Area-Efficient FPGA Implementation of Minimalistic Convolutional Neural Network Using Residue Number System

Nikolay I. Chervyakov, Pavel A. Lyakhov,
Maria V. Valueva, Georgii V. Valuev
North-Caucasus Federal University
Stavropol, Russian Federation
k-fmf-primath@stavsu.ru

Dmitrii I. Kaplun, George A. Efimenko,
Denis V. Gnezdilov
Saint Petersburg Electrotechnical University "LETI"
St. Petersburg, Russian Federation
dikaplun@etu.ru

Abstract—**Convolutional Neural Networks (CNN) is the promising tool for solving task of image recognition in computer vision systems. However, the most known implementation of CNNs require a significant amount of memory for storing weights in training and work. To reduce the resource costs of CNN implementation we propose the architecture that separated on hardware and software parts for performance optimization. Also we propose to use Residue Number System (RNS) arithmetic in the hardware part which implements the convolutional layer of CNN. Software simulation using Matlab 2017b shows that CNN with a minimum number of layers can be quickly and successfully trained. Hardware simulation using FPGA Kintex7 xc7k70tfbg484-2 demonstrates that using RNS in convolutional layer of CNN allows to reduce hardware costs by 32% compared with the traditional approach based on the binary number system.**

I. Introduction

Convolutional Neural Networks (CNN) is the promising tool for solving task of image recognition. The idea of CNN is based on human vision system. The brain performs successively a number of recognition tasks, for example, recognizing a familiar face in an unfamiliar environment. CNN-based algorithms are widely used in embedded machine vision systems which includes the solution of handwriting recognition problems [1], face detection [2], locating [3] and object recognition [4]. Neural networks have a number of advantages that distinguish them among approaches to solving problems of artificial intelligence. The main of them are parallelization of information processing and self-learning ability, i.e. creating of generalizations [4]. The most known CNN realizations require a significant amount of memory for storing weights in training and work [1], [5], [6]. This makes the problem of searching for minimalistic realizations of CNN relevant.

The idea of using artificial neural networks for visual information processing was proposed in [1] to solve a problem of automation of digit handwriting recognition. The architecture proposed in this article was called the Convolutional Neural Network (CNN) and its main feature was union convolution layers and multilayer perceptron. The evolution of this scientific idea and the development of computer technology have led to the fact 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. This leads to a high computational complexity of the implementation of such systems. For example, The architecture of network [7] showing the best image recognition result of ImageNet database in 2010 consists about 650000 neurons, 60 million custom settings and requires 27 gigabytes of disk space for training. In [8] presents the development of Google, which showed the best image recognition result of ImageNet in 2014. For image recognition this CNN performs over one and a half billion computing operations. This motivated Google to develop a special tensor processor to optimize the performance of this CNN [9]. In conclusion, modern CNN architectures are resource intensive, that severely limits their wide practical application.One of the way to improve CNN performance is hardware implementation [10 - 13].

The promising tool for performance improvement of CNN is the Residue Number System (RNS) arithmetic. The method using Sobel filters in convolutional layer of CNN and its FPGA hardware implementation by using RNS was proposed in [14]. Authors demonstrates increasing of device speed and reduce hardware costs compared by Binary Number System (BNS) realization. The disadvantage of method proposed in [14] is fixing the coefficients of the convolutional layer which significantly slows down the training time of CNN. To overcome the shortcomings of the approach from [14] we will present in this paper the architecture of CNN which separated on hardware and software parts. We propose to use RNS in the hardware part which implements the convolutional layer of CNN. We will demonstrate area-efficiency of the proposed approach by hardware modeling using FPGA Xilinx.

The article should contain the following structural components: Convolutional Neural Networks, background on RNS, CNN architecture and training, simulation results and conclusions.

II. Convolutional Neural Networks

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN consist of convolutional layers, pooling layers, fully connected layers and normalization layers. In this article we will use the feature extraction part consists of alternating spatial convolutional layers and max pooling layers [5].

Suppose that the CNN input receives an image I consisting of R rows, C columns and D layers. This means that the CNN input can be described as a three-dimensional function $I(x,y,z)$, where $0 \le x < R$, $0 \le y < C$ and $0 \le z < D$ are spatial coordinates, and an amplitude I at any point with coordinates (x,y,z) is pixel intensity at this point. The procedure for obtaining feature maps in the convolutional layer can be represented in the form:

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

where I_f is the feature map, $W_{i,j,k}$ are 3D-filter coefficients for processing D two-dimensional arrays and b is bias [14]. The procedure for obtaining feature maps is shown schematically in Fig.1.

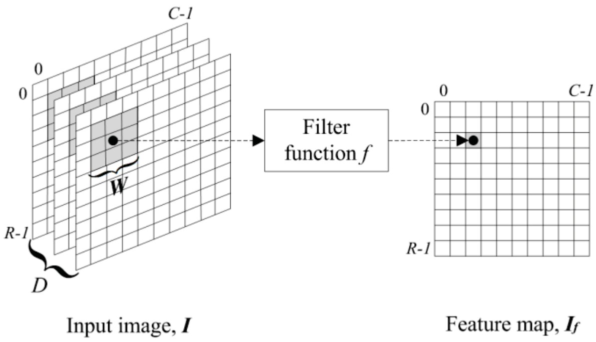

Fig. 1. The procedure for feature maps obtaining

CNN typically use a large number of filters in the convolutional layer. This leads to a sharp increase in the amount of data within the network. Max pooling layer of is used to reduce this volume. Fig. 2 shows schematically the max pooling procedure by using $m \times m$ filter mask and stride m. The output of this layer transfers to the input of the recognition classifier, which is organized as the traditional multi-layer perceptron neural network.

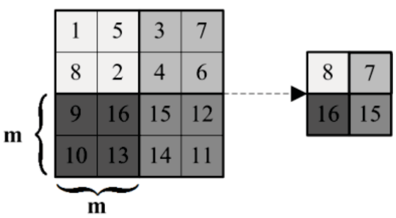

Fig. 2. The max pooling procedure for feature map

As an experimental base, we developed a CNN for 8 patterns recognizing in the sample image database of the University of Illinois [15]. Image classes from that dataset are shown in Fig. 3. Fig. 4 shows example of images from one class. The images size of database was unified to 256×192 pixels using the Adobe Photoshop CS6 software by the bicubic interpolation algorithm. 161 images from database were used for CNN training.

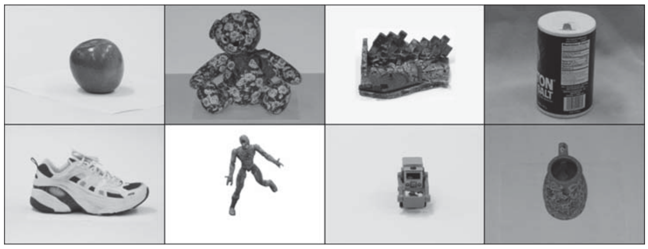

Fig. 3. Image classes from database [15]

We set the main goal of minimizing the structure of the CNN. For this purpose, we tried to use the minimum possible number of CNN layers. In addition, we used the RNS arithmetic instead of traditional binary arithmetic, where it was possible.

III. Background on RNS

In RNS, numbers are represented in the basis of mutually prime numbers $\{m_1,...,m_n\}$, $\gcd(m_i, m_j) = 1$, $i \ne j$ called modules. The product of all RNS modules $M = m_1 m_2 ... m_n$ is called the dynamic range of the system. Any integer $0 \le X < M$ can be uniquely represented in RNS as a vector $\{x_1, x_2, ..., x_n\}$, where $x_i = |X|_{m_i} = X \bmod m_i$ in accordance with Chinese Remainder Theorem (CRT) [16].

The addition, subtraction and multiplication operations in RNS are defined by formulas

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

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

Equalities (2) – (3) show the parallel nature of RNS, free of bitwise shifts. Thus, the advantages of representing numbers in RNS can be represented as follows [17].

Choosing moduli set is an important issue in RNS design. Special type of moduli set $\{2^{p_1}, 2^{p_2}-1, ..., 2^{p_n}-1\}$ allows to use high-speed algorithms for addition, multiplication, forward and reverse conversion [18], [19].

A. Binary to RNS conversion

We consider a special moduli set $\{2^{p_1}, 2^{p_2}-1, ..., 2^{p_n}-1\}$. It is necessary to calculate the remainder of the division by each of the moduli to conversion a number into RNS [16].

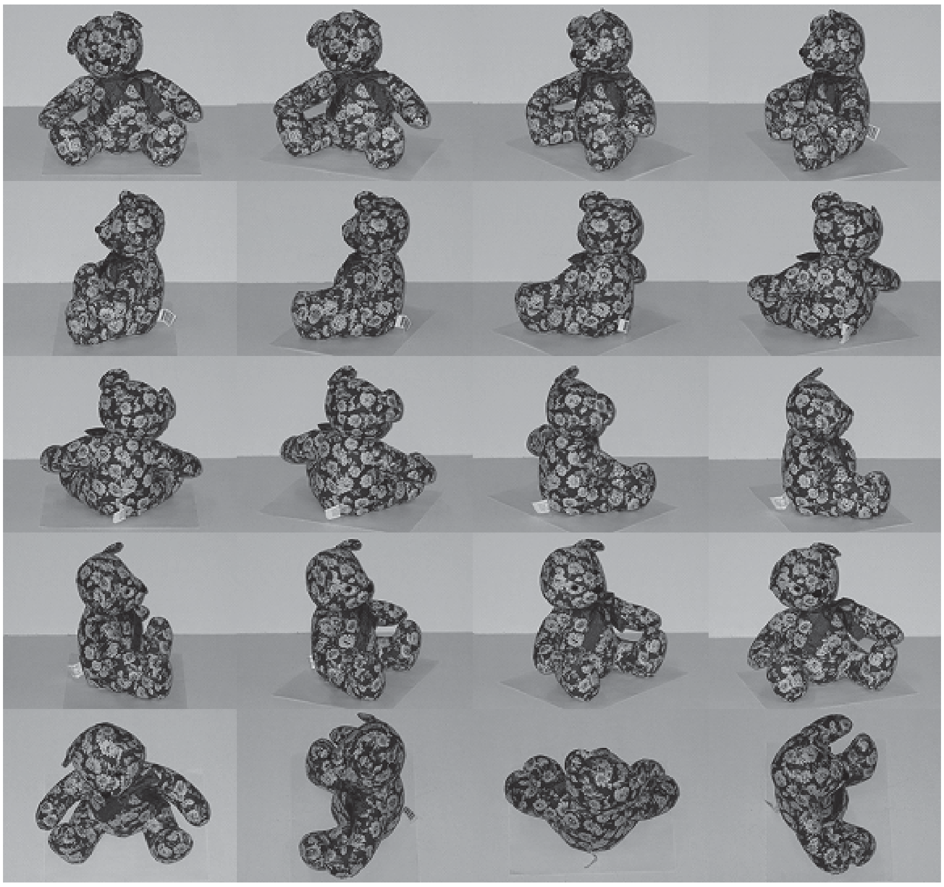

Fig. 4. Example of images belong one class from database [15]

The operation of calculating the remainder of the division by modulo 2^p is just reading of p least significant bits of the number. Calculating the remainder of the division by modulo 2^p-1 is more difficult. Let $X=\overline{X_{g-1}X_{g-2}\dots X_0}$. is an g-bits original number. It can be divided into $s=\lceil g/p\rceil$ parts of p bits width. To this end we complete X from the right to 0 to the dimension $g'=s\cdot p$, now $X'=\overline{X_{g'-1}X_{g'-2}\dots X_0}$. Then $Y_0=\overline{X_{p-1},..,X_1,X_0}$, $Y_1=\overline{X_{2p-1},..,X_{p+1},X_p}$, ..., $Y_s=\overline{X_{g'-1},..,X_{(s-1)p+1},X_{(s-1)p}}$ are the parts of X'. The number X' can be represented as $X'=Y_0+Y_1\cdot 2^p+Y_2\cdot 2^{2p}+\ldots+Y_s\cdot 2^{sp}$. Transformations using number-theoretic properties give the following chain of equalities:

$$\left|X'\right|_{2^p-1}=\left|Y_0+Y_1\cdot 2^p+Y_2\cdot 2^{2p}+\ldots+Y_s\cdot 2^{sp}\right|_{2^p-1}=$$
$$=\left|\left|Y_0\right|_{2^p-1}+\left|Y_1\cdot 2^p\right|_{2^p-1}+\left|Y_2\cdot 2^{2p}\right|_{2^p-1}+\ldots+\left|Y_s\cdot 2^{sp}\right|_{2^p-1}\right|_{2^p-1}=$$
$$=\left|\left|Y_0\right|_{2^p-1}+\left|Y_1\cdot 2^p+Y_1-Y_1\right|_{2^p-1}+\left|Y_2\cdot 2^{2p}+Y_2-Y_2\right|_{2^p-1}+\ldots\right.$$
$$\left.+\left|Y_s\cdot 2^{sp}+Y_s-Y_s\right|_{2^p-1}\right|_{2^p-1}=\left|\left|Y_0\right|_{2^p-1}+\left|Y_1\cdot\left(2^p-1\right)+Y_1\right|_{2^p-1}\right.=$$
$$=\left|Y_2\cdot\left(2^{2p}-1\right)+Y_2\right|_{2^p-1}+\ldots+\left.\left|Y_s\cdot\left(2^{sp}-1\right)+Y_s\right|_{2^p-1}\right|_{2^p-1}=$$
$$=\left|\left|Y_0\right|_{2^p-1}+\left|Y_1\right|_{2^p-1}+\left|Y_2\right|_{2^p-1}+\ldots+\left|Y_s\right|_{2^p-1}\right|_{2^p-1}=\left|Y_0+Y_1+Y_2+\ldots+\right.$$
$\left.+Y_s\right|_{2^p-1}$. In this way we obtain

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

That is, the calculation of the remainder of the division by modulo 2^p-1 is addition of p-bits numbers by modulo 2^p-1. To add by modulo 2^p-1 we use tree of end-around-carry carry-save adders with modulo 2^p-1 Kogge-Stone adder proposed in [18].

B. RNS to Binary Conversion

The most common method to achieve equivalent weighted number from residues is using the CRT [20]. Computing weighted number X form its RNS representation, i.e. (x_1, x_2, …, x_n), based on the moduli set {m_1, m_2, …, m_n} is as follows:

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

where $M_i=M/m_i$ and $\left|M_i^{-1}\right|_{m_i}$ is the multiplicative inversion of M_i modulo m_i for $i=1,2,\ldots,n$. In order to implement

CRT, the remainder of the division by a large number, i.e. *M*, is required, and implementation of this operation in hardware results in increase of area and delay.

The modification of the Chinese remainder theorem using fractional values, namely approximate CRT, introduced for the first time in [21] to perform sign-detection and division in RNS. The effective hardware design of this approach is based on compression technique of summands and Kogge-Stone adder modulo 2^N is proposed in [22]. We used that method to implement the RNS to Binary converter in this article.

C. Convolution in RNS

RNS is most effective when performing calculations that contain only operations of addition and multiplication. This can be seen from formulas (2) and (3). Formula (1) shows that convolution operation in CNN uses only these operations. This means that RNS may be very effective for hardware implementation of CNN convolutional layer. Unfortunately, the difficulty of performing a comparison operation in RNS does not allow to expect its successful application in max pooling layer and multi-layer perceptron neural network parts of CNN. This motivated us to propose an approach to partitioning CNN architecture between hardware and software parts. We propose to use hardware circuit for RNS realization of CNN convolutional layer and to use software calculations in remaining layers of CNN.

The coefficients $W_{i,j,k}$ and bias b from formula (1) in the trained CNN are constants. This means that convolution circuit must implement multiplication by constants with the summation of the results. Since we suggest using modules $\left\{2^{p_1}, 2^{p_2}-1, ..., 2^{p_n}-1\right\}$ in RNS that multiplication by a constant can be implemented very effectively using the technique described in [15]. We use that approach for hardware implementation of CNN convolutional layer.

IV. CNN ARCHITECTURE AND TRAINING

We proposed to use the CNN architecture presented in Fig 5. The input of CNN is an RGB image of size 256×192 , the first two layers are responsible for identifying the features of the image. The first two layers produce convolution operation by 8 filters, the size of filter mask is $[3\times3]\times3$, with stride 3 . The result of calculations of the first layer is 8 feature maps in size 85×64 . The second layer performs 2×2 max pooling operation with stride 2. 8 feature maps in size 42×32 are the outputs of the second layer and connected to the inputs of the last two layers which are responsible for the image classification. The third layer consists of 10 neurons, and fourth one consists of 8 neurons, each of them corresponds to a certain class.

The convolutional operation takes most part of working time in network. To increase speed of work we split up the architecture of CNN on hardware and software parts. The convolutional layer is implemented in hardware on FPGA by using calculations in RNS. Because comparison operation and non-lineal activation function are difficult to implement in RNS so the max pooling layer and the fully connected network are realized in software part.

Neural Pattern Recognition Toolbox performed the CNN training in Matlab R2017b. Calculations were made on PC with CPU Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz, 4.00GHz, memory of RAM volume 16,0 GB and 64-bit operation system Windows 10. 161 images belonging to 8 different classes were used for training [15]. The neural network was trained for 30 iterations during 57 seconds. Fig. 6 shows a graph of the learning process generated by Matlab software. The results of work of CNN are shown in Fig. 7.

An example of filter mask from convolutional layer is shown in Table I. For the hardware implementation, we quantized the values by 12 bits. The obtained filter coefficients are also given in Table I.

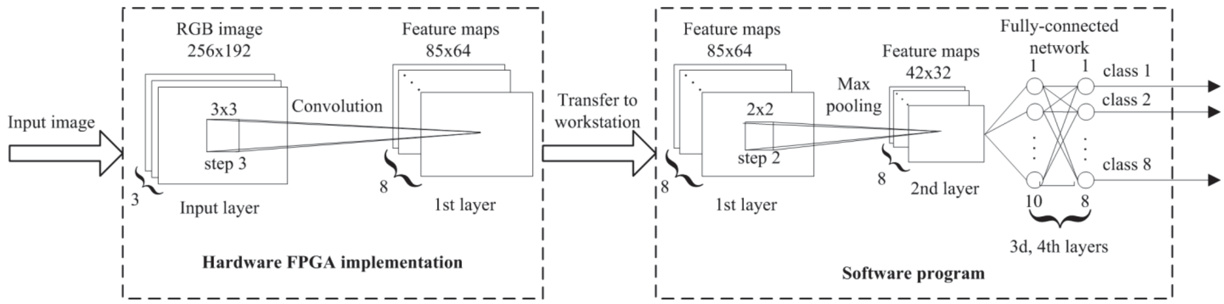

Fig. 5. The proposed CNN architecture

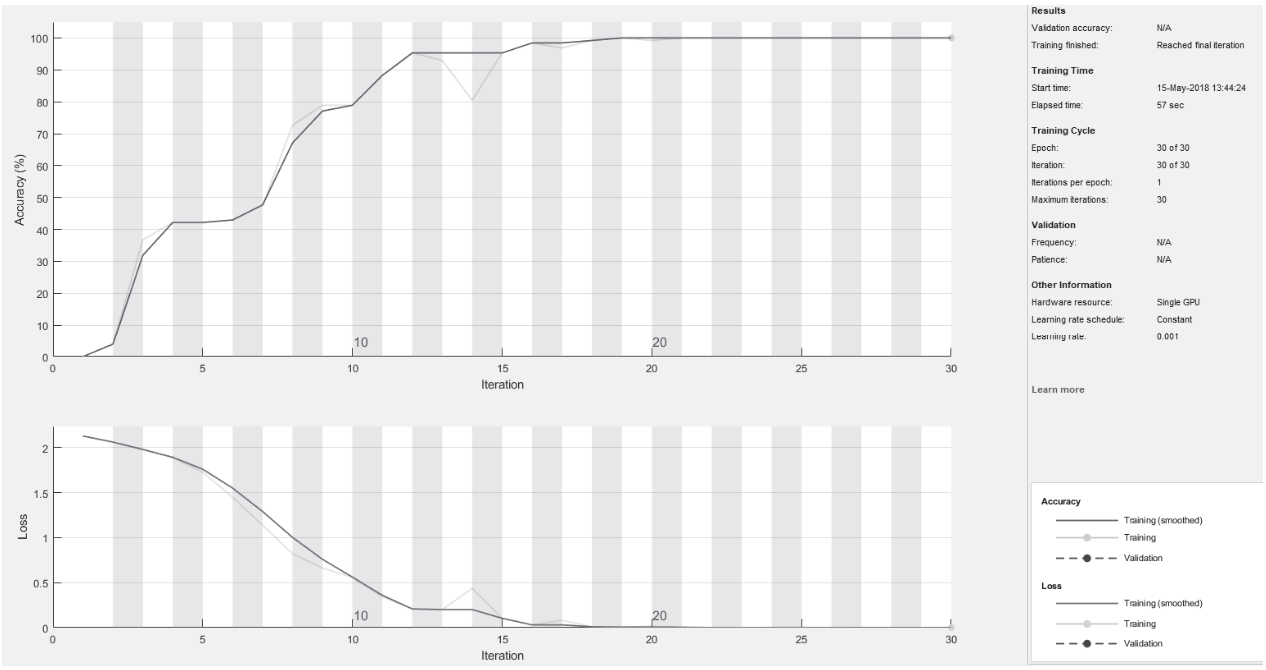

Fig. 6. The CNN training report from Matlab software

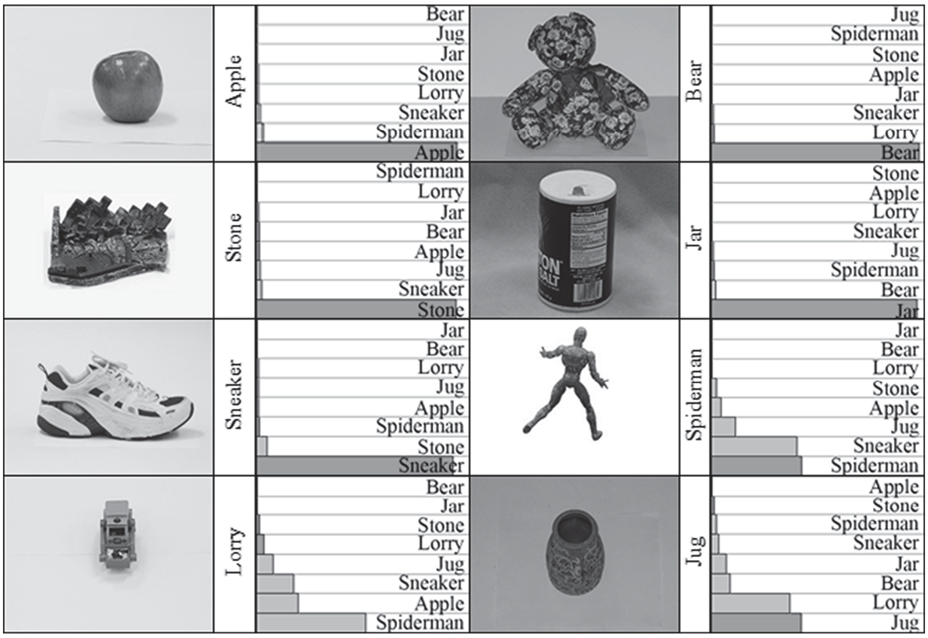

Fig. 7. Results of work of CNN

VI. Simulation results of convolutional layer hardware implementation

Hardware simulation was implemented on FPGA Kintex7 xc7k70tfbg484-2 in Xilinx Vivado 16.3. We used “High Performance Optimized” modeling parameter for simulations. The goal of simulation was comparison the usage of BNS and RNS.

The convolution operation was simulated by different modules of the form 2^p and 2^p-1. The results are presented in Table II and Fig. 8. and shows that circuit delay varies from 8,721 ns to 16,035 ns.

Taking into account the values of the quantized filter coefficients, and the need to represent negative numbers in RNS we obtained the condition $M \geq 278970$ for RNS dynamic range. Use of this condition as well as data from Table 2 allowed us to choose two moduli sets $\left\{2^5-1, 2^6-1, 2^8\right\}$ and $\left\{2^3-1, 2^4-1, 2^5-1, 2^7\right\}$ for simulation of full RNS system containing Binary to RNS converter, RNS convolution and RNS to Binary converter.

TABLE I. AN EXAMPLE OF 3D-FILTER MASK FROM CONVOLUTIONAL LAYER OF TRAINED CNN

Layer	Filter mask	Quantized filter mask
R	$\begin{pmatrix} 0.008708132 & -0.01040934 & 0.00319623 \\ 0.01531038 & -0.01347609 & 0.006832784 \\ 0.02441053 & 0.003114703 & 0.008579108 \end{pmatrix}$	$\begin{pmatrix} 36 & -42 & 14 \\ 63 & -55 & 28 \\ 100 & 13 & 36 \end{pmatrix}$
G	$\begin{pmatrix} -0.008610572 & -0.01240873 & -0.00146828 \\ 0.006109328 & -0.005821416 & -0.0116995 \\ 0.001523695 & 0.01010766 & -0.02120716 \end{pmatrix}$	$\begin{pmatrix} -35 & -50 & -6 \\ 26 & -23 & -47 \\ 7 & 42 & -86 \end{pmatrix}$
B	$\begin{pmatrix} -0.00484508 & 0.0003131653 & 0.003700315 \\ 0.00084957 & -0.01582666 & -0.02015062 \\ -0.00416168 & -0.004977863 & 0.003042456 \end{pmatrix}$	$\begin{pmatrix} -19 & 2 & 16 \\ 4 & -64 & -82 \\ -17 & -20 & 13 \end{pmatrix}$
Bias	-0.000331978	-1

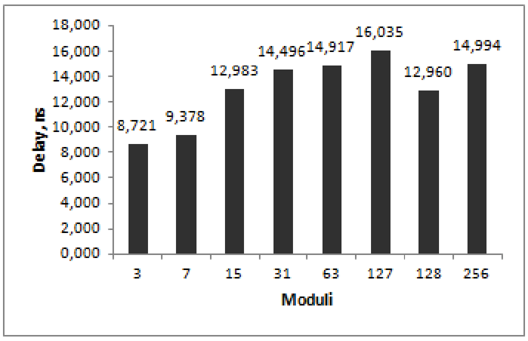

Fig. 8. Delay of convolution operation for different moduli values

TABLE II. DELAY OF CONVOLUTION OPERATION FOR DIFFERENT MODULI

Moduli	Delay, ns
2^2-1	8.721
2^3-1	9.378
2^4-1	12.983
2^5-1	14.496
2^6-1	14.917
2^7-1	16.035
2^7	12.960
2^8	14.994

Taking into account the values of the quantized filter coefficients, and the need to represent negative numbers in RNS we obtained the condition $M \geq 278970$ for RNS dynamic range. Use of this condition as well as data from Table 2 allowed us to choose two moduli sets $\{2^5-1, 2^6-1, 2^8\}$ and $\{2^3-1, 2^4-1, 2^5-1, 2^7\}$ for simulation of full RNS system containing Binary to RNS converter, RNS convolution and RNS to Binary converter.

Simulation results obtained by using BNS and RNS are presented in Fig. 9. Simulation shows that using RNS with moduli set $\{2^5-1, 2^6-1, 2^8\}$ allow to reduce hardware costs by 32% and using moduli set $\{2^3-1, 2^4-1, 2^5-1, 2^7\}$ by 29.5% compare with BNS. This allows us to conclude that the use of RNS for CNN convolutional layer hardware implementation is more effective in area compared to BNS implementation.

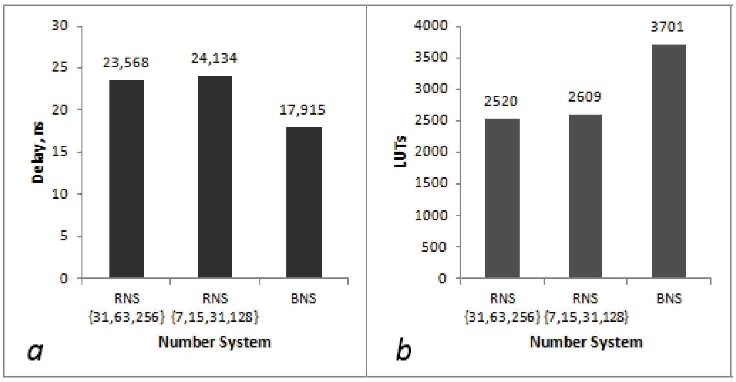

Fig. 9. Simulation results of convolutional layer hardware realization: (a) delay; (b) number of occupied LUTs.

VII. CONCLUSIONS

The paper presents a method of hardware implementation of CNN for pattern recognition using computations in RNS. The minimalistic CNN configuration includes the convolutional layer, the max pooling layer and the recognition classifier, which is organized as the traditional multi-layer perceptron neural network. The hardware simulation of convolution operation showed that using the proposed method based on RNS with special moduli allows to reduce hardware costs by 32% in comparison with BNS implementation. A generalization of this result to cases of large filter masks requires further practical investigations. The research results may be applied in the area-efficient development of video surveillance systems, for recognition of handwriting, faces, objects and location.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio and P. Haffiner "Gradient-based learning applied to document recognition," Proc. of the IEEE, Vol.86. No. 11, 1998, pp.2278-2324.

[2] M. Sankaradas, V. Jakkula, S. Gadami, S. Chakradhar, I. Duranovic, E. Cosatto and H.P. Graf "A massively parallel coprocessor for convolutional neural networks," *20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP2009),* 2009, pp.53-60.

[3] M. Peemen, A.A.A. Setio, B. Mesman and H. "Corporal memory centric accelerator design for convolutional neural networks," *31st International Conference on Computer Design (ICCD2013)*, 2013, pp.13-19.

[4] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello "Hardware accelerated convolutional neural networks for synthetic vision systems," Int'l Symp. on Circuits and Systems (ISCAS2010), 2010, pp.257-260.

[5] H. Nakahara and T. Sasao "A deep convolutional neural network based on nested residue number system," 2015 *25th International Conference on Field Programmable Logic and Applications (FPL)*, London, 2015, pp.1-6.

[6] J. Fan, W. Xu and Y. Wu "Human Tracking Using Convolutional Neural Networks," IEEE Transactions on Neural Networks, VOL. 21, NO. 10, OCTOBER 2010, pp.1610 -1623.

[7] A. Krizhevsky, I. Sutskever, G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in neural information processing systems 25 (2), 2012.

[8] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9.

[9] N. Jouppi, C. Young, N. Patil and D. Patterson, "Motivation for and Evaluation of the First Tensor Processing Unit," in IEEE Micro, vol. 38, no. 3, pp. 10-19, May./Jun. 2018.

[10] E. Köse and M. E. Yalçın, "Emulating CNN with template learning on FPGA," *2017 European Conference on Circuit Theory and Design (ECCTD)*, Catania, 2017, pp. 1-4.

[11] Gan Feng, Zuyi Hu, Song Chen and Feng Wu, "Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks," 2016 *13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT),* Hangzhou, 2016, pp. 624-626.

[12] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii and A. Acquaviva, "A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms," 2017 *New Generation of CAS (NGCAS),* Genova, 2017, pp. 49-52.

[13] T. Manabe, Y. Shibata and 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.

[14] Chervyakov, N.I., Lyakhov, P. A., Valueva, M. V. "Increasing of Convolutional Neural Network Performance Using Residue Number System" International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), 18-22 September 2017, pp. 135-140.

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

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

[17] G.C. Cardarilli, A. Nannarelli and M. Re "Residue number system for low-power DSP applications" // Proc. 41st Asilomar Conf. Signals, Syst., Comput. 2007.pp. 1412 - 1416.

[18] H.T. Vergos, G. Dimitrakopoulos "On Modulo 2^n+1 Adder Design," *IEEE Transactions on Computers*, Vol 61, No.2, 2012, pp 173-186.

[19] P.M. Kogge, H.S. Stone "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations" IEEE Transaction on computers, Vol. C-22, No. 8, 1973, pp 786-793.

[20] Chervyakov N.I., Molahosseini A.S., Lyakhov P.A., Babenko M.G., Deryabin M.A. "Residue-to binary conversion for general moduli sets based on approximate Chinese remainder theorem," International journal of computer mathematics, Vol. 94, No. 9, pp. 1833–1849, 2017.

[21] C.Y. Hung and B. Parhami, "An approximate sign detection method for residue numbers and its application to RNS division," Computers and Mathematics with Applications, vol. 27, no. 4, pp. 23-25, 1994.

[22] R. de Matos, R. Paludo, N. Chervyakov, P. A. Lyakhov and H. Pettenghi, "Efficient implementation of modular multiplication by constants applied to RNS reverse converters," 2017 *IEEE International Symposium on Circuits and Systems (ISCAS),* Baltimore, MD, 2017, pp. 1-4.