, 3, 5, \ldots$. In this paper, we adopt the RNS for implementing the most resource-intensive convolutional layer of the CNN, which performs calculations in accordance with formula (1). Hence, modular adders will have highest hardware cost among all elements of the arithmetic device. Since formula (1) includes the summation of a large number of terms, the

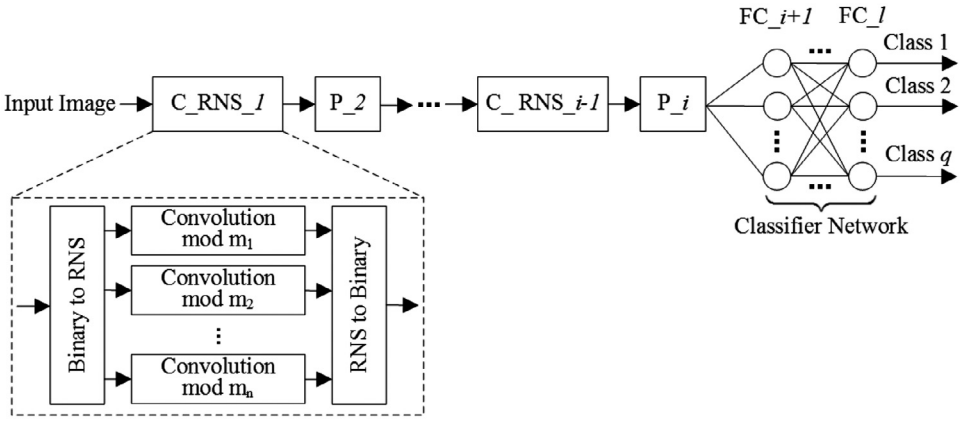

Fig. 5. The proposed architecture of CNN with RNS implementation of convolutional layers in general case.

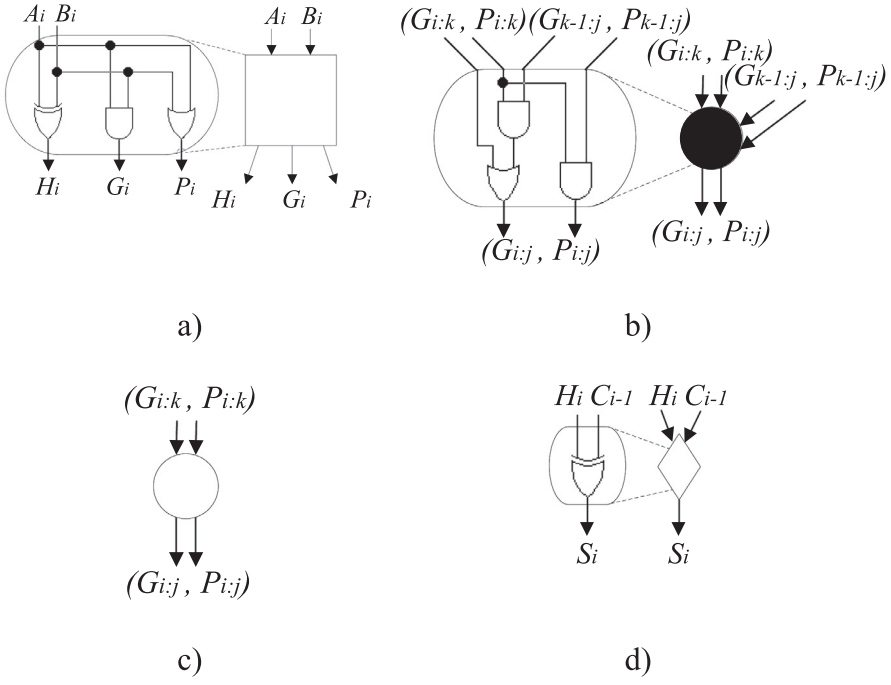

Fig. 6. Summing blocks of parallel prefix adder [36]: a) preliminary calculations; b) and c) carry calculation; d) final sum calculation.

most efficient modular summation techniques should be chosen, which are similar to those used in the traditional binary number system. Such techniques were developed for the moduli 2^{α} and $2^{\alpha} \pm 1$, $\alpha \in N$, only; see [36].

For any moduli of the form 2^{α}, the Carry Save Adders (CSAs) [37] and also the parallel prefix Kogge–Stone Adders (KSAs) [38] are effective approaches to implement summation. The KSA is used to calculate the sum of n-bit numbers $A = \sum_{i=0}^{n-1} 2^i A_i$ and $B = \sum_{i=0}^{n-1} 2^i B_i$. This adder has the following principle of operation. At the preliminary stage, for any i, $0 \le i \le n-1$, the bits G_i generating the carry, the bits P_i passing the carry and also the half-sums H_i are calculated by the formulas

$$G_i = A_i \wedge B_i,\ P_i = A_i \vee B_i,\ \ H_i = A_i \oplus B_i, \tag{24}$$

where $\wedge$, $\vee$ and $\oplus$ denote the bitwise AND, the bitwise OR and the bitwise XOR, respectively. The structural diagram of this block is shown in Fig. 6a.

Then, using G_i and P_i the carry signals C_i, $0 \le i \le n-1$, are calculated. The carry is written in the parallel prefix form using the operator $\circ$, which connects the pairs of generating and passing bits as follows:

$$(G, P) \circ \left(G', P'\right) = \left(G \vee (P \wedge G'), P \wedge P'\right) \tag{25}$$

The successive calculation of the pairs of generating and passing bits (G, P) is denoted by $(G_{k:j}, P_{k:j})$, $k > j$, where a corresponding pair is obtained using the bits $k, k-1, \ldots, j$ in by the formula

$$\left(G_{k:j}, P_{k:j}\right) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \ldots \circ \left(G_j, P_j\right) \tag{26}$$

Since for any $i > 0$ the carry is $C_i = G_{i:0}$, all carries can be calculated using the operator $\circ$ only [36]. The structural diagram of carry calculation blocks is shown in Figs. 6b and 6c.

The final stage (Fig. 6d) is intended to calculate the sum

$$S_i = H_i \oplus C_{i-1} \tag{27}$$

The structures of the 8-bit Kogge–Stone Adder (KSA) and the Kogge–Stone Adder with the End Around Carry (EAC-KSA) are demonstrated in Figs. 7a and Fig. 7b, respectively.

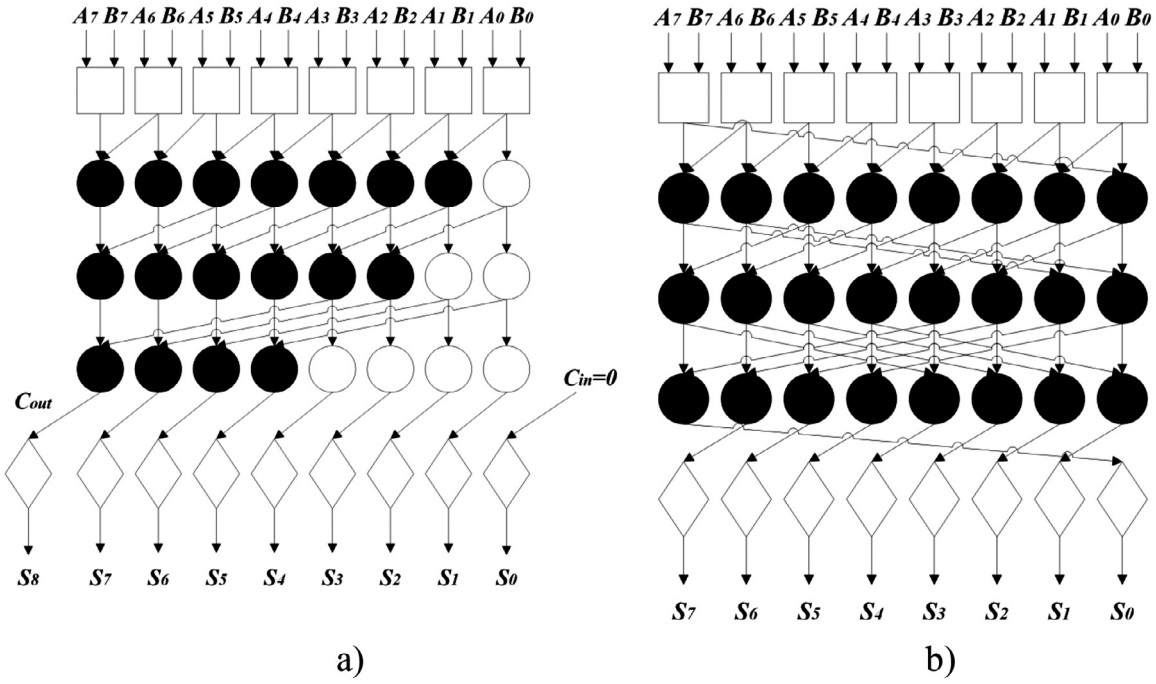

Fig. 7. Structure of 8-bit parallel prefix Kogge–Stone Adder: a) for BNS [38] and b) with End Around Carry and modulo (2^8-1) calculations [36].

Like for the moduli 2^α, for the moduli $2^\alpha-1$ the effective approaches to summation are CSA and KSA; their only difference consists in the End Around Carry (EAC) (see Fig. 7b).

For implementing a mod $(2^\alpha+1)$ computing channel, where $\alpha \in N$, an additional logic (like diminished-1) has to be introduced for tracking the zero code combination, which is undesirable in the development of a system with minimum hardware and time cost [39,40]. Thus, for designing the CNN with RNS calculations, a reasonable solution is using only the moduli $2^\alpha-1$, where $\alpha \in N$, as the odd system moduli.

4.2. BNS-to-RNS conversion

In the case of RNS calculations, the first unit is to represent numbers in the RNS. For performing the BNS-to-RNS conversion, the remainders of division by each modulus have to be calculaed. This operation is resource-intensive. It can be eliminated by choosing the moduli $\{2^{\alpha_1}, 2^{\alpha_2}-1, \ldots, 2^{\alpha_n}-1\}$. The remainder of mod 2^{α_1} division is obtained by simply keeping the α_1 least significant bits of an initial number and discarding the other (most significant) bits. For the moduli $2^\alpha-1$, the remainder of the division is calculated as follows. Let $X=\sum_{i=0}^{g-1} X_i 2^i$ be an initial g-bit number represented in the BNS by the bits X_i, $0 \le i < g$. Separate it into $s=\lceil \frac{g}{\alpha} \rceil$ parts, each containing α bits (the symbol $\lceil \bullet \rceil$ denotes rounding up). For this purpose (if necessary), complement X with most significant bits equal to 0 for reaching the size of $g'=s\alpha$ bits. Thus, the number X' in the BNS is written as $X'=\sum_{i=0}^{g'-1} X_i 2^i$ while its α-bit parts have the form $Y_0=\sum_{i=0}^{\alpha-1} X_i 2^i$, $Y_1=\sum_{i=\alpha}^{2\alpha-1} X_i 2^i$, …, $Y_s=\sum_{i=(s-1)\alpha}^{g'-1} X_i 2^i$. With the notations $Y_0, Y_1, \ldots, Y_s$, the number X' can be represented by $X'=Y_0+Y_1\cdot 2^\alpha+Y_2\cdot 2^{2\alpha}+\ldots+Y_s\cdot 2^{s\alpha}$. Since $|Y_i\cdot 2^{i\alpha}|_{2^\alpha-1}=|Y_i|_{2^\alpha-1}$, $0 \le i < s$, it follows that

$$\left|X'\right|_{2^\alpha-1}=|Y_0+Y_1+Y_2+\ldots+Y_s|_{2^\alpha-1} \tag{28}$$

In accordance with formula (28), the remainder of mod $(2^\alpha-1)$ division can be calculated by performing the mod $(2^\alpha-1)$ summation of the α-bit numbers, which is a particular case of distributed arithmetic. For the mod $(2^\alpha-1)$ summation of more than two terms, the CSA tree and the EAC are often used. The result is transferred to the EAC-KSA.

4.3. Convolution in RNS

RNSs are most effective in cases where calculations mainly require the operations of summation and multiplication. In accordance with formula (1), the convolution operation in the RNS involves only these operations. This means that an RNS can be effectively used for the hardware implementation of the convolutional layer of the CNN. The RNS implementation of the comparison operation has high complexity and hence should not be chosen for implementing the max pooling layer and also the fully connected layer of neurons. This fact suggests an approach consisting in the separation of the CNN into the hardware and software parts. The idea proposed in this paper is to use a hardware solution for implementing the convolutional part of the CNN in the RNS and a software solution for implementing the other layers of the CNN.

4.3.1. Analysis of quantization noise generated by hardware implementation of convolutional layers of CNN

The coefficients of the filter mask W and bias B from formula (1) are constants in the trained CNN. This means that the convolution device must implement multiplication by constants with further summation of the result. The hardware implementation of calculations in the convolutional layer of the CNN by formula (1) requires the quantization of the filter coefficients W, which generates quantization noise. The bias B from formula (1) does not affect the rounding of the filter coefficients and hence is not related to the convolution error. Multiply the coefficients by 2^T, where T is the scaling factor, and then round up. Then divide the RNS result of the convolution by 2^T and round down. This approach has several advantages as follows.

1. The rounding errors will have different signs and partially compensate each other. Thus, the influence of quantization noise on the convolution result will be reduced.
2. The rounding operations performed in this order require less hardware resources than the operation of rounding to the

nearest integer. This fact has an obvious explanation: the filter coefficients are a priori known and their quantization with rounding up can be performed in advance. Convolution is performed using arithmetic-logic devices and the result is rounded down by simply dropping the fractional part, which does not incur additional hardware and time cost.

3. In the binary representation, the operations of multiplication and division by 2^T correspond to the right and left bit shifts, respectively, which considerably simplifies and accelerates their execution.

If the scaling factor T increases, the influence of quantization noise on the convolution result will decrease, but the hardware and time cost for its execution will grow. This leads to a natural question as follows. Which value of the parameter T is effective in terms of the hardware implementation on modern devices and, at the same time, necessary for obtaining the convolution result of acceptable quality? To answer this question, we will estimate the effect of quantization noise on the convolution result. Determine the absolute errors (AEs) E_+ and E_- for rounding the positive and negative coefficients W from formula (1) by

$$\begin{aligned} E_+ &= \sum_{W_{i,j,k}>0} \left(\left\lceil 2^T W_{i,j,k}\right\rceil - 2^T W_{i,j,k}\right), \\ E_- &= \sum_{W_{i,j,k}<0} \left(\left\lceil 2^T W_{i,j,k}\right\rceil - 2^T W_{i,j,k}\right) \end{aligned} \tag{29}$$

Hence, the limited absolute error (LAE) E_1 of rounding the filter coefficients W can be calculated as

$$E_1 = \max\{E_+, E_-\} \tag{30}$$

Denote by β the maximum value of the signal (here, the image intensity). Then the LAE E_2 of the convolution results is

$$E_2 = E_1 \cdot \beta \tag{31}$$

The LAE E_3 of the normalized convolution results can be obtained from the LAE E_2 of the convolution results by taking the parameter T into account as follows:

$$E_3 = \frac{E_2}{2^T} = 2^{-T}\beta \cdot E_1 \tag{32}$$

Denote by $\lambda \in [0, 1)$ the fractional part of the exact convolution result. Then the LAE E_4 of rounding the normalized convolution results can be found using the formula

$$E_4 = E_3 + \lambda - \lfloor E_3 + \lambda \rfloor \tag{33}$$

After the convolution operation, the exact value of the result is generally not an integer. Thus, the value E_4 depends not only on E_3 but also on λ. If $E_3 = 0$, then E_4 in formula (33) is an irreducible error. It correlates with E_3, always affecting the accuracy of calculations by the method under study. With an increase in the value of the parameter T, the influence of this error on the result of calculations will decrease. In practice, the influence of this error becomes negligible upon reaching a certain value T.

The total error of calculating the convolution calculation is the LAE E_5 of rounding down the normalized convolution results:

$$E_5 = |E_3 - E_4| \tag{34}$$

Formula (34) indicates that the error E_4 of rounding down the convolution results is partially compensating the error E_3 of the convolution results based on rounding up the filter coefficients W (the error E_1). At a certain stage of increasing the value T, these two errors will become commensurate by absolute value. In practice, at this stage the resulting error E_5 of the method will have the smallest value. Further increasing the value T will reduce the error E_3, in contrast to the error E_4, which will cause higher values of the resulting error E_5.

One of the parameters affecting the value E_5 is λ. Note that its specific value cannot be determined during theoretical calculations. In view of its effect, this parameter has to be eliminated. For this purpose, in formula (34) we will express E_4 through E_3 and λ in accordance with formula (33):

$$E_5 = |E_3 - (E_3 + \lambda - \lfloor E_3 + \lambda \rfloor)| = |\lfloor E_3 + \lambda \rfloor - \lambda| \tag{35}$$

Consider two cases as follows.

1. $\lfloor E_3 + \lambda \rfloor - \lambda > 0 \Rightarrow \lfloor E_3 + \lambda \rfloor \geq 1$. In this case, higher values of $\lfloor E_3 + \lambda \rfloor$ lead to higher values of E_5. Thus, $\lfloor E_3 + \lambda \rfloor = \lfloor E_3 \rfloor + 1$ and λ is the one's complement for the fractional part of the number E_3, i.e., $\lambda = \lfloor E_3 \rfloor + 1 - E_3$. Substituting this expression into formula (35) yields

$$E_5 = \lfloor E_3 + \lfloor E_3 \rfloor + 1 - E_3 \rfloor - (\lfloor E_3 \rfloor + 1 - E_3) = E_3 . \frac{1}{2} \tag{36}$$

2. $\lfloor E_3 + \lambda \rfloor - \lambda \leq 0 \Rightarrow \lfloor E_3 + \lambda \rfloor \leq \lambda \Rightarrow \lfloor E_3 + \lambda \rfloor = 0 \Rightarrow E_5 = |0 - \lambda| = \lambda$. In this case, higher values of λ lead to higher values of E_5. Moreover, $\lfloor E_3 + \lambda \rfloor = 0 \Rightarrow E_3 + \lambda = 1 - \varepsilon$, which gives $\lambda = 1 - \varepsilon - E_3$. Under these conditions, use the AE of the normalized convolution results instead of E_3, letting it equal to 0:

$$E_5 = |\lfloor 0 + 1 - \varepsilon \rfloor - (1 - \varepsilon)| = 1 - \varepsilon, \tag{37}$$

where ε denotes the machine zero.

Because the value E_5 describes the maximum possible error, formula (37) should be used for $E_3 \geq 1$. Therefore, from formulas (35) and (36) we may uniquely determine E_5 depending on E_3 regardless of the specific values of λ:

$$E_5 = \begin{cases} E_3, & E_3 \geq 1, \\ 1 - \varepsilon, & E_3 < 1. \end{cases} \tag{38}$$

Using expression (32) rewrite formula (38) as follows:

$$E_5 = \begin{cases} 2^{-T}\beta \cdot E_1, & 2^{-T}\beta \cdot E_1 \geq 1, \\ 1, & 2^{-T}\beta \cdot E_1 < 1. \end{cases} \tag{39}$$

After the above-mentioned actions, the peak signal-to-noise ratio (*PSNR*) can be calculated as

$$PSNR = 10 \lg \frac{\beta^2}{{E_5}^2} = 20 \lg \frac{\beta}{E_5} \tag{40}$$

Using expression (39) rewrite formula (39) as follows:

$$PSNR = \begin{cases} 20 \lg \frac{2^T}{E_1}, & 2^T \leq \beta \cdot E_1, \\ 20 \lg \frac{\beta}{1-\varepsilon}, & 2^T > \beta \cdot E_1. \end{cases} \tag{41}$$

Formula (41) describes how the peak signal-to-noise ratio of the convolution results calculated by formula (1) depends on the scale factor T, the size of the filter of the convolutional layer and the maximum value β of the image intensity. In practice, formula (41) can be applied as follows. One should choose the smallest value of the parameter T that guarantees an acceptable quality of the processed image. This will minimize hardware cost for implementing the convolution. The bit-width r of the filter coefficients after quantization is determined by the formula

$$r = \max\{r_{i,j,z}\}, \tag{42}$$

where $r_{i,j,z}$ specifies the number of digits required for storing the quantized coefficient $\lceil 2^T W_{i,j,z} \rceil$. This number is given by

$$r_{i,j,z} = 1 + \left\lceil \log_2 \left(\left| \left\lceil 2^T W_{i,j,z} \right\rceil \right| + 1 \right) \right\rceil \tag{43}$$

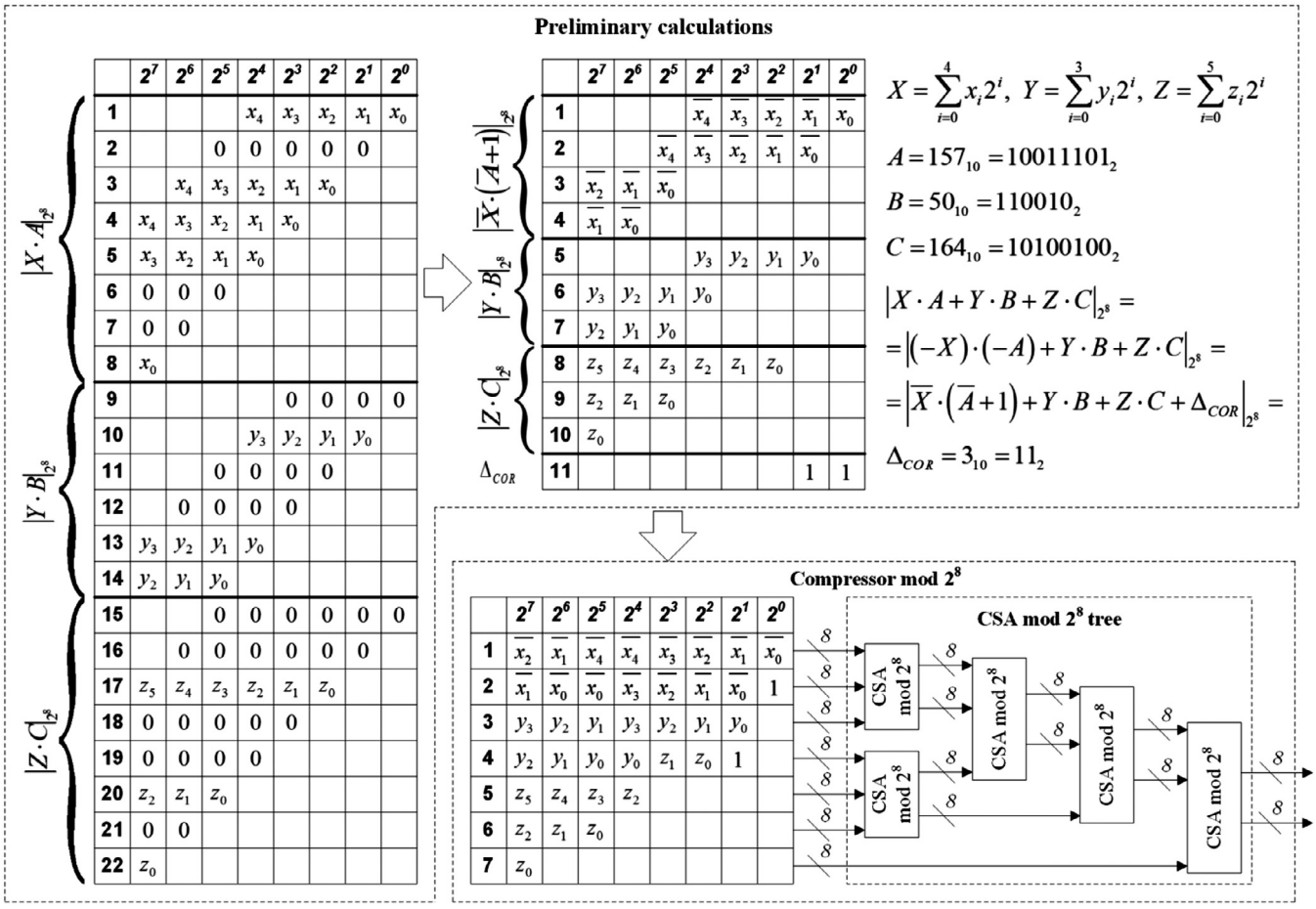

Fig. 8. Compressor for expression $|X\cdot A+Y\cdot B+Z\cdot C|_{256}$.

4.3.2. Method for implementing calculations with multiplication by constant

For multiplying by some constant, the partial products can be compressed in accordance with its value [41]. Compression techniques allow decreasing the number of additions.

Consider the use of compression techniques for the mod 2^α and mod $2^{\alpha-1}$ multiplication by a constant. Multiplication by a constant is reduced to the operations of summation and bit shift [37]. The partial products equal to 0 can be excluded from further calculations. Therefore, if a constant contains more bits equal to 1 than equal to 0, the inverse of the constant has to be used. This leads to the inversion of the second factor and the addition of one term, called the correction factor Δ_{COR}; see Fig. 8.

Consider the multiplication of a number $X=\sum_{i=0}^{g-1}x_i2^i$, $g<\alpha$, by a constant $C=\sum_{i=0}^{f-1}c_i2^i$, $f<\alpha$. For mod 2^α calculations, the negative numbers are reduced to the two's complement representation, i.e.,

$$|(-X)\cdot(-C)|_{2^\alpha}=\left|\overline{X}\cdot\left(\overline{C}+1-2^f\right)+\Delta_{COR}\right|_{2^\alpha}, \tag{44}$$

where the correction coefficient has the form

$$\Delta_{COR}=\left|(1-2^g)\cdot\left(\overline{C}+1-2^f\right)\right|_{2^\alpha} \tag{45}$$

For mod $(2^\alpha-1)$ calculations, the negative numbers are reduced to the one's complement representation, i.e.,

$$|(-X)\cdot(-C)|_{2^\alpha-1}=\left|\overline{X}\cdot\left(\overline{C}+1-2^f\right)+\Delta_{COR}\right|_{2^\alpha-1}, \tag{46}$$

where the correction coefficient has the form

$$\Delta_{COR}=\left|(1-2^g)\cdot\left(\overline{C}+1-2^f\right)\right|_{2^\alpha-1} \tag{47}$$

The resulting partial products are then compressed vertically, as is shown in Fig. 8. In this way, the terms are generated, which are then added using the CSA tree. In the case of mod 2^α calculations, the adders cut the most significant carry bit; in the case of mod $(2^\alpha-1)$ calculations, they perform EAC.

Example 1. . Consider the above compression technique for the expression $|X\cdot A+Y\cdot B+Z\cdot C|_{256}$, where $X=\sum_{i=0}^{4}x_i2^i$, $Y=\sum_{i=0}^{3}y_i2^i$, $Z=\sum_{i=0}^{5}z_i2^i$, $A=157_{10}=10011101_2$, $B=50_{10}=110010_2$ and $C=164_{10}=10100100_2$. Note that the constant A contains more 1s than 0s. Hence, multiplication by inversion has to be used, and the correction coefficient is $\Delta_{COR}=11_2$; see formula (45). Next, the partial products are compressed vertically and summed up using the tree of mod 256 CSAs. Thus, instead of 22 terms just 7 terms are summed up.

The coefficients W and the bias B from formula (1) are constants in the trained CNN. This means that the convolution device must implement multiplication by constants with the subsequent summation of the result. Since the idea is to choose the RNS moduli $\{2^{\alpha_1},2^{\alpha_2}-1,\ldots,2^{\alpha_n}-1\}$, multiplication by a constant can be implemented very efficiently using the described compression technique. The diagram in Fig. 9 illustrates the hardware implementation of the mod 2^α convolution operation with the proposed compressor. This approach is used for implementing the convolutional layers of the CNN.

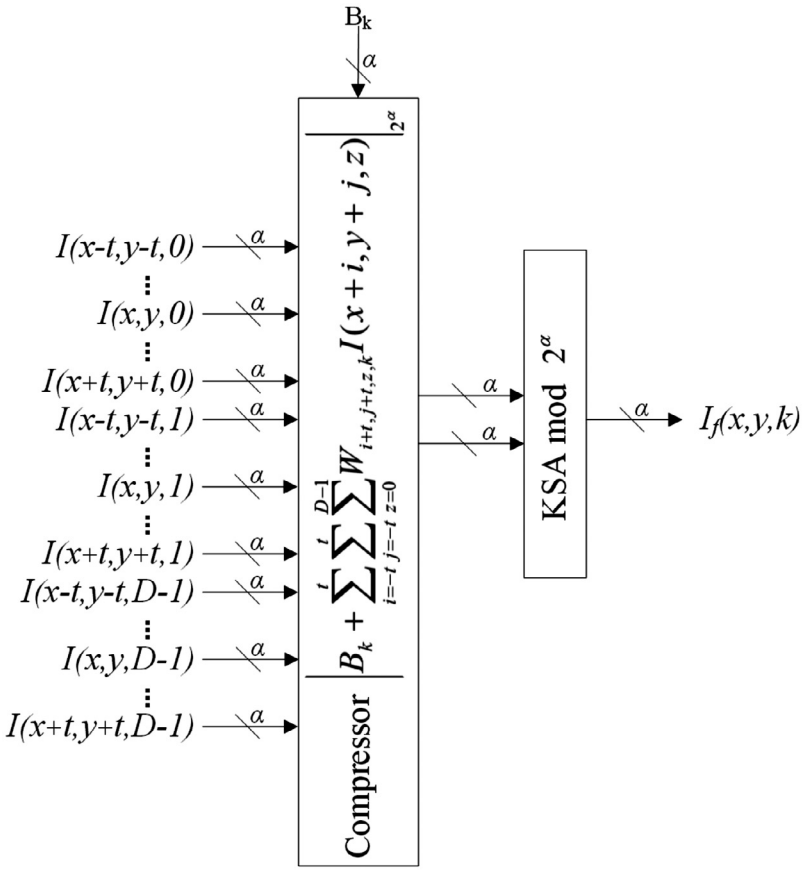

Fig. 9. Hardware implementation of mod 2^{α} convolution operation.

4.4. RNS-to-BNS conversion

As a matter of fact, the RNS-to-BNS conversion is one of the most complex operations in terms of hardware implementation. This conversion is performed using the corollaries of the CRT [34]. The positional representation of a number X in the BNS is calculated from its RNS representation (i.e., the remainders (x_1, x_2, ..., x_n) of this number on the moduli $\{m_1, ..., m_n\}$) in accordance with the formula

$$X = \left| \sum_{i=1}^{n} \left| M_i^{-1} \right|_{m_i} M_i x_i \right|_M , \tag{48}$$

where $M_i = \frac{M}{m_i}$ and $|M_i^{-1}|_{m_i}$ denotes the mod m_i multiplicative inverse element of M_i, $i = 1, 2, \ldots, n$. A direct application of formula (48) requires the calculation of the remainder of the division by the number M with the bit-width equal to the complete dynamic range DR of the system. The hardware implementation of this operation is computationally intensive, which is an undesirable effect for the implementation of the CNN with RNS calculations.

The main goal of most studies of the reverse transform is the maximum simplification of this operation, the reduction of mod M calculations of size DR to less intensive operations. One of the most effective approaches is the modification of the CRT [42], which consists in approximating the relative position of numbers on the number line. This method is called the CRT with fractions (CRTf). The core of the method is to modify formula (48) so that it can be used for estimating the position of a number on the number line without complex arithmetic operations.

Divide both sides of (48) by M:

$$\frac{X}{M} = \left| \sum_{i=1}^{n} x_i \frac{\left| M_i^{-1} \right|_{m_i} M_i}{M} \right|, \tag{49}$$

where $|*|_1$ denotes the fractional part of a number. With the notation $k_i = \frac{|M_i^{-1}|_{m_i} M_i}{M}$, expression (49) can be written in the compact form

$$\frac{X}{M} = \left| \sum_{i=1}^{n} x_i k_i \right|_1 \tag{50}$$

Obviously, the values $\frac{X}{M}$, k_1, k_2, ..., k_n are rational numbers within the interval [0, 1). In addition, the values k_i depend on the RNS moduli only, thereby being the RNS constants.

Formula (50) contains calculations with fractional numbers of arbitrary length and hence cannot be used in hardware implementation. For solving this problem, we may pass to the integer arithmetic with relative values. More specifically, multiply each constant from formula (50) by 2^N and round the result up:

$$\bar{k}_i = \left\lceil 2^N \frac{\left| M_i^{-1} \right|_{m_i} M_i}{M} \right\rceil \tag{51}$$

The sum of the mod 2^N products of the remainders by such constants will exceed the relative value $2^N \frac{X}{M}$. Then expression (50) will be written as

$$\bar{X} = \left\lfloor 2^N \frac{X}{M} \right\rfloor = \left| \sum_{i=1}^{n} x_i \bar{k}_i \right|_{2^N} \tag{52}$$

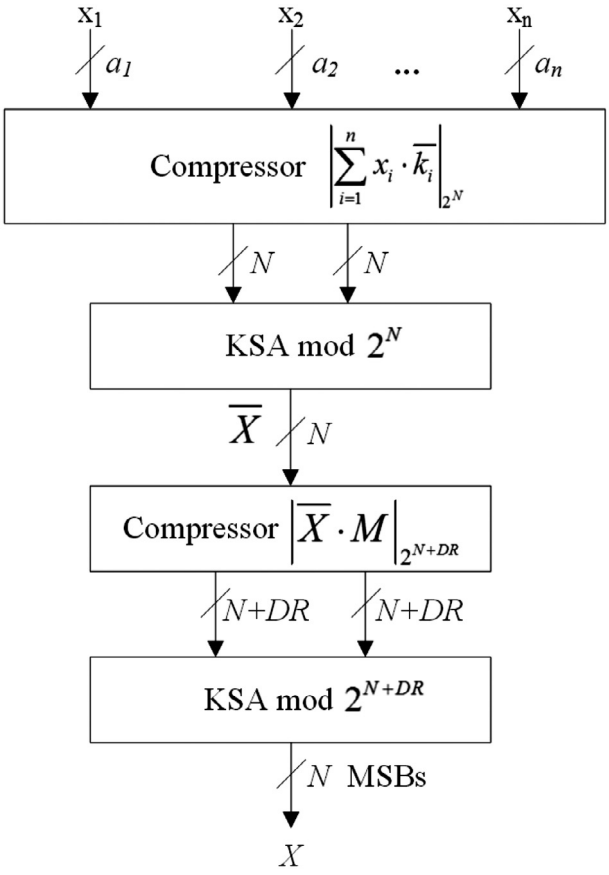

Fig. 10. Architecture of RNS-to-BNS conversion algorithm based on CRTf.

As was established in [42], the exact RNS-to-BNS conversion can be guaranteed by choosing

$$N = \lceil \log_2(M\mu) \rceil - 1, \tag{53}$$

where $\mu = -n + \sum_{i=1}^{n} m_i$.

The last step to calculate the value X using the value $\lfloor 2^N \frac{X}{M} \rfloor$ is multiplication by the modulus M. The algorithm yields the most significant bits starting from the $(N+1)$th one. Thus,

$$X = \frac{\bar{X}M}{2^N} \tag{54}$$

During the hardware implementation, the division operation in (54) is ignored, since the most significant bits starting from the $(N+1)$th one are actually outputted. In the software implementation, this operation is equivalent to the right N-bit shift.

Thus, the RNS-to-BNS conversion algorithm based on formulas (52) and (54) allows eliminating the calculation of the mod M remainder of the size DR by replacing it with multiplication, which is a simpler operation in terms of hardware implementation.

Fig. 10 shows the architecture of the hardware implementation of the RNS-to-BNS conversion algorithm. For the effective hardware implementation of multiplication by a constant, this approach involves a compressor that has been described in subsection 4.3.2 and also the mod 2^N KSA.

Example 2. . Let an RNS is defined by the set of moduli $m_1 = 2$, $m_2 = 3$, $m_3 = 5$ and $m_4 = 7$. Then $M = 2 \cdot 3 \cdot 5 \cdot 7 = 210$ and $\mu = -4 + 2 + 3 + 5 + 7 = 13$. In accordance with formula (53), calculate $N = \lceil \log_2(210 \cdot 13) - 1 \rceil = 11$. The constants k_i in formula (50) are $k_1 = \frac{1}{2}$, $k_2 = \frac{1}{3}$, $k_3 = \frac{3}{5}$ and $k_4 = \frac{4}{7}$. For integer calculations using (52), transform the constants into $\bar{k}_1 = 1024$, $\bar{k}_2 = 683$, $\bar{k}_3 = 1229$ and $\bar{k}_4 = 1171$.

Let $X = (1, 1, 2, 5)$ be some number represented in the RNS with the moduli $\{2, 3, 5, 7\}$. Restore the positional representation of this number by the described method. In accordance with formula (52),

$$\bar{X} = |1024 \cdot 1 + 683 \cdot 1 + 1229 \cdot 2 + 1171 \cdot 5|_{2^{11}} = 1828.$$

The positional characteristic $\bar{X}$ can be used for restoring the desired number with formula (54):

$$X = \left\lfloor \frac{1828 \cdot 210}{2^{11}} \right\rfloor = 187,$$

which is the correct value of the number X.

5. Experimental simulation of CNN with RNS calculations

As an experimental base, the CNN for recognizing 8 classes of images from the Illinois University image database [43]. was developed. The classes of images from this database are shown in Fig. 11. All images of the database were converted to the resolution of 256×192 pixels using the standard bicubic interpolation algorithm of Adobe Photoshop CS6. The CNN was trained on 161 images from the database.

The main purpose of the simulation was to minimize the hardware and time cost for implementing the CNN. The convolution operation consumes most of the CNN working time. To increase the performance, we divided the CNN architecture into the hardware and software parts. The convolutional layer was implemented in the hardware part using the FPGA with RNS computations. The pooling layer and also the fully connected network were implemented in the software part.

The proposed architecture of CNN is demonstrated in Fig. 12. An RGB image of size 256×192 is supplied to the CNN input; the first two layers are responsible for the selection of image features. The first layer (C_RNS_1) performs the convolution operation using 8 filters of size $3 \times 3 \times 3$, with the step equal to 3. The convolution is calculated using the RNS; see Fig. 5. The result calculated by the first layer is a set of 8 feature maps of size 85×64. The second layer (P_2) performs the pooling operation using a mask of size 2×2, with the step equal to 2. The output of the second layer is an array of 8 feature maps of size 42×32, which are connected to the inputs of the last two layers responsible for image classification. The third layer (FC_3) consists of 10 neurons, and the fourth layer (FC_4) consists of 8 neurons, each of which corresponds to a particular class.

The CNN was trained in MATLAB R2017b. The calculations were performed on a PC with an Intel (R) Core (TM) i7-4790K CPU@4.00GHz processor, 16GB RAM and 64-bit Windows 10. 161 images from 8 classes were used for training [43].. Fig. 13 shows the graph of the training process plotted in MATLAB. The neural network was trained in 35 iterations, and the recognition accuracy reached 97.56%.

In Table 1, the filter mask of the convolutional layer of the trained CNN for $k = 0$ is presented, before and after quantization. This mask corresponds to the 3D filter from the set W for $k = 0$ and is used to calculate the feature map by formula (1). For the hardware implementation, the values of the filter coefficients were reduced to the 8-bit representation. The bit-width $r = 8$ was determined by formula (43) with $T = 11$. This is the smallest value of the parameter T under which the result of image processing by the filter of size $3 \times 3 \times 3$ reaches a guaranteed quality of 40 dB in accordance with formula (41). This value describes the difference between the two images that is invisible to the human eye [39].

In what follows, we will present the hardware simulation with the quantized filter mask (see Table 1).

The hardware simulation was performed on a Kintex7 xc7k70tfbg484-2 FPGA using Xilinx Vivado 16.3. For the simulation, the parameter “High Performance Optimized” was selected.

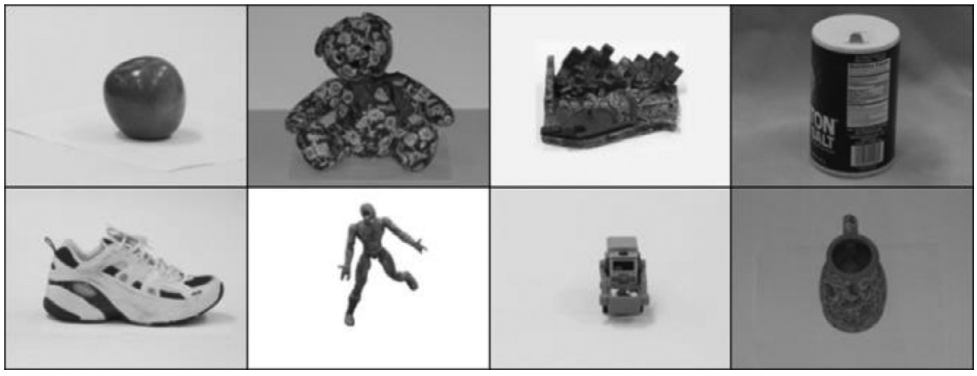

Fig. 11. Classes of images from database [43].

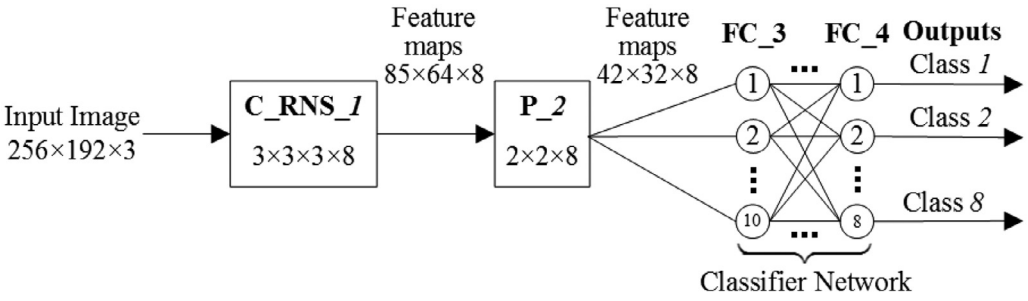

Fig. 12. The proposed architecture of CNN with RNS implementation of convolutional layer, for image database [43].

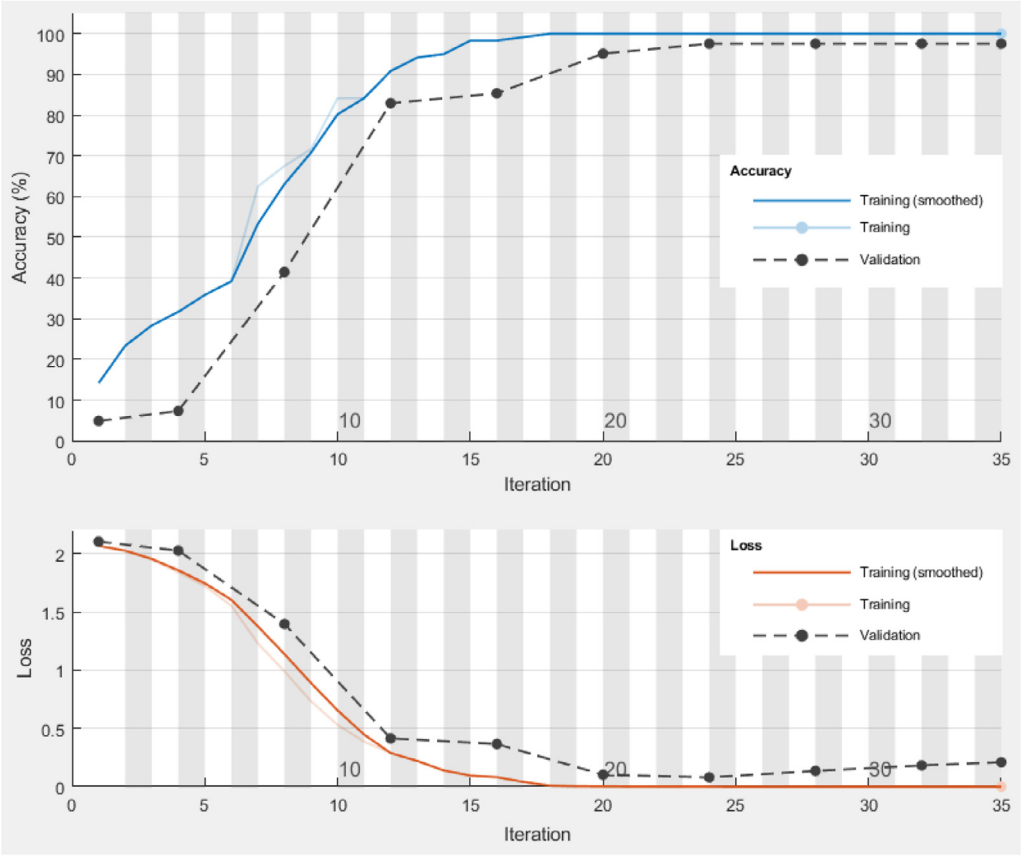

Fig. 13. Graph of CNN training in MATLAB.

Table 1
3D mask of the filter of convolutional layer in trained CNN, $k = 0$.

Components of filter mask	Filter mask before quantization	Filter mask after quantization
$W_{i.j.0.0}$	−0, 0394456 − 0, 0257306 − 0, 0239846 (−0, 0312350 − 0, 0403409 − 0, 0520126) −0, 0434387 − 0, 0364724 − 0, 0520036	−80 − 52 − 49 (−63 − 82 − 106) −88 − 74 − 106
$W_{i.j.1.0}$	−0, 0280364 − 0, 0397294 − 0, 0528939 (−0, 0423464 − 0, 0429127 − 0, 0613428) −0, 0128505 − 0, 0548165 − 0, 0599076	−57 − 81 − 108 (−86 − 87 − 125) −26 − 112 − 122
$W_{i.j.2.0}$	−0, 0304336 − 0, 0289807 − 0, 0600239 (−0, 0578725 − 0, 0404073 − 0, 0482339) −0, 0492809 − 0, 0410201 − 0, 0474128	−62 − 59 − 122 (−118 − 82 − 98) −100 − 84 − 97
B_0	-0,0003320	0

Table 2
Simulation of convolution operation of CNN for different moduli.

Modulus	Delay, ns	LUTs
$2^2 - 1$	8,805	68
$2^3 - 1$	10,474	113
$2^4 - 1$	12,794	293
$2^5 - 1$	13,669	455
$2^6 - 1$	15,187	672
$2^7 - 1$	17,110	1030
$2^8 - 1$	15,988	1329
$2^9 - 1$	15,085	1591
$2^{10} - 1$	17,274	1947
2^7	13,187	342
2^8	15,756	459
2^9	13,755	619
2^{10}	15,998	855

Table 3
Hardware simulation results for convolutional layer of CNN.

Number system	Delay		Occupied LUTs	
	Value, ns	Ratio to BNS, %	Quantity	Ratio to BNS, %
BNS	18.769	100.0	4392	100.0
RNS $\{2^3 - 1, 2^4 - 1, 2^5 - 1, 2^9\}$	24.717	131.7	**2961**	**67.4**
RNS $\{2^4 - 1, 2^9 - 1, 2^9\}$	26.245	139.8	**3587**	**81.7**
RNS $\{2^5 - 1, 2^7, 2^9 - 1\}$	24.167	128.8	**3284**	**74.8**

The purpose of the hardware simulation was to compare the use of the BNS and RNS.

The simulation of the convolution operation was performed for different moduli of the form 2^α and $2^\alpha - 1$. The results are illustrated in Table 2; clearly, the delay varied from 8.805 ns to 17.274 ns and the number of lookup tables (LUTs) varied from 68 to 1947.

Recall that negative numbers in the RNS have to be represented by formulas (20) and (21). Taking this aspect and also the values of the filter coefficients into account, we establish that the dynamic range of the RNS must satisfy the condition $M \geq 2 \cdot 255 \cdot \max\{0, 2326\} = 1186260$ [44], where 255 is the maximum intensity of the image; 0 is the sum of all positive filter coefficients; 2326 is the absolute value of the sum of its negative coefficients. This condition and the data from Table 2 suggest that the three sets of RNS moduli, $\{2^3 - 1, 2^4 - 1, 2^5 - 1, 2^9\}$, $\{2^4 - 1, 2^9 - 1, 2^9\}$ and $\{2^5 - 1, 2^7, 2^9 - 1\}$, can be chosen for the CNN containing the BNS-to-RNS conversion, the convolution in the RNS and the reverse RNS-to-BNS conversion. All these sets of RNS moduli have minimum delay and also completely cover the range M.

The simulation results using the RNS and BNS are combined in Table 3. In accordance with the experimental evidence, the RNS with the moduli $\{2^3 - 1, 2^4 - 1, 2^5 - 1, 2^9\}$ allows reducing the hardware cost by 32.6%; the RNS with the moduli $\{2^4 - 1, 2^9 - 1, 2^9\}$, by 18.3%; the RNS with the moduli $\{2^5 - 1, 2^7, 2^9 - 1\}$, by 25.2% (all figures compared with the BNS). Hence, we may conclude that the RNS-based hardware implementation of the convolutional layer of the CNN is more efficient in terms of the area compared to the BNN-based implementation, which is clear from Table 3 (see the cells in bold face). A possible generalization of the obtained results to the case of larger filter masks needs further research in practice.

Table 4
Average image recognition time.

Architecture	System components	Time, s
Software implementation	Convolutional layer	0,0380
	Other layers of CNN	0,0540
	Total time	**0,0920**
Hardware-software implementation	Image transfer to FPGA	0,0025
	Convolutional layer	0,0001
	Result transfer to workstation	0,0013
	Other layers of CNN	0,0540
	Total time	**0,0579**

The data interface between the FPGA unit and the workstation was USB 2.0. The transfer time of the image to the FPGA unit was 0.0025 s; the back transfer of the result to the workstation took 0.0013 s. The data on the average recognition time of one image are given in Table 4. Consequently, resting on the presented data on the system performance, we may conclude that the proposed hardware-software implementation allows reducing the average image recognition time by 37.06%.

The main conclusion from the results of software-hardware simulation of the CNN with RNS calculations in the convolutional layer is the reduced resource cost in comparison with the traditional BNS. The suggested architecture of the CNN can be widely used in the design of video surveillance systems as well as in the recognition of handwritten texts, persons, objects and terrain in different applications.

6. Conclusion

The existing architectures of CNNs are very demanding in terms of the hardware and time cost of a computing system, which considerably restricts their practical use in embedded systems, real-time systems, and mobile volatile devices. The main computational load of a CNN is required for the operation of convolutional layers, which involve the arithmetic operations of addition and multiplication. These operations can be efficiently implemented using RNS calculations, and this fact has been utilized above in a new CNN architecture. More specifically, we have proposed a new CNN architecture in which convolution layers are implemented at the hardware level on FPGA using RNS with the moduli of special kind and the other layers are implemented at the software level. To

further reduce the hardware and time cost of the CNN implementation, we have proposed a quantization method for the coefficients of the convolutional layer of the CNN, a compression technique for an array of partial products for the hardware implementation of convolution in the RNS and the reverse RNS-to-BNS conversion based on the CRTf. In accordance with the results of hardware simulation on Kintex7 xc7k70tfbg484-2 FPGA, the use of RNS in the convolutional layer of a neural network reduces hardware cost by 32.6% compared to the traditional approach based on the binary number system. In addition, the use of the proposed hardware-software architecture reduces the average image recognition time by 37.06% compared to the software implementation. The experimental results allow us to conclude that the proposed architecture of CNNs is more efficient than the well-known analogs in terms of the hardware and time cost. This fact expands the possibilities of practical implementation of CNNs in the area of image recognition.

Further improvement of the CNN implementation method proposed in this paper may include the development of new approaches to the structural implementation of CNNs on FPGA using tabular calculations in RNSs. Another field of further research is the use of the proposed CNN architecture in applications such as computer vision, speech recognition, identification of albuminous sequences in bioinformatics, production control, time series analysis in finance, and others.

Declaration of Competing Interest

None.

Acknowledgments

This work was supported by the basic part of the State order (no. 2.6035.2017/BCh), by the Russian Foundation for Basic Research (projects nos. 18-07-00109 A, 19-07-00130 A, 19-07-00856 A and 18-37-20059 mol-a-ved) and also by the Presidential Council for grants (projects MK-6294.2018.9, SP-126.2019.5 and SP-2245.2018.5).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.neucom.2020.04.018.

References

[1] J. Zhang, K. Shao, X. Luo, Small sample image recognition using improved Convolutional Neural Network, Journal of Visual Communication and Image Representation 55 (2018) 640–647.
[2] Y. Chen, S. Duffner, A. Stoian, J.-Y. Dufour, A. Baskurta, Deep and Low-level Feature based Attribute Learning for Person *Re*-identification, Image and Vision Computing 79 (2018) 25–34.
[3] X. Cheng, J. Lu, J. Feng, B. Yuan, J. Zhou, Scene recognition with objectness, Pattern Recognition 74 (2018) 474–487.
[4] S.S. Sarikan, A.M. Ozbayoglu, O. Zilcia, Automated Vehicle Classification with Image Processing and Computational Intelligence, Procedia Computer Science 114 (2017) 515–522.
[5] A. Qayyum, S.M. Anwar, M. Awais, M. Majid, Medical image retrieval using deep convolutional neural network, Neurocomputing 266 (2017) 8–20.
[6] A. Wong, M.J. Shafiee, M. St. Jules, MicronNet: A Highly Compact Deep Convolutional Neural Network Architecture for Real-Time Embedded Traffic Sign Classification, IEEE Access 6 (59) (2018) 803–859 810, doi:10.1109/ACCESS.2018.2873948.
[7] P. Swietojanski, A. Ghoshal, S. Renals, Convolutional Neural Networks for Distant Speech Recognition, IEEE Signal Process. Lett. 21 (1109) 1120–1124. doi:10.1109/LSP.2014.2325781.
[8] G. Aoki, Y. Sakakibara, Convolutional neural networks for classification of alignments of non-coding RNA sequences, Bioinformatics 34 (2018) i237–i244, doi:10.1093/bioinformatics/bty228.
[9] Z. Zhu, G. Peng, Y. Chen, H. Gao, A convolutional neural network based on a capsule network with strong generalization for bearing fault diagnosis, Neurocomputing 323 (2019) 62–75, doi:10.1016/j.neucom.2018.09.050.
[10] J.-F. Chen, W.-L. Chen, C.-P. Huang, S.-H. Huang, A.-P. Chen, Financial Time-Series Data Analysis Using Deep Convolutional Neural Networks, in: 2016 7th Int. Conf. Cloud Comput. Big Data, IEEE, 2016, pp. 87–92, doi:10.1109/CCBD.2016.027.
[11] D.G. Bailey, Design for embedded image processing on FPGAs, n.d. https://ieeexplore.ieee.org/book/6016259 (accessed September 12, 2019).
[12] Y. LeCun, L. Bottou, Y. Bengio, P. Haffiner, Gradient-based learning applied to document recognition, Proc. of the IEEE 86 (11) (1998) 2278–2324.
[13] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Advances in neural information processing systems 25 (2) (2012).
[14] B. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1–9.
[15] N. Jouppi, C. Young, N. Patil, D. Patterson, Motivation for and Evaluation of the First Tensor Processing Unit, IEEE Micro 38 (3) (2018) 10–19.
[16] S. Ahn, J. Kim, E. Lim, S. Kang, Soft Memory Box, A Virtual Shared Memory Framework for Fast Deep Neural Network Training in Distributed High Performance Computing, IEEE Access 6 (26) (2018) 493 –26,504, doi:10.1109/ACCESS.2018.2834146.
[17] X. Luo, Xiaona Yang, Weiping Wang, X. Chang, X. Wang, Zhigang Zhao, A novel hidden danger prediction method in cloud-based intelligent industrial production management using timeliness managing extreme learning machine, China Commun 13 (2016) 74–82, doi:10.1109/CC.2016.7559078.
[18] L. Guvenc, B.A. Guvenc, M.T. Emirler, Connected and Autonomous Vehicles, Internet Things Data Anal. Handb., John Wiley & Sons, Inc., Hoboken, NJ, USA, 2016, pp. 581–595, doi:10.1002/9,781,119,173,601.ch35.
[19] A.L.P. Tay, J.M. Zurada, Lai-Ping Wong, Jian Xu, The Hierarchical Fast Learning Artificial Neural Network (HieFLANN) – An Autonomous Platform for Hierarchical Neural Network Construction, IEEE Trans. Neural Networks. 18 (2007) 1645–1657, doi:10.1109/TNN.2007.900231.
[20] J. Wang, J. Lin, Z. Wang, Efficient convolution architectures for convolutional neural network, in: 2016 8th Int. Conf. Wirel. Commun. Signal Process., IEEE, 2016, pp. 1–5, doi:10.1109/WCSP.2016.7752726.
[21] J. Wang, J. Lin, Z. Wang, Efficient Hardware Architectures for Deep Convolutional Neural Network, IEEE Trans. Circuits Syst. I Regul. Pap. 65 (2018) 1941–1953, doi:10.1109/TCSI.2017.2767204.
[22] Y.-J. Lin, T.S. Chang, Data and Hardware Efficient Design for Convolutional Neural Network, IEEE Trans. Circuits Syst. I Regul. Pap. 65 (2018) 1642–1651, doi:10.1109/TCSI.2017.2759803.
[23] L. Gong, C. Wang, X. Li, H. Chen, X. Zhou, MALOC: A Fully Pipelined FPGA Accelerator for Convolutional Neural Networks with All Layers Mapped on Chip, IEEE Trans. Comput. Des. Integr. Circuits Syst. 37 (2018) 2601–2612, doi:10.1109/TCAD.2018.2857078.
[24] A. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, J. Cong, Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster, in: Proc. 2016 Int. Symp. Low Power Electron. Des. - ISLPED '16, ACM Press, New YorkUSA, 2016, pp. 326–331, doi:10.1145/2,934,583.2934644. New York.
[25] H. Nakahara, T. Sasao, A deep convolutional neural network based on nested residue number system, in: 2015 25th Int. Conf. F. Program. Log. Appl., IEEE, 2015, pp. 1–6, doi:10.1109/FPL.2015.7293933.
[26] H. Nakahara, T. Sasao, A High-speed Low-power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector, in: 2018 IEEE Int. Symp. Circuits Syst, IEEE, 2018, pp. 1–5, doi:10.1109/ISCAS.2018.8351850.
[27] S. Salamat, M. Imani, S. Gupta, T. Rosing, RNSnet: In-Memory Neural Network Acceleration Using Residue Number System, in: 2018 IEEE Int. Conf. Rebooting Comput., IEEE, 2018, pp. 1–12, doi:10.1109/ICRC.2018.8638592.
[28] T. Manabe, Y. Shibata, K. Oguri. "FPGA implementation of a real-time super-resolution system with a CNN based on a residue number system", 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, 2017, pp. 299–300.
[29] M. Abdelhamid, S. Koppula, Applying the Residue Number System to Network Inference, (2017). http://arxiv.org/abs/1712.04614 (accessed September 10, 2019).
[30] N.I. Chervyakov, P.A. Lyakhov, M.V. Valueva, Increasing of Convolutional Neural Network Performance Using Residue Number System, International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON) (2017) 135–140.
[31] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, (n.d.). http://yann.lecun.com/exdb/mnist/ (accessed June 16, 2019).
[32] H. Habibi Aghdam, E. Jahani Heravi, Guide to Convolutional Neural Networks, Springer International Publishing, Cham, 2017, doi:10.1007/978-3-319-57550-6.
[33] S.S. Haykin, Neural networks: a comprehensive foundation, Prentice Hall, 1999.
[34] A. Omondi, B. Premkumar, Residue Number Systems: Theory and Implementation, Imperial College Press, 2007, p. 296.
[35] G.C. Cardarilli, A. Nannarelli, M. Re., Residue number system for low-power DSP applications, in: Proc. 41st Asilomar Conf. Signals, Syst., Comput, 2007, pp. 1412–1416.
[36] H.T. Vergos, G. Dimitrakopoulos, On Modulo 2^n+1 Adder Design, IEEE Transactions on Computers 61 (2) (2012) 173–186.
[37] C. Parhami, Computer arithmetic: algorithms and hardware designs, Oxford University Press, 2010.

[38] P.M. Kogge, H.S. Stone, A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, IEEE Trans. Comput. C–22 (1973) 786–793, doi:10.1109/TC.1973.5009159.
[39] K.R. Rao, P.C. Yip, The Transform and Data Compression Handbook, CRC press, 2001, p. 399.
[40] C. Živaljević, N. Stamenković, V. Stojanović, Digital filter implementation based on the RNS with diminished-1 encoded channel, in: 2012 35th International Conference on Telecommunications and Signal Processing (TSP), Prague, 2012, pp. 662–666.
[41] R. de Matos, R. Paludo, N. Chervyakov, P.A. Lyakhov, H. Pettenghi, Efficient implementation of modular multiplication by constants applied to RNS reverse converters, in: 2017 IEEE Int. Symp. Circuits Syst.,, IEEE, 2017, pp. 1–4, doi:10.1109/ISCAS.2017.8050779.
[42] N.I. Chervyakov, A.S. Molahosseini, P.A. Lyakhov, M.G. Babenko, M.A. Deryabin, Residue-to-binary conversion for general moduli sets based on approximate Chinese remainder theorem, International journal of computer mathematics 94 (9) (2017) 1833–1849.
[43] Object Recognition Database, F. Rothganger, S. Lazebnik, C. Schmid and J. Ponce. http://www-cvr.ai.uiuc.edu/ponce_grp/data/objects (accessed February 15, 2019).
[44] N.I. Chervyakov, P.A. Lyakhov, D.I. Kalita, K.S. Shulzhenko, Effect of RNS dynamic range on grayscale images filtering, in: 2016 XV International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY), St. Petersburg, 2016, pp. 33–37.

Chervyakov Nikolai Ivanovich: Head of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Received Ph.D. degree from Stavropol State University in 1972. Defended his doctoral dissertation in 1987. Assistant professor since 1974 and professor since 1989 in Stavropol State University. Worked at Stavropol State University as lecturer (1968-1970), senior lecturer (1970-1975), head of Department of Algebra (1975-1994), professor of Department of Algebra (1994-2004). The head of Department of Applied Mathematics and Computer Science since 2004. Author of over 700 scientific publications and over 100 patents. Scientific advisor of over 70 doctoral and PhD dissertations. Scientific advisor and executor of more than 10 projects supported by the Russian Foundation for Basic Research. Research interests: modular arithmetic, artificial neural networks, digital image processing and cryptography.

Lyakhov Pavel Alekseevich: Associate Professor of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Senior Researcher, Department of Automation and Control Processes, Saint Petersburg Electrotechnical University "LETI", Saint Petersburg, Russia. Received master degree in mathematics from Stavropol State University in 2009, and Ph.D. degree from Stavropol State University in 2012. Research interests: high performance computing, modular arithmetic, artificial neural networks and digital image and signal processing.

Deryabin Maxim Anatolievich: Associate Professor at the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Received bachelor degree in mathematics from Stavropol State University in 2011, master degree in parallel computing from North-Caucasus Federal University in 2013, and Ph.D. degree from North-Caucasus Federal University in 2016. Research interests: modular arithmetic, residue number systems, FPGA, threshold cryptography, high performance computing.

Nagornov Nikolai Nikolaevich: Postgraduate student of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Received bachelor degree in applied mathematics and computer science from North-Caucasus Federal University in 2014, master degree in parallel computing from North-Caucasus Federal University in 2016. Research interests: modular arithmetic and digital image processing.

Valueva Maria Vasilyevna: Postgraduate student of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Received bachelor degree in applied mathematics and computer science from North-Caucasus Federal University in 2016, master degree in parallel computing from North-Caucasus Federal University in 2018. Research interests: high performance computing, modular arithmetic, artificial neural networks and digital image processing.

Valuev Georgii Vyacheslavovich: Master student of the Department of Applied Mathematics and Mathematical Modeling, North-Caucasus Federal University, Stavropol, Russia. Received bachelor degree in information systems and technologies from Don State University in 2016. Research interests: high performance computing, artificial intelligence and digital image processing.