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 \( 0 \leq x < R \), \( 0 \leq y < C \) and \( 0 \leq z < D \) 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:

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

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.

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.

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 \( 256 \times 192 \) pixels using the Adobe Photoshop CS6 software by the bicubic interpolation algorithm. 161 images from database were used for CNN training.

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 \( \{m_1, m_2, \ldots, m_n\} \), \( \gcd(m_i, m_j) = 1 \), \( i \neq j \) called modules. The product of all RNS modules \( M = m_1 m_2 \ldots m_n \) 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 \( x_i = X \mod m_i \) in accordance with Chinese Remainder Theorem (CRT) [16].

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

\[
A + B = \left( |a_1 + b_1|_{m_1}, \ldots, |a_n + b_n|_{m_n} \right),
\]

\[
A \cdot B = \left( |a_1 \cdot b_1|_{m_1}, \ldots, |a_n \cdot b_n|_{m_n} \right).
\]

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 \( \{2^n, 2^p - 1, \ldots, 2^p - 1\} \) 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 \( \{2^n, 2^p - 1, \ldots, 2^p - 1\} \). It is necessary to calculate the remainder of the division by each of the moduli to conversion a number into RNS [16].
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 \(X = X_{g-1}X_{g-2}...X_0\) is an \(g\)-bits original number. It can be divided into \(s = \lfloor g / p \rfloor\) parts of \(p\) bits width. To this end we complete \(X\) from the right to 0 to the dimension \(g = s \cdot p\), now \(X' = X_{g-1}X_{g-2}...X_0\). Then \(Y_0 = X_{g-1}, Y_1 = X_{2g-1}, Y_2 = X_{3g-1}, ...\), \(Y_{g-1} = X_{(g-1)p+1}, Y_{g} = X_{(g-1)p+2}, Y_{g+1} = X_{(g-1)p+3}, ...\) are the parts of \(X'\). The number \(X'\) can be represented as \(X' = Y_0 + Y_1 \cdot 2^p + Y_2 \cdot 2^{2p} + ... + Y_{g-1} \cdot 2^{(g-1)p}\). Transformations using number-theoretic properties give the following chain of equalities:

\[
\begin{align*}
|X|_{2^p-1} &= |Y_0 + Y_1 \cdot 2^p + Y_2 \cdot 2^{2p} + ... + Y_{g-1} \cdot 2^{(g-1)p}|_{2^p-1} = \\
&= |Y_0|_{2^p-1} + |Y_1 \cdot 2^p|_{2^p-1} + |Y_2 \cdot 2^{2p}|_{2^p-1} + ... + |Y_{g-1} \cdot 2^{(g-1)p}|_{2^p-1} = \\
&= |Y_0|_{2^p-1} + |Y_1 \cdot 2^p + Y_2 \cdot 2^p + Y_3 \cdot 2^p + Y_4 \cdot 2^p + ... + Y_{g-1} \cdot 2^{(g-1)p}|_{2^p-1} = \\
&= |Y_1 \cdot 2^p + Y_2 \cdot 2^{2p} + Y_3 \cdot 2^{3p} + ... + Y_{g-1} \cdot 2^{(g-1)p}|_{2^p-1} = \\
&= |Y_1 \cdot (2^{2p}-1) + Y_2 \cdot (2^{2p}-1) + ... + Y_{g-1} \cdot (2^{2p}-1)|_{2^p-1}.
\end{align*}
\]

In this way we obtain

\[
|X|_{2^p-1} = |Y_0 + Y_1 \cdot 2^p + Y_2 \cdot 2^{2p} + ... + Y_{g-1} \cdot 2^{(g-1)p}|_{2^p-1} = |Y_0 + Y_1 + Y_2 + ... + Y_{g-1}|_{2^p-1}.
\]

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:

\[
X = \left\lfloor \sum_{i=1}^{n} \left[ \frac{M_i}{m_i} \right]_{M_i} M_i x_i \right\rfloor_M
\]

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 \(i = 1, 2, ..., n\). 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 $\left\{2^N, 2^N - 1, \ldots, 2^N - 1\right\}$ 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 $256 \times 192$, 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 $[3 \times 3] \times 3$, with stride 3. The result of calculations of the first layer is 8 feature maps in size $85 \times 64$. The second layer performs $2 \times 2$ max pooling operation with stride 2. 8 feature maps in size $42 \times 32$ 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-linear 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.
VI. SIMULATION RESULTS OF CONVOLUTIONAL LAYER HARDWARE IMPLEMENTATION

Hardware simulation was implemented on FPGA Kintex7 xc7k70tg484-2 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 $\{2^5-1, 2^3-1, 2^2\}$ and $\{2^3-1, 2^5-1, 2^2-1, 2^1\}$ 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

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter mask</th>
<th>Quantized filter mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0.008708132 -0.01040934 0.00319623</td>
<td>36 -42 14</td>
</tr>
<tr>
<td></td>
<td>0.01531038 -0.01347609 0.006832784</td>
<td>63 -55 28</td>
</tr>
<tr>
<td></td>
<td>0.02441053 0.003114703 0.008579108</td>
<td>100 13 36</td>
</tr>
<tr>
<td>G</td>
<td>-0.008610572 -0.01240873 -0.00146828</td>
<td>-35 -50 -6</td>
</tr>
<tr>
<td></td>
<td>0.006109328 -0.005821416 -0.0116995</td>
<td>26 -23 -47</td>
</tr>
<tr>
<td></td>
<td>0.001525695 0.01010766 -0.02120716</td>
<td>7 42 -86</td>
</tr>
<tr>
<td>B</td>
<td>-0.00484508 0.0003131653 0.003700315</td>
<td>-19 2 16</td>
</tr>
<tr>
<td></td>
<td>0.00084957 -0.01582666 -0.02015062</td>
<td>4 -64 -82</td>
</tr>
<tr>
<td></td>
<td>-0.00416168 -0.004977863 0.003042456</td>
<td>-17 -20 13</td>
</tr>
<tr>
<td>Bias</td>
<td>-0.000331978</td>
<td>-1</td>
</tr>
</tbody>
</table>

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

TABLE II. DELAY OF CONVOLUTION OPERATION FOR DIFFERENT MODULI

<table>
<thead>
<tr>
<th>Moduli</th>
<th>Delay, ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>8.721</td>
</tr>
<tr>
<td>31</td>
<td>9.378</td>
</tr>
<tr>
<td>41</td>
<td>12.983</td>
</tr>
<tr>
<td>51</td>
<td>14.496</td>
</tr>
<tr>
<td>61</td>
<td>14.917</td>
</tr>
<tr>
<td>71</td>
<td>16.035</td>
</tr>
<tr>
<td>28</td>
<td>12.960</td>
</tr>
<tr>
<td>29</td>
<td>14.994</td>
</tr>
</tbody>
</table>

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 $\{2^3 - 1, 2^3 - 1, 2^3\}$ and $\{2^3 - 1, 2^3 - 1, 2^3\}$ 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 $\{2^3 - 1, 2^3 - 1, 2^3\}$ allow to reduce hardware costs by 32% and using moduli set $\{2^3 - 1, 2^3 - 1, 2^3\}$ 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


