Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 177 (2020) 232–243
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

Application of the residue number system to reduce hardware costs of the convolutional neural network implementation

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

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

Received 30 May 2019; received in revised form 10 February 2020; accepted 29 April 2020
Available online 6 May 2020

Abstract

Convolutional neural networks are a promising tool for solving the problem of pattern recognition. Most well-known convolutional neural networks implementations require a significant amount of memory to store weights in the process of learning and working. We propose a convolutional neural network architecture in which the neural network is divided into hardware and software parts to increase performance and reduce the cost of implementation resources. We also propose to use the residue number system (RNS) in the hardware part to implement the convolutional layer of the neural network. Software simulations using Matlab 2018b showed that convolutional neural network with a minimum number of layers can be quickly and successfully trained. The hardware implementation of the convolution layer shows that the use of RNS allows to reduce the hardware costs on 7.86%–37.78% compared to the two's complement implementation. The use of the proposed heterogeneous implementation reduces the average time of image recognition by 41.17%.

Keywords: Image processing; Convolutional neural networks; Residue number system; Quantization noise; Field-programmable gate array (FPGA).

1. Introduction

The modern development of science and technology implies the widespread introduction of data mining methods. The range of tasks requiring the use of artificial intelligence methods in the field of image processing is constantly expanding: personal identification [5], scene recognition [6], information processing from external sensors in unmanned land and aircraft vehicles [28], medical diagnostics [23], and so on. The use of intelligent methods based on artificial neural networks is a promising tool for solving the problem of image recognition [35].

The idea of using artificial neural networks for processing visual information was proposed in [15] to solve the problem of automating the recognition of handwritten numbers. The architecture proposed in this paper was called the Convolutional Neural Network (CNN). The combination of convolutional layers, realizing the extraction of visual signs, and a multilayer perceptron, realizing the recognition operation according to the convolution results, is its main feature. The evolution of this scientific idea and the development of computer technology have led to the fact

* Corresponding author.
E-mail address: sparta1392@mail.ru (N.N. Nagornov).

https://doi.org/10.1016/j.matcom.2020.04.031
0378-4754/

that at present the theory of CNN and its practical application methods are developing along the path of an extensive increase in the number of layers of CNN, which leads to the high computational complexity of the implementation of such systems. For example, the network architecture [14], which showed the best ImageNet image recognition result in 2010, contains about 650 thousand neurons, 60 million tunable parameters and requires 27 GB of disk space for training. The paper [30] presents the development of Google, which showed the best image recognition result of ImageNet base in 2014. This CNN performs more than one and a half billion computational operations for image recognition, which motivated Google to develop a special tensor processing unit to optimize the operation of this CNN [13]. Summarizing, modern CNN architectures are very resource intensive, which limits the possibilities for their wide practical application. A unified approach to reducing resource costs in the implementation of CNN in practice is not currently available.

The paper [3] presents the parallel hardware architecture Verilog HDL to neural network training, which may be synthesized for FPGA or ASIC. The method of acceleration of the training process of the neural network using FPGA is presented in the work [27]. The ASIC implementation to an acceleration of the training process of the deep neural network is shown in [19]. In this paper, the fully digital neuron was developed and its performance was optimized due to pipelining of the arithmetical unit with floating point format, and memory bandwidth problems were solved using the proposed architecture bandwidth memory. Then, several neurons were connected to create a small deep neural network, further, neurons were extrapolated to a standard dataset. The paper [26] aims to create fast and accuracy universal fuzzy neural network controller Neuro-Fuzzy based on FPGA hardware. The library VHDL was created to achieve this goal. The proposed controller is compared with its software realization based on the microprocessor. The results showed that the hardware controllers are faster and more accurate than software ones. During the learning phase, offline learning algorithms (based on Matlab's Neuro-Fuzzy toolbox) are used to estimate the controller's structure and the values of its parameters. The hardware implementation architecture of the softmax layer in the deep neural network is presented in [34]. To the neural network training, the use of the simplified activation function is proposed in [10]. In this paper are presented the hardware implementation of nonlinear activation function to apply in the different artificial neural network, including wavelet neural networks. Similar solutions can be used in fuzzy neural networks. The work [16] presents the accelerator of CNN, which optimizes the use of equipment and data bandwidth by the flexible and effective architecture to achieve CNN acceleration in real time. The efficient hardware architectures to acceleration deep CNN models are proposed in [33]. This paper presents the theoretical conclusion of the parallel fast algorithm with finite impulse response (FIR). The fast convolution units are developed by FIR. The authors propose new storage and data reuse schemes, in which all intermediate pixels are stored on a chip and bandwidth requirements are reduced. In the paper, one of the largest and most accurate networks VGG16 was chosen as an example and implemented on Xilinx Zynq ZC706 and Virtex VC707. The paper [11] proposes CNN architecture based on FPGA mapping all its layer on the chip and pipeline at the same time. A comprehensive mapping and optimization methodology is proposed. It based on the optimization model, which can ensure optimal using of resources and computational efficiency. Furthermore, to facilitate the programming process, the development environment is proposed, which can provide developers with a universal function to create the accelerator with the proposed optimization method. The recent existing methods of acceleration the deep learning network on FPGA are discussed in [29]. The author identifies key functions used by various methods to improve acceleration performance and provides recommendations on how to increase the use of FPGA to accelerate CNN.

The Residue Number System (RNS) is one of the promising tools for improving the technical characteristics of CNN [8,17,20,21]. A method using Sobel filters in the convolutional layer of CNN and its hardware FPGA implementation was proposed in [8]. The authors show an increase in the speed of the device and a reduction in equipment costs in comparison with the implementation of the two's complement representation. The disadvantage of the method presented in [8] is the fixation of the coefficients of the convolutional layer, which significantly slows down the learning time of CNN.

The goal of this paper is to reduce the hardware resources required to build and operate CNNs. We present the heterogeneous architecture of CNN in this paper, which is divided into hardware and software parts. We propose to use RNS with special moduli set in the hardware part, which implements the convolutional layer of CNN. For the hardware implementation of the convolution operation, a representation of the filter masks weights in a fixed-point format is required. To solve this problem, a quantization method for the coefficients of a convolutional layer is proposed. We will demonstrate the effectiveness of using the proposed approach with the help of hardware implementation on FPGA Xilinx.

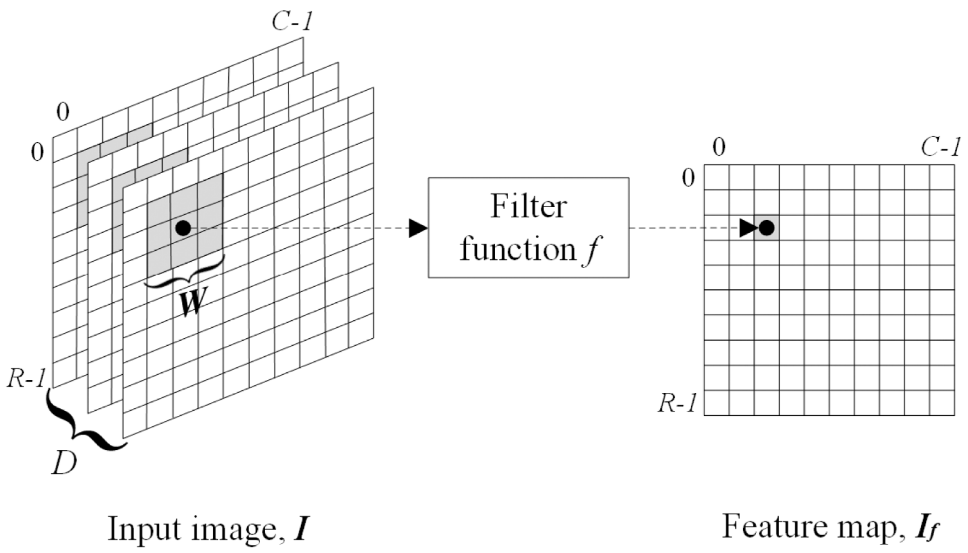

Fig. 1. The procedure of obtaining the feature maps in the convolutional layer of the CNN.

2. Convolutional neural networks

CNN consists of input and output layers, as well as several hidden layers. The hidden part of CNN consists of a convolutional layer, pooling layer, and a fully connected classifier, which is a perceptron that processes the features obtained on the previous layers. Consider the part of CNN, which is responsible for the selection of features, and consists of a spatial convolution layer and a pooling layer that implements the max pooling operation [15].

Suppose that the image I consisting of R rows, C columns and D color components is the input of CNN. Images in the RGB format have a $D = 3$ for example, and red, green and blue levels of image pixels are color components. The CNN input can be described as a three-dimensional function $I(x, y, z)$ in this case, where $0 \leq x < R$, $0 \leq y < C$ and $0 \leq z < D$ are the spatial coordinates, and the amplitude I at any point with coordinates (x, y, z) is the intensity of the pixels at that point. Procedure for obtaining characteristics in arrays convolutional layer can be represented as

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

where: I_f — feature maps; $W_{i,j,k}$ — $w \times w \times D$ filter for processing D two-dimensional arrays; b — offset [8] and $t = \frac{w-1}{2}$. The procedure for obtaining the feature maps is shown in Fig. 1.

CNN usually uses a large number of filters in the convolutional layer. This leads to a sharp increase in the amount of data processed in the network. The pooling layer of max pooling in some considered neighborhood is used to reduce them. Fig. 2 schematically shows a max pooling operation using a filter mask of size $w \times w$ and stride w. The output of this layer is transmitted to the input of the classifier, which is organized as a traditional multilayer perceptron.

We propose to use RNS to improve the technical characteristics of CNN. A brief description of RNS is given in the next section.

3. Background on RNS

The RNS numbers are represented in the basis of pairwise coprime numbers, called moduli $\{m_1, \ldots, m_n\}$. The product of all moduli RNS $M = m_1 m_2 \ldots m_n$ is called the dynamic range of the system. Any integer $0 \leq X < M$ can be uniquely represented in 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) [22]. The operations of addition, subtraction and multiplication in

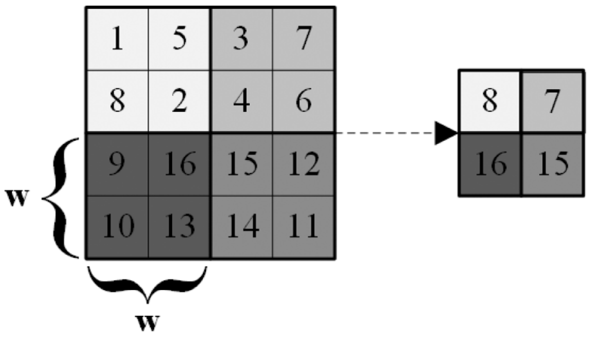

Fig. 2. The principle of operation of the pooling layer that implements the max pooling operation.

RNS are determined by the formulas

$$A \pm B = \left(|a_1 \pm b_1|_{m_1}, \ \ldots, \ |a_k \pm b_k|_{m_n} \right), \tag{2}$$

$$A \cdot B = \left(|a_1 \cdot b_1|_{m_1}, \ \ldots, \ |a_k \cdot b_k|_{m_n} \right). \tag{3}$$

Equalities (2) and (3) show the parallel nature of RNS, free of carrying propagation. The main advantages of representing numbers in RNS are [4]:

1. Large numbers are encoded into a set of small residues, which reduces the arithmetic complexity of computations.
2. Independence of the residues of the number from each other opens the possibility of their parallel processing, which allows speeding up the calculation process.
3. RNS is a nonpositional system with independent arithmetic units. Therefore, an error in one channel does not apply to others. Thus the processes of error detection and error correction are simplified.

The main disadvantages of representing numbers in RNS are [4]:

1. Inefficient implementation of nonmodular operations, such as division, extraction of roots, comparison of numbers.
2. The limitation of the action by the sphere of integer numbers.
3. The absence of simple signs of the output of the results of operations beyond the dynamic range of the system.

3.1. Selection of moduli RNS

The selection of the moduli set $\{m_1, \ldots, m_n\}$ is an important task in the development of an application system that uses computations in RNS. The modulus 2^α, $\alpha \in N$, where N — the set of natural numbers, is used in most modern works devoted to the application of RNS. This is explained by the fact that calculations modulo 2^α are the simplest from the point of view of hardware implementation since they can be implemented as ordinary arithmetic devices of a two's complement with a bit-width α. Using the modulus 2^α, as well as the requirement of pairwise coprime all moduli RNS leads to the fact that all other moduli must be odd, that is, to have a look $2^\alpha \pm k$, $\alpha \in N$, $k = 1, 3, 5, \ldots$. Since in this paper we use RNS to implement the most resource-consuming convolutional layer of CNN, which implements the calculations using formula (1), then the costliest, from a hardware point of view, the elements of an arithmetic unit will be modular adders. Since the implementation of calculations using formula (1) requires the summation of a large number of terms, it is necessary to use multi-input adders. Currently, the most effective techniques summing a large number of terms of absolute value, similar to those used in conventional two's complement, are designed only to form $2^\alpha \pm 1$, $\alpha \in N$ [31]. Practical implementation computing modulo $2^\alpha + 1$, $\alpha \in N$, requires the introduction of additional logic diminished-1 method for tracking the null code word. It is undesirable in the development system with minimal hardware and time consuming [32]. Thus, it is advisable to use a type of moduli $2^\alpha - 1$, $\alpha \in N$, as odd moduli, for design calculations on the basis of CNN in RNS.

3.2. Conversion from two's complement to RNS

Consider the moduli set of a special type $\{2^{p_1}, 2^{p_2}-1, \ldots, 2^{p_n}-1\}$. Conversion of the number in RNS is carried out by finding the remainder of the division for each modulus [22]. The operation of calculating the remainder of the modulo 2^p division is performed by leaving the p lower bits and dropping the remaining upper bits. Calculation of the remainder of the division for the modulus 2^p-1 is more complicated. Let $X = \overline{X_{g-1}X_{g-2}\ldots X_0}$ — g-bit enabled strong initial number. Divide it into $s = \lceil g/p \rceil$ parts of the p bits. We add to the X right 0 to the dimension $g' = s \cdot p$ for that. Now $X' = \overline{X_{g'-1}X_{g'-2}\ldots X_0}$, then $Y_0 = \overline{X_{p-1},\ldots,X_1,X_0}$, $Y_1 = \overline{X_{2p-1},\ldots,X_{p+1},X_p}$, …, $Y_s = \overline{X_{g'-1},\ldots,X_{(s-1)p+1},X_{(s-1)p}}$ — parts of X'. Imagine the number X' as $X' = Y_0 + Y_1\cdot 2^p + Y_2\cdot 2^{2p} + \cdots + Y_s \cdot 2^{sp}$. Transformations using number-theoretic properties give the following chain of equalities

$$\begin{aligned}
\left|X'\right|_{2^p-1} &= \left|Y_0 + Y_1\cdot 2^p + Y_2\cdot 2^{2p} + \cdots + Y_s\cdot 2^{sp}\right|_{2^p-1} \\
&= \left|\left|Y_0\right|_{2^p-1} + \left|Y_1\cdot 2^p\right|_{2^p-1} + \left|Y_2\cdot 2^{2p}\right|_{2^p-1} + \cdots + \left|Y_s\cdot 2^{sp}\right|_{2^p-1}\right|_{2^p-1} \\
&= \left|\left|Y_0\right|_{2^p-1} + \left|Y_1\cdot 2^p + Y_1 - Y_1\right|_{2^p-1} + \left|Y_2\cdot 2^{2p} + Y_2 - Y_2\right|_{2^p-1} + \cdots + \left|Y_s\cdot 2^{sp} + Y_s - Y_s\right|_{2^p-1}\right|_{2^p-1} \\
&= \left|\left|Y_0\right|_{2^p-1} + \left|Y_1\cdot\left(2^p-1\right) + Y_1\right|_{2^p-1} + \left|Y_2\cdot\left(2^{2p}-1\right) + Y_2\right|_{2^p-1} + \cdots + \left|Y_s\cdot\left(2^{sp}-1\right) + Y_s\right|_{2^p-1}\right|_{2^p-1} \\
&= \left|\left|Y_0\right|_{2^p-1} + \left|Y_1\right|_{2^p-1} + \left|Y_2\right|_{2^p-1} + \cdots + \left|Y_s\right|_{2^p-1}\right|_{2^p-1} = \left|Y_0 + Y_1 + Y_2 + \cdots + Y_s\right|_{2^p-1}
\end{aligned}$$

We get the following equality in this case

$$\left|X'\right|_{2^p-1} = \left|Y_0 + Y_1 + Y_2 + \cdots + Y_s\right|_{2^p-1}. \tag{4}$$

That is, the calculation of the remainder of the modulo 2^p-1 division is reduced to the addition of p-bit numbers modulo 2^p-1. The Kogge–Stone adders with cyclic modulo transfer, proposed in [31], are used for modulo 2^p-1 addition.

3.3. Conversion from RNS to two's complement

The use of the CRT is the most generalized method for obtaining an equivalent weighted number [9]. Calculate the weighted number X from the representation in the RNS, i.e. $(x_1, x_2, \ldots, x_n)$, based on moduli set $\{m_1, \ldots, m_n\}$ as follows

$$X = \left|\sum_{i=1}^{n} \left|M_i^{-1}\right|_{m_i} M_i x_i\right|_M, \tag{5}$$

where $M_i = M/m_i$ and $\left|M_i^{-1}\right|_{m_i}$ — is a multiplicative inverse M_i modulo m_i for $i = 1, 2, \ldots, n$. We must calculate the remainder of dividing by a large number M to implement CRT. The implementation of this operation in hardware leads to an increase in area and delay. The modified CRT, which uses fractional values and is called the approximate CRT, was first introduced in [12] to determine the sign and division in RNS. The effective hardware implementation of this approach based on the addendum compression technique and Kogge–Stone adder modulo 2^α is presented in [18]. The described method was used to implement the conversion from RNS to two's complement.

4. Convolution in the RNS

The use of RNS is most effective where the calculations require performing mainly operations of addition and multiplication. Formula (1) shows that the convolution uses only these operations. Thus, RNS can be effectively used for the hardware implementation of the convolutional layer of CNN. The high complexity of the implementation of the comparison operation in RNS makes it difficult to use it to implement the layer of max pooling and a fully connected layer of neurons. This prompts to propose an approach consisting in dividing CNN into hardware and software parts. It is proposed to use a hardware device to implement the convolutional part of CNN in RNS and use the software simulation for the remaining layers of CNN. Fig. 3 shows the convolution operation using RNS.

The coefficients $W_{i,j,k}$ and the offset b from formula (1) are constants in trained CNN. Therefore, the convolution device must implement multiplication by constants with the subsequent addition of the result. The hardware

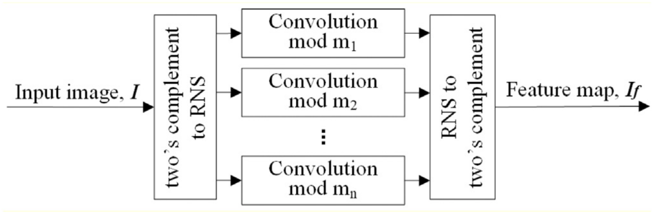

Fig. 3. The scheme of convolution in RNS.

implementation of calculations in the convolutional layer of CNN by formula (1) makes it necessary to quantize the filter coefficients $W_{i,j,k}$, which entails the emergence of quantization noise. The offset b from formula (1) has no effect when rounding the filter coefficients, therefore it is not associated with the error of the convolution result. Let the filter of the convolutional layer have a size $w \times w \times 3$, where 3 — is the number of color components. Multiply its coefficients by 2^N, where N — is the scaling parameter, and round up them. The result of the convolution in RNS divided into 2^N and rounded down. This approach has the following advantages.

1. Rounding errors will have different signs and partially compensate each other. This will reduce the effect of quantization noise of the convolution result.
2. Applying rounding operations in exactly this order requires less resource in hardware implementation than rounding operations to the nearest integer. This is explained by the fact that the coefficients of the filters are known a priori and their quantization with rounding up can be performed in advance. The convolution is performed using arithmetic logic devices and the rounding down of its result is performed simply by dropping the fractional part and does not require additional hardware and time costs.
3. The operations of multiplication and division by 2^N in binary notation correspond to a comma shift by N signs to the right or left, respectively, which greatly simplifies and speeds up their implementation.

The effect of quantization noise of the convolution result decreases with increasing value of the scaling parameter N. But the hardware and time costs increase for its execution. The question arises about the choice of a parameter value N, that is effective from the point of view of hardware implementation on modern devices, and which is necessary to maintain the convolution result of acceptable quality. We estimate the effect of quantization noise of the convolution result to answer this question. Define the absolute error AE_+ and AE_- of rounding positive and negative coefficients $W_{i,j,k}$ of formula (1), respectively

$$AE_+ = \sum_{W_{i,j,k}>0} \left(\lceil 2^N W_{i,j,k} \rceil - 2^N W_{i,j,k} \right), \quad AE_- = \sum_{W_{i,j,k}<0} \left(\lceil 2^N W_{i,j,k} \rceil - 2^N W_{i,j,k} \right). \tag{6}$$

Hence, we find the limited absolute error LAE_1 of rounding coefficients $W_{i,j,k}$

$$LAE_1 = \max \{AE_+, AE_-\}. \tag{7}$$

Denoting the maximum value of the brightness of the image as B, LAE_2 of the convolution result equal to

$$LAE_2 = LAE_1 \cdot B. \tag{8}$$

LAE_3 of normalized convolution result can be obtained from LAE_2 after taking into account the parameter N

$$LAE_3 = \frac{LAE_2}{2^N} = 2^{-N} B \cdot LAE_1. \tag{9}$$

If we denote by $\lambda \in [0, 1)$ the fractional part of the exact convolution result, then LAE_4 of rounding the normalized convolution result can be found as follows

$$LAE_4 = LAE_3 + \lambda - \lfloor LAE_3 + \lambda \rfloor \tag{10}$$

The exact value of a convolution is rarely an integer. Thus, LAE_4 depends not only on LAE_3, but also on λ. LAE_4 in formula (10) is an unrecoverable error when $LAE_3 = 0$. It correlates with LAE_3 and always influences the accuracy of the calculations of the researching method. The effect of this error on the result of calculations decreases with increasing N.

The resulting error of the convolution LAE_5 is rounded down the normalized convolution result.

$$LAE_5 = |LAE_3 - LAE_4| \, . \tag{11}$$

Formula (11) suggests that the rounding error to smaller convolution results LAE_4 partially compensates for the convolution result LAE_3, based on rounding up the filter coefficients $W_{i,j,k}$. These two errors will become commensurate in their absolute value at a certain stage of increase N. The resulting error LAE_5 will have the least value at this stage in practice. The error LAE_3 will begin to decrease with a further increase N, in contrast to LAE_4. This will increase LAE_5.

The λ is one of the parameters affecting the magnitude of the LAE_5. The specific value of λ when conducting theoretical calculations cannot be determined. Eliminate it, given the impact they have on the result of calculations. Express in the formula (11) LAE_4 through LAE_3 and λ according to formula (10) for this.

$$LAE_5 = |LAE_3 - (LAE_3 + \lambda - \lfloor LAE_3 + \lambda \rfloor)| = |\lfloor LAE_3 + \lambda \rfloor - \lambda| \, . \tag{12}$$

Let us consider two cases:

1. $\lfloor LAE_3 + \lambda \rfloor - \lambda > 0 \Rightarrow \lfloor LAE_3 + \lambda \rfloor \geq 1$. The larger $\lfloor LAE_3 + \lambda \rfloor$, the larger LAE_5 in this case. Thus, $\lfloor LAE_3 + \lambda \rfloor = \lfloor LAE_3 \rfloor + 1$ and λ represents the addition of the LAE_3 fractional part to one, that is $\lambda = \lfloor LAE_3 \rfloor + 1 - LAE_3$. Substitute the resulting expression in formula (12).

$$LAE_5 = \lfloor LAE_3 + \lfloor LAE_3 \rfloor + 1 - LAE_3 \rfloor - (\lfloor LAE_3 \rfloor + 1 - LAE_3) = LAE_3. \tag{13}$$

2. $\lfloor LAE_3 + \lambda \rfloor - \lambda \leq 0 \Rightarrow \lfloor LAE_3 + \lambda \rfloor \leq \lambda \Rightarrow \lfloor LAE_3 + \lambda \rfloor = 0 \Rightarrow LAE_5 = |0 - \lambda| = \lambda$. The larger λ, the larger LAE_5 in this case. But $\lfloor LAE_3 + \lambda \rfloor = 0 \Rightarrow LAE_3 + \lambda = 1 - \varepsilon \Rightarrow \lambda = 1 - \varepsilon - LAE_3$. Using AE_3 instead of LAE_3, and equating its value to zero. Thus, formula (11) takes the following form.

$$LAE_5 = |\lfloor 0 + 1 - \varepsilon \rfloor - (1 - \varepsilon)| = 1 - \varepsilon, \tag{14}$$

where ε — machine epsilon.

Formula (13) is advantageous to use for $LAE_3 \geq 1$ since the value LAE_5 is the maximum possible error value. Thus, LAE_5 can be uniquely determined depending on the values LAE_3 and independently of specific values λ from formulas (13) and (14).

$$LAE_5 = \begin{cases} LAE_3, & LAE_3 \geq 1, \\ 1 - \varepsilon, & LAE_3 < 1. \end{cases} \tag{15}$$

Rewrite formula (15) using expression (9).

$$LAE_5 = \begin{cases} 2^{-N} B \cdot LAE_1, & 2^{-N} B \cdot LAE_1 \geq 1, \\ 1, & 2^{-N} B \cdot LAE_1 < 1; \end{cases} \tag{16}$$

Peak signal-to-noise ratio ($PSNR$) after performing these actions can be found by follows.

$$PSNR = 10 \lg \frac{B^2}{LAE_5^2} = 20 \lg \frac{B}{LAE_5}. \tag{17}$$

Transform formula (17) using expression (16).

$$PSNR = \begin{cases} 20 \lg \dfrac{2^N}{LAE_1}, & 2^N \leq B \cdot LAE_1, \\ 20 \lg \dfrac{B}{1 - \varepsilon}, & 2^N > B \cdot LAE_1. \end{cases} \tag{18}$$

Formula (18) describes the dependence $PSNR$ of the convolution result according to formula (1) on the value of the scaling parameter N, the size $w \times w \times 3$ of the filter convolutional layer and the maximum brightness value B of the image. The practical application of formula (18) can be described as follows: it is necessary to choose the smallest values of N, which allows maintaining an acceptable quality of the processed image. This will minimize hardware costs for the implementation of convolution. The bit-width r of the filter coefficients after quantization is determined by follows.

$$r = \max_{\substack{-t \leq i \leq t \\ -t \leq j \leq t \\ 0 \leq k \leq D-1}} \{r_{i,j,k}\}, \tag{19}$$

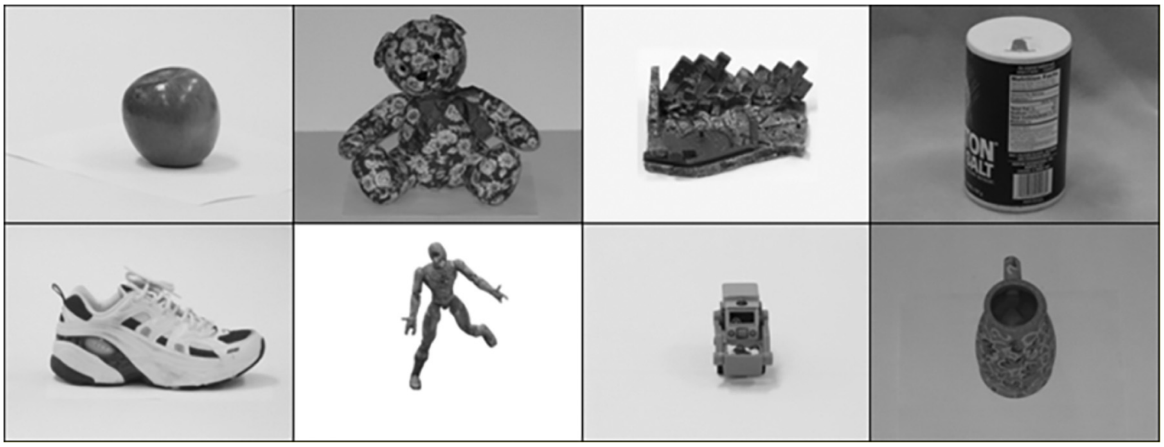

Fig. 4. Database image classes [25].

where $r_{i,j,k}$ — the number of digits required to store the quantized coefficient $\lceil 2^N W_{i,j,k} \rceil$, determined by follows.

$$r_{i,j,k} = 1 + \left\lceil \log_2 \left(\left| \lceil 2^N W_{i,j,k} \rceil \right| + 1 \right) \right\rceil . \tag{20}$$

Software simulation using derived formulas is presented below. The principle of formulas is demonstrated by the example of a specific filter.

5. Software simulation

CNN was developed for recognizing 8 classes of 161 images from the University of Illinois image base [25] as an experimental base. The image classes from this dataset are shown in Fig. 4. All base images are converted to resolution 256 × 192 pixel using the bicubic interpolation algorithm embedded in Adobe Photoshop CS6.

Minimizing the hardware and time spent on training and implementation of CNN was the main purpose of the simulation. The minimum possible number of layers used for that. In addition, RNS arithmetic is used in the convolutional layer of CNN instead of two's complement.

The developed CNN architecture is shown in Fig. 5. The RGB images of size 256 × 192 are fed to CNN input. The first two layers are responsible for highlighting features of the image. The first layer performs a convolution using 8 filters of size 3 × 3 × 3 and stride 3. 8 feature maps of size 85 × 64 are the calculation result of the first layer are. The second layer performs the max pooling operation using a mask with a size 2 × 2 and stride 2. 8 feature maps of size 42 × 32 are the output of the second layer and are connected to the inputs of the last two layers responsible for the image classification. The third layer consists of 10 neurons, and the fourth layer consists of 8 neurons, each of which corresponds to a particular class.

The convolution takes a significant part of the working time. We divided CNN architecture into hardware and software parts to increase the speed of work. The convolutional layer is the hardware part implemented on the FPGA using RNS calculations. The layer of max pooling and a fully connected network are implemented in the simulation since the comparison operations and the nonlinear activation function are difficult to implement in RNS.

CNN is trained in Matlab R2018b using the Neural Pattern Recognition Toolbox tool. The calculations were performed on a PC with an Intel (R) Core (TM) i7-4790K CPU @ 4.00 GHz processor with 16.0 GB of RAM and 64-bit Windows 10. 161 images from 8 classes [25] were used for training. The neural network is trained in 30 iterations within 57 s.

3D mask of one filter from the convolutional layer of trained CNN before and after quantization is shown in Table 1. The values of the filter coefficients are reduced to an 8-bit representation for the hardware implementation. Bit-width $r = 8$ is determined by the formula (19) at $N = 11$. The selected value of scaling parameter N is the smallest, at which the result of image processing with a filter of size 3 × 3 × 3 is guaranteed to reach 40 dB quality according to formula (18). This value describes the difference between the two images, almost imperceptible to the viewer [24].

Hardware implementation using quantized filter mask from Table 1 is presented below.

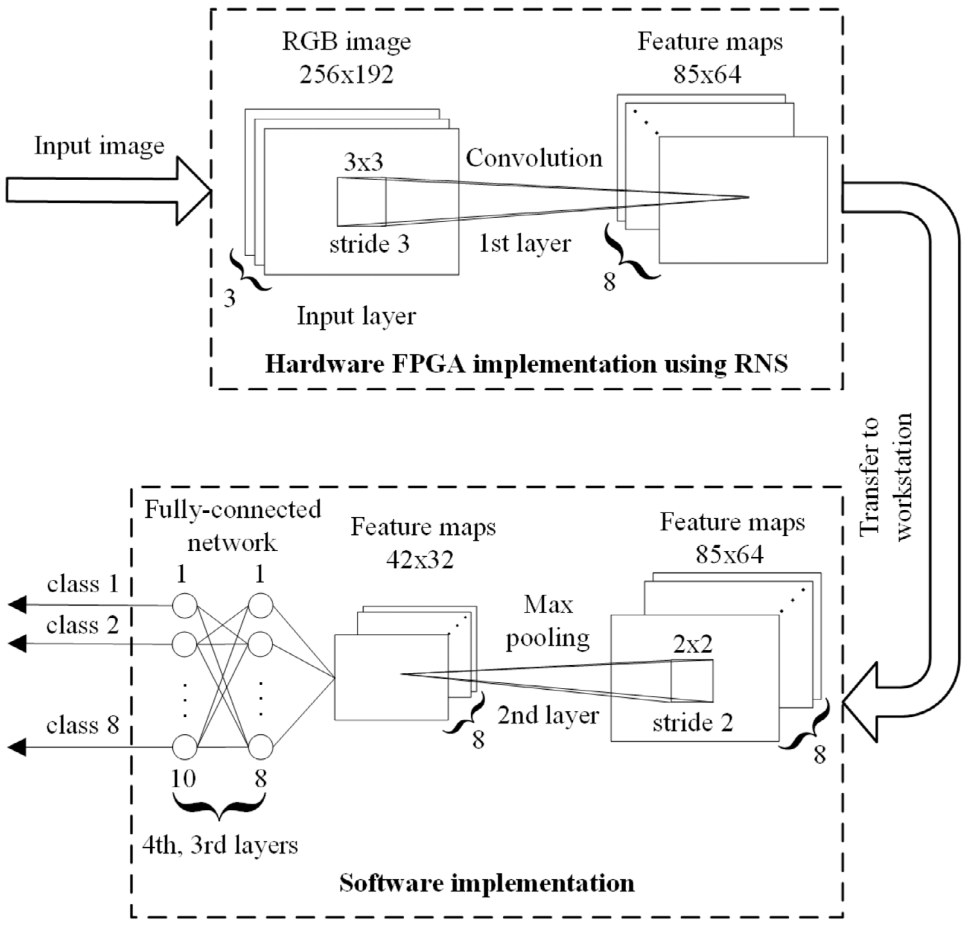

Fig. 5. Proposed CNN architecture using RNS in the convolution layer.

Table 1
3D mask of one filter from the convolutional layer of trained CNN.

Layer	Filter mask before quantization	Filter mask after quantization
R	$\begin{pmatrix} -0.0394456 & -0.0257306 & -0.0239846 \\ -0.0312350 & -0.0403409 & -0.0520126 \\ -0.0434387 & -0.0364724 & -0.0520036 \end{pmatrix}$	$\begin{pmatrix} -80 & -52 & -49 \\ -63 & -82 & -106 \\ -88 & -74 & -106 \end{pmatrix}$
G	$\begin{pmatrix} -0.0280364 & -0.0397294 & -0.0528939 \\ -0.0423464 & -0.0429127 & -0.0613428 \\ -0.0128505 & -0.0548165 & -0.0599076 \end{pmatrix}$	$\begin{pmatrix} -57 & -81 & -108 \\ -86 & -87 & -125 \\ -26 & -112 & -122 \end{pmatrix}$
B	$\begin{pmatrix} -0.0304336 & -0.0289807 & -0.0600239 \\ -0.0578725 & -0.0404073 & -0.0482339 \\ -0.0492809 & -0.0410201 & -0.0474128 \end{pmatrix}$	$\begin{pmatrix} -62 & -59 & -122 \\ -118 & -82 & -98 \\ -100 & -84 & -97 \end{pmatrix}$
Bias	−0.0003320	0

6. Hardware implementation

Hardware implementation was performed on FPGA Kintex7 xc7k480tffg901-2 in Xilinx Vivado 18.3 with the parameters “High Performance Optimized”. Comparison of RNS and two’s complement is the main purpose of the implementation.

Table 2
The delay of the convolution operation of the CNN for different moduli.

Modulus	Delay (ns)	LUTs	$A \times D$
2^2-1	6.868	74	508,232
2^3-1	7.461	126	940,086
2^4-1	8.470	252	2134,440
2^5-1	8.936	409	3654,824
2^6-1	9.510	628	5972,280
2^7-1	9.758	842	8216,236
2^8-1	10.042	1150	11548,300
2^9-1	9.862	1395	13757,490
$2^{10}-1$	10.041	1930	19379,130
2^7	6.955	345	2399,475
2^8	7.696	456	3509,376
2^9	8.052	611	4919,772
2^{10}	8.396	806	6767,176

Hardware implementation results are presented for only one filter but can be generalized to the other seven since they have the same structure and differ only in weighting coefficients. The FPGA implementation of the CNN filters was implemented in the form of a combinational circuit without using memory to achieve maximum performance. Kogge Stone Adders (KSAs) and Carry Save Adders used for the implementation of the convolution operation [31]. Thus, all eight filters of the convolutional layer of the considered CNN can be implemented. At the same time, the universality of the implemented approach allows us to generalize the conclusion obtained for one filter to all others. The distribution and general use of hardware blocks Slice and LUT Slices with this approach are performed automatically based on algorithms built into the Computer-aided design (CAD) system (in our case Xilinx Vivado 18.3). We used VHDL to develop the hardware architectures. All source files are available at [1].

Implementation of a convolution is performed for various moduli of the form 2^p and 2^p-1. The results are presented in Table 2. We use metric $A \times D$ to compare implementation results which is calculated as the product of the number of LUTs by the device delay and allows us to estimate the area and time resources consumed by the device. The delay varies from 6.868 ns to 10.042 ns.

The dynamic range of RNS must satisfy the condition $M \geq 2 \cdot 255 \cdot \max\{0, 2326\} = 1186260$ [7], where: 2 — factor, the presence of which is explained by the need to represent negative numbers in RNS; 255 — the maximum brightness value of the image; 0 — the sum of positive filter coefficients; 2326 — module of the sum of its negative coefficients. This condition, as well as the data from Table 2, allows us to choose to implement a complete RNS system with three sets of moduli $\left\{2^3-1,\ 2^4-1,\ 2^5-1,\ 2^9\right\}$, $\left\{2^4-1,\ 2^9-1,\ 2^9\right\}$ and $\left\{2^5-1,\ 2^7,\ 2^9-1\right\}$, having the lowest delay and spanning the dynamic range M, for conversion from two's complement to RNS, convolution and reverse conversion from RNS to two's complement.

The obtained implementation results using RNS and two's complement are presented in Table 3. The implementation showed that using RNS with moduli set $\left\{2^3-1,\ 2^4-1,\ 2^5-1,\ 2^9\right\}$ reduces the hardware costs by 7.86%, $\left\{2^5-1,\ 2^7,\ 2^9-1\right\}$ — by 2.07% compared to the two's complement. The use of moduli set $\left\{2^4-1,\ 2^9-1,\ 2^9\right\}$ allows to reduce the device area, but it increases the device delay, therefore this moduli set requires 8.26% more hardware resources compared to the two's complement. This allows us to conclude that the use of RNS for hardware implementation of the convolutional layer of CNN is more efficient in the area of the device, compared to the implementation in two's complement. The generalization of the results obtained in the case of large filter masks requires further research in practice. The comparison was made with the implementation of the convolution function of a single CNN filter obtained by the HDL Coder converter built into Matlab into a code written in VHDL to demonstrate the advantages of the proposed method. Consequently, the proposed approach using RNS $\left\{2^3-1,\ 2^4-1,\ 2^5-1,\ 2^9\right\}$ reduces the hardware resources by 37.78% compared to automatically generated code in Matlab.

The difference between the two's complement implementations is the use of different adders. In the case of our two's complement implementation, we used KSAs, which can reduce the delay, but lead to an increase in the device area. And in the case of the automatically generated code by Matlab, carry propagate adders were used, which increase the delay, but reduce the area of the device.

Table 3
Hardware implementation results of the CNN convolutional layer.

Number system	Parameters	Delay (ns)	LUTs	$A \times D$
RNS	$\{2^3-1, 2^4-1, 2^5-1, 2^9\}$	16.251	2900	47127.900
	$\{2^4-1, 2^9-1, 2^9\}$	15.975	3490	55752.750
	$\{2^5-1, 2^7, 2^9-1\}$	16.040	3123	50092.920
Two's complement	Using KSAs	12.746	4013	51149.698
	Generated by Matlab	33.124	2324	76980.176

Table 4
Average image recognition time.

Architecture	System components	Time (s)
Software implementation	Convolutional layer	0.038000
	The remaining CNN layers	0.054000
	Total processing time in the software implementation	**0.092000**
Proposed heterogeneous architecture	Image transfer to FPGA	0.000015
	Convolutional layer	0.000089
	Result transfer to the workstation	0.000018
	The remaining CNN layers	0.054000
	Total processing time in the hardware–software implementation	**0.054122**

The software part of CNN is implemented in Matlab. We use AXI4 as the data interface between the FPGA module and the workstation [2]. The transfer time of the image to the FPGA module is 0.000015 s, and the return transfer of the result to workstation takes 0.000018 s. Data on the average recognition time of one image is shown in Table 4. Thus, we can conclude that the use of the proposed hardware and software implementation reduces the average time of image recognition by 41.17%.

The fact that resource costs are reduced compared to using traditional two's complement representation is the main conclusion on the results of software simulation and hardware implementation of CNN with calculations in RNS in the convolutional layer. The developed architecture of CNN can be widely used to create efficient video surveillance systems, in the recognition of handwriting text, persons, objects and terrain in various practical applications.

7. Conclusion

The paper presents a method of the hardware implementation of the CNN for pattern recognition using computations in the RNS. The minimized configuration of the CNN includes a convolutional layer, a layer of max pooling, and a classifier, which is organized as a traditional multilayer perceptron. The hardware implementation of the convolution layer shows that the use of RNS allows to reduce the hardware costs on 7.86%–37.78% compared to the two's complement. The use of the proposed heterogeneous implementation reduces the average time of image recognition by 41.17%. The generalization of the obtained results in the case of large filter masks requires further research in practice. The research results can be applied to create effective video surveillance systems, for recognizing handwriting, individuals, objects and terrain.

Acknowledgment

This work was supported by the Russian Science Foundation, Russia (project no. 19-19-00566).

References

[1] Zombo26 / conv – Bitbucket [Electronic resource] – Access mode: https://bitbucket.org/Zombo26/conv/src/master/.

[2] Vivado AXI reference [optional] vivado design suite AXI reference guide, 2017, [Electronic resource] – Access mode: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.

[3] A. Bezborah, A hardware architecture for training of artificial neural networks using particle swarm optimization, in: Third International Conference on Intelligent Systems Modelling and Simulation, 2012, pp. 67–70.

[4] G.C. Cardarilli, A. Nannarelli, M. Re, Residue number system for low-power DSP applications, in: Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007, pp. 1412–1416.

[5] Y. Chen, S. Duffner, A. Stoian, J.-Y. Dufour, A. Baskurta, Deep and low-level feature based attribute learning for person re-identification, Image Vis. Comput. 79 (2018) 25–34.

[6] X. Cheng, J. Lu, J. Feng, B. Yuan, J. Zhou, Scene recognition with objectness, Pattern Recognit. 74 (2018) 474–487.

[7] N.I. Chervyakov, P.A. Lyakhov, D.I. Kalita, K.S. Shulzhenko, Effect of RNS dynamic range on grayscale images filtering, in: XV International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY), 2016, pp. 33–37.

[8] N.I. Chervyakov, P.A. Lyakhov, M.V. Valueva, Increasing of Convolutional Neural Network performance using residue number system, in: International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), 2017, pp. 135–140.

[9] 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, Int. J. Comput. Math. 94 (9) (2017) 1833–1849.

[10] Z. Dlugosz, R. Dlugosz, Nonlinear activation functions for artificial neural networks realized in hardware, in: 25th International Conference Mixed Design of Integrated Circuits and System (MIXDES), 2018, pp. 381–384.

[11] 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.-Aided Des. Integr. Circuits Syst. 37 (11) (2018) 2601–2612.

[12] C.Y. Hung, B. Parhami, An approximate sign detection method for residue numbers and its application to RNS division, Comput. Math. Appl. 27 (4) (1994) 23–25.

[13] 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.

[14] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep Convolutional Neural Networks, Adv. Neural Inf. Process. Syst. 25 (2) (2012).

[15] Y. LeCun, L. Bottou, Y. Bengio, P. Haffiner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.

[16] Y. Lin, T.S. Chang, Data and hardware efficient design for convolutional neural network, IEEE Trans. Circuits Syst. I. Regul. Pap. 65 (5) (2018) 1642–1651.

[17] T. Manabe, Y. Shibata, K. Oguri, FPGA implementation of a real-time super-resolution system with a CNN based on a residue number system, in: International Conference on Field Programmable Technology (ICFPT), 2017, pp. 299–300.

[18] R. de Matos, R. Paludo, N.I. Chervyakov, P.A. Lyakhov, H. Pettenghi, Efficient implementation of modular multiplication by constants applied to RNS reverse converters, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1–4.

[19] R. Mehta, Y. Huang, M. Cheng, S. Bagga, N. Mathur, J. Li, J. Draper, S. Nazarian, High performance training of deep neural networks using pipelined hardware acceleration and distributed memory, in: 19th International Symposium on Quality Electronic Design (ISQED, 2018, pp. 383–388.

[20] H. Nakahara, T. Sasao, A deep convolutional neural network based on nested residue number system, in: 25th International Conference on Field Programmable Logic and Applications (FPL), 2015, pp. 1–6.

[21] 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: IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1–5.

[22] A. Omondi, B. Premkumar, Residue Number Systems: Theory and Implementation, Imperial College Press, 2007, p. 296.

[23] A. Qayyum, S.M. Anwar, M. Awais, M. Majid, Medical image retrieval using deep convolutional neural network, Neurocomputing 266 (2017) 8–20.

[24] K.R. Rao, P.C. Yip, The Transform and Data Compression Handbook, CRC press, 2001, p. 399.

[25] F. Rothganger, S. Lazebnik, C. Schmid, J. Ponce, Object Recognition Database [Electronic resource] – Access mode: http://www-cvr.ai.uiuc.edu/ponce_grp/data/objects.

[26] A. Sameh, M.S.A.A.E. Kader, Generic floating point library for neuro-fuzzy controllers based on FPGA technology, in: Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004, pp. 369–372.

[27] R. Sang, Q. Liu, Q. Zhang, FPGA-based acceleration of neural network training, in: IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), 2016, pp. 1–2.

[28] S.S. Sarikan, A.M. Ozbayoglu, O. Zilcia, Automated vehicle classification with image processing and computational intelligence, Procedia Comput. Sci. 114 (2017) 515–522.

[29] A. Shawahna, S.M. Sait, A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, IEEE Access 7 (2019) 7823–7859.

[30] C. Szegedy, W. Liu, Y. Jia1, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[31] H.T. Vergos, G. Dimitrakopoulos, On Modulo 2∧n+1 Adder design, IEEE Trans. Comput. 61 (2) (2012) 173–186.

[32] D. Živaljević, N. Stamenković, V. Stojanović, Digital filter implementation based on the RNS with diminished-1 encoded channel, in: 35th International Conference on Telecommunications and Signal Processing (TSP), 2012, pp. 662–666.

[33] J. Wang, J. Lin, Z. Wang, Efficient hardware architectures for deep convolutional neural network, IEEE Trans. Circuits Syst. I. Regul. Pap. 65 (6) (2018) 1941–1953.

[34] B. Yuan, Efficient hardware architecture of softmax layer in deep neural network, in: 29th IEEE International System-on-Chip Conference (SOCC), 2016, pp. 323–326.

[35] J. Zhang, K. Shao, X. Luo, Small sample image recognition using improved Convolutional Neural Network, J. Vis. Commun. Image Represent. 55 (2018) 640–647.