Hardware Implementation of Convolutional Neural Networks Based on Residue Number System

Roman Soloviev, Dmitry Telpukhov, Ilya Mkrtchan, Alexander Kustov, Alexander Stempkovskiy
Institute for Design Problems in Microelectronics of Russian Academy of Sciences (IPPM RAS)
Moscow 124365, Russian Federation

***Abstract*—The paper examines the use of residue number system (RNS) for hardware implementation of neural networks based on VLSI or FPGA. Widely known mobile neural networks that are highly accurate and best suited for implementation at hardware level have been explored. Major difficulties in their RNS implementationare examined. Several methods for solving the related problems are proposed: convolutions with step value greater than one, rather than previously used MaxPooling layers, are considered; non-standard activation functions containing only addition, subtraction and multiplication are investigated; efficient algorithm for scaling implementation is proposed and compared with conventional algorithm. Finally, we propose complete flow for design and transfer of MobileNet neural network to hardware level on RNS basis.**

Keywords — neural networks (NN), residue number system (RNS), scaling operation, hardware implementation.

I. Introduction

Neural networks (NN) cope well with a variety of tasks related to classification and processing of images, audio and video data; in some cases they do even better than humans [1]. Most modern NN architectures incorporate convolutional blocks (for example, Inception [3], ResNet [4], U-Net [5], Mask-RCNN [2]), for this reason, such networks are called convolutional neural nets (CNN). Computational complexity during classification is so large that even powerful processors cope poorly with calculations. This problem is especially acute for real time processing of video data.

Some structures of neural networks with very high accuracy of image classification have properties that enable easy transfer to hardware platform. There is a direction related to design of neural networks for mobile devices, such as MobileNets [7, 30, 31], SqueezeNet [8], MnasNet [29]. They have small number of weights and relatively small number of arithmetic operations. However, they are implemented at software level and use floating point calculations. Unfortunately, in some cases, for example, in real-time video processing, it is not always possible to provide continuous processing of video stream at 30 frames per second without significant optimization, even if mobile networks are used.

To use neural networks at the stage when we have a trained model in a real device, we can apply a set of optimizations to speed up calculations. A number of techniques for such optimization already exist, for example, compression of weights [9] or calculations on data with less precision [10].

Increasing requirements to hardware that deals with neural networks stimulate design of special hardware accelerators for use in VLSI and FPGA. Acceleration of calculations can be achieved due to:

- hardwareparallel implementation of convolution unit, that is faster than convolution performed at software level;

- transition from floating point to fixed point calculations;

- smaller dimension of calculations with acceptable accuracy;

- neural network reduction, while keeping classification accuracy;

- modification of neural network structure with slight decrease in accuracy (or even keeping the same accuracy) while performance grows and the size of hardware implementation and stored weights is reduced.

As current studies have shown, the use of residue number system in computing with neural networks can lead to additional speed up of calculations and reduction of power consumption.

Researchers from Japan in [11] have proposed a novel implementation of deep CNN using nested residue number system, analogous to recursive RNS [26]. They have suggested replacing massive 48-bit MAC unit with a large set of 4-bit MAC units implemented as FPGA lookup tables. Due to reduction of residues dimensions to 4 bits, they managed to significantly increase clock frequency of the device — from 100-200 to 400 MHz, and significantly increase the performance of the device as a whole.

Researchers from Italy in paper [12] have proposed an improved MAC-Unit based on residue number system and a moduli set$\{2^{l_1}, 2^{l_2}-1, \dots, 2^{l_N}-1\}$. Slightly reducing the accuracy of the network due to rounding and intermediate calculations, they managed to increase operation speed by 1.5-2.4 times and reduce energy consumption by 3-9 times. At the same time, classification accuracy of AlexNet neural network fell from 79% to 75% only.

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

Researchers from Russia [13] presented architecture of a convolutional neural network based on RNS calculations with basis of special type, due to which they increased performance by 37% compared to binary implementation.

These works demonstrate significant potential of residue number system in the design of hardware neural networks. In this study, several methods are proposed that are suitable for implementing neural networks in small specialized microelectronic devices, for which the use of GPU is unprofitable due to significant energy consumption and high cost.

II. Residue number system for neural net calculations Advantages and disadvantages

The typical structure of a convolutional neural network for the classification of images is shown in Fig. 1. Later, some improved structures with branches [3] [4] appeared, but the essence remained the same: the size of the image from layer to layer decreases, and the number of filters increases. At the end of the convolution network, a set of features is formed, which is fed to the classification layer (or layers) and the output neurons signal the likelihood that the image belongs to a particular class.

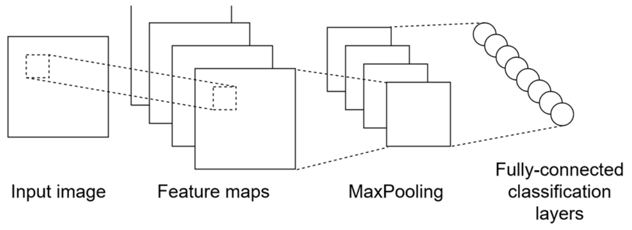

Fig. 1. Typical structure of convolutional neural network

To speed up computations, we usually switch from computing at software level to hardware implementation, as well as from floating point to fixed point calculations [14] [15]. In this case, significant prospects open up for using residue number system [26] [27] [28] due to some of its features:

1) residue number system is known to do poor job with floating point calculations; on the other side, for fixed point, where dynamic range of calculations is known in advance, a suitable RNS basis can be selected and applied;

2) weights of a trained neural network are known in advance, therefore they can be directly written and stored in residual form using known RNSmoduli set.Therefore, hardware implementation of forward converter for weights, size of which is very large, is not required;

3) main computing units of a neural network are: convolutional and fully connected layers, which require 90 percent (or more) of computing power [16].These types of layers contain addition and multiplication operations only. Exactly at these operations, residue number system shows its advantages over positional [32];

4) input data arrives to neural network in the form of a color image of a small size. Then, a large set of arithmetic operations is performed as the data moves from layer to layer over the neural network. Here, in case of classification, the result is a small set of output numbers. This means that forward and reverse converters make up just a very small fraction of the total amount of calculations;

However, there are several problems when using certain types of layers, which can lead to performance degradation when using RNS:

1) in neural networks, MaxPooling or AvgPooling blocks are used to reduce the size of intermediate feature maps. Major problems for these blocks in terms of RNS are as follows. For the first block, Max operation is used, which is a special case of comparing numbers – a complex operation in RNS that requires a large distinct block. In the second case, calculation of average is required, which entails dividing by 4, which can also cause difficulties. However, this problem can be solved by using convolution block with strides parameter of 2 instead of MaxPooling, which is already widely used in modern networks, for example, MobileNet. In case of using convolutional block instead of MaxPooling, only addition and multiplication operations remain;

2) BatchNormalization layers contain division, however, after constants are reduced, only one addition and one multiplication operations remain. Moreover, if BatchNormalization layer goes immediately after the convolutional block, and this is how it is located in most modern neural networks, it can be removed due to convolutional layer weights recalculation;

3) activation layers of neurons are a major problem: such layers appear after almost all convolutional layers and bring in non-linearity. The most popular activation types in modern networks are:

Rectified linear unit (RELU) – due to its simplicity in terms of implementation in positional arithmetic; and Sigmoid or Softmax, which are commonly used in output layers of networks;

$$RELU \to f(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases} \qquad Sigmoid \to f(x) = \frac{1}{1+e^{-x}}$$

RELU – is extremely simple for positional arithmetic, it becomes difficult in RNS because of comparison with zero (sign detection [17]). Sigmoid is computationally complex in both cases because of the presence of function$e^{-x}$and division.

One of the possible solutions to the problem is developing activation function using only operations that are "friendly" for RNS, or creating efficient hardware implementation for one of the existing activations. RELU is one of such activations, for whichefficient algorithms can be used [18]. However, approach implicating development of special activation functions for residue number system is more effective.

III. Custom activation functions

In this paper, activation functions that contain only addition and multiplication operations have been studied. It have been shown experimentally that activation function $f(x) = x^2 - kx$ converges well for some values of k. A MobileNet network was

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

built in which all RELU layers were replaced with the proposed function. It was shown that for the task of classifying cars from the Open Images Dataset [24] neural network was trained and showed classification accuracy of up to 93% (see Fig. 2). The accuracy on validation data is slightly higher than on the training set due to the fact that very strong augmentations were applied to the input data.

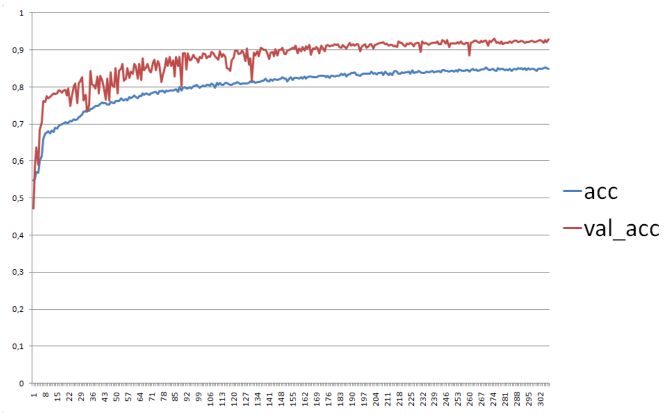

Fig. 2. MobileNet training process with activations x^2-50x. Horizontal axis shows era number, vertical axis shows classification accuracy. The blue line is the accuracy for training data, the red line is the accuracy for validation data.

The proposed activation function allows us to get rid of comparison operations in neural network and thereby eliminate the bottleneck for hardware implementation based on residue number system. Disadvantages of this approach are related to complexity of training, inability to use pre-trained neural networks and lower accuracy of classification.

IV. Neural network design flow based on fixed point calculations

Consider the sequence of actions required to prepare hardware implementation of a neural network using a specific example. Let us take MobileNet (v1) [7] as a neural network, which is well-known and showed good results in practice. It has large number of layers (about 100), does not contain branches, and has few weights, so that internal memory of even small FPGA chips is enough to store them. Classification accuracy reaches 70.6% for ImageNet set. Modification we used, implied detection if the image belonged to single specific class.

In total, MobileNet includes 8 types of layers:

1) ZeroPadding2D – adds zeros around the image. The layer guarantees that the size of feature map at convolution output match the size of input feature map;

2) Conv2D (kernel = 3x3) – two-dimensional convolution with 3x3 kernel;

3) Conv2D (kernel = 1x1) – two-dimensional convolution with 1x1 kernel;

4) DepthwiseConv2D – simplified version of Conv2D (fewer additions, multiplications and, accordingly, fewer weight matrices);

5) Activation (Relu6) – activation function, improved RELU function – additional upper bound is imposed. All values greater than 6 are equated to 6. So potential maximum values are not allowed to go far from 0, which is very useful for representing numbers in fixed point mode;

6) GlobalAvgPooling – this layer is used in neural network only once; it is the next-to-last layer. It has no weights, and it finds average values of feature maps of the previous layer. This layer generates a vector of image features (vector length is usually from 512 to 4096);

7) Dense (Fully Connected) – the last fully connected network layer, performs image classification. In case of single class classification, it consists of two neurons. The first one signals that the object is absent, the second – that the object is present. At the output of this layer, Softmax activation is used, which returns probabilities of presence of the expected objects;

8) BatchNormalization – layer performing normalization that comes fitted with each convolutional layer.

Step 1. Work begins with the preparation of a training data set. For simplicity, we assume that we need to determine whether the object in the picture belongs to a certain class, which may be useful for security camera or mobile phones.

Step 2. At this step, input image size for neural network should be selected, on which its configuration will depend. Typical image sizes suitable for MobileNetare: 128x128, 160x160, 192x192 and 224x224 pixels.

Step 3. Then we train the network on our data set, using any suitable framework: tensorflow, pytorch, caffe, keras, etc. The goal is to get weights for the network, which are a set of floating point numbers.

Step 4. Bundles of Conv2D + BatchNormalization and DepthwiseConv2D + BatchNormalization layers can be merged into one Conv2D and one DepthwiseConv2D layers, recalculating weights using formulas of BatchNorm Fusion method. At this point, MobileNet will decrease from 100 layers to 70 layers. It will work faster, and the values at the network output will be the same as for the full version.

Note 1: after layer reduction, the network is no longer suitable for training; more precisely, training will be much less effective than on a full network.

Note 2: in the initial version of the network, layers Conv2D and DepthwiseConv2D did not have bias, it will appear after reduction.

Step 5. Value 6 of the RELU(6) activation function can be replaced by 1, then the values of the attribute maps after each activation will be normalized to the interval from 0 to 1, just

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

like the input image. This operation is rather straightforward. It is enough to divide all the weights of the first layer by 6, then divide bias at each layer by 6 and replace all instances of RELU(6) with RELU(1). Position of the maximum on the last layer will not change, that is, classification will be performed exactly as by the initial model.

Step 6. At this stage, we managed to normalize the input image and all the values of the intermediate feature maps for interval from 0 to 1; however, generally, weights and biases can exceed these values. Therefore, we need to find maximum and minimum to determine how many bits of top digits we need to store additionally. For this, formulas can be used:

$$ConvW = \lceil \log_2 \max(weights) \rceil,$$

$$ConvB = \lceil \log_2 \max(bias) \rceil$$

where *weights* – all weight values at all layers, *bias* – all bias values at all layers, *ConvW* – how many additional bits we need to store each weight, *ConvB*– how many additional bits we need to store each bias.

If the network is trained on a large dataset and has good classification accuracy, then ConvW and ConvB should not be very large compared to the dimension of feature maps and input images. For our test network the values were ConvW = 7 and ConvB = 3.

Step 7. Further, in order to switch to fixed-point calculations, it is necessary to determine N – how many bits are enough to keep calculations accurate, that is, the classifier would give the same result in floating-point and fixed-point calculations. We can somewhat complicate the task and find optimal dimensions separately for the feature maps, weights and biases (Fig. 3).

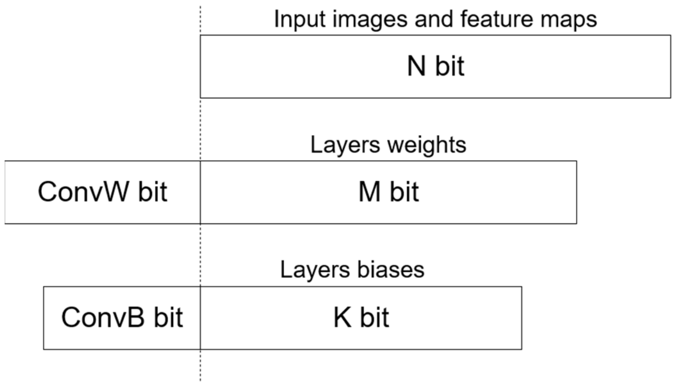

Fig. 3. Different dimensions of weights, biases and feature maps

To do this, we can use the following type of algorithm. We set the variable N to some large value, on which the network will certainly work accurately. We pass all validation images through the network and compare classification results with the results for floating point mathematical model. If the accuracy is higher than required, we reduce N until we find N for which the accuracy is lower than the specified one. Then we try to separately reduce dimension of weights M and biases K, until we find the optimum. This operation is known as quantization with calibration [25].

For MobileNet test network from this study, values of N, M, and K may slightly vary for different datasets. Values N∈ [10,12], M∈ [9,11], K∈ [8,10] (excluding sign of numbers). More detailed description of transition from floating point to fixed point in neural networks is given in papers [35] [36].

Step 8. As we plan to perform scaling operation after each convolution block, we have to find maximum value of intermediate calculations achievable for the given parameters. Scaling strategy may vary. For example, we can perform scaling after each elementary operation of convolution block: 9 multiplications and 9 additions (including bias). But since scaling operations in residue number system are computationally expensive, it is better to perform them after calculating feature map for the next layer or even once for several layers. Therefore, we have to find maximum number of additions and multiplications for the entire network that will be performed before the application of scaling operation. If scaling is performed after each individual convolutional layer, then the dimension of the dynamic range D can be calculated by the formula:

$$D = \lceil \log_2 S \rceil + N + (ConvW + M)$$

where*S* is the required number of additions. Scaling after each convolutional layer should be done by 2^M bits, which is similar to shifting *M* positions to the left for a number in positional notation.

For computing with RNS, the major challenge is the scaling operation coming after all addition and multiplication operations in convolution block. If we try to avoid this operation, then dimension of the numbers and the required dynamic range will increase greatly from layer to layer and a large set of RNS moduli will be required, which will make the use of residue number system less efficient. However, for scaling operations in RNS, special fast methods were developed [19-22].

Step 9. Depending on the calculated dynamic range, RNS moduli basisis selected.

Step 10. After choosing a moduli set for representing numbers, it is necessary to convert all weights and biases first into binary and fixed-point records, then into RNS form based on the selected moduli set; and after doing so, save them to RAM of the device.

Step 11. After reduction, MobileNetis structured as a sequence of “convolution” + “activation” blocks. That is, in RNS form, we need a set of operations of modular channel consisting of additions and multiplications only. It is a good practice to first perform RELU(1) activation, that is, do comparison with 0 and 1 to reduce dimensions in scaling operation. Comparison and scaling operations can be performed in parallel with modular calculations as soon as the results of the first pixel calculations for feature maps are ready. In other words, to calculate results for the entire network, a set of three consecutive blocks of operations is required:

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

● massive set of fast modular calculations on a convolutional block, including only parallel multiplications and additions of small dimension;

● operation of comparison with 0 and 1;

● operation of scaling by 2^M.

The described flow for positional number system was implemented at software level. The code is available on github [33], and work demonstration can be seen on youtube [34].

V. Fixed point scaling method in residue number system

Two methods from publications [23] and [20] were taken as the basis of scaling algorithm for hardware implementation of neural network. In the first paper, a method for implementing operation of expanding modular basis (base extension) based on lookup tables (LUT) was proposed. In the second paper, a method for scaling to an arbitrary number K is proposed, where the number K, which is called the scaling factor, is co-prime with all modules of the selected base. The method is also based on LUT tables. Due to the use of tables and small arithmetic block, the speed of scaling operation increases significantly.

Consider in more detail the method of expanding the modular basis. Let modular basis of mutually prime modules $\{p_1, p_2, \dots p_n\}$and number$X = \{x_1, x_2, \dots, x_n\}$within the margins of this base be given. Applying the Chinese remainder theorem, we can obtain:

$$X = \left|\sum_{i=1}^{n} M_i \left|\frac{x_i}{M_i}\right|_{p_i}\right|_M = \sum_{i=1}^{n} M_i \left|\frac{x_i}{M_i}\right|_{p_i} - eM$$

where$e \in [0; n)$and$M_i = \frac{M}{p_i}$.

In paper [23] it is proved that for any modulep_{n+1}value $|X|_{p_{n+1}} = x_{n+1}$can be calculated by formula:

$$x_{n+1} = |X_E - eM|_{p_{n+1}} =$$

$$\left| M^{(p)} \left|\sum_{i=1}^{n} a_{i,p}\right|_{\frac{M}{M^{(p)}}} + \sum_{i=1}^{p} \frac{M^{(p)}}{p_i} \left| \frac{M}{M^{(p)}} \left|\frac{x_i}{M_i}\right|_{p_i} \right|_{p_i} - eM \right|_{p_{n+1}} \quad (1)$$

In this formula: $M^{(p)} = \prod_{j=1}^{p} p_j$, $M_i = \frac{M}{p_i}$, $a_{i,p} = \left\lfloor \frac{M_i \left|\frac{x_i}{M_i}\right|_{p_i}}{M^{(p)}} \right\rfloor$and$e = \{0,1\}$.

Also, in this formula,$p$can take any value in the range from 1 to n: $1 \le p < n$. Value of $e$is 1only if the following conditions are met:

$$\begin{cases} \left|\sum_{i=1}^{n} a_{i,p}\right|_{\frac{M}{M^{(p)}}} \ge \frac{M}{M^{(p)}} - p + 1 \\ |X_E|_M \le (p-1)M^{(p)} - M^{(p)} \sum_{i=1}^{p} \frac{1}{p_i} \end{cases} \quad (2)$$

Values $a_{i,p}$can be pre-calculated for a given base and then obtained from tables. Values$a_{i,p}$are in the interval: $0 \le a_{i,p} \le \frac{M}{M^{(p)}} - \frac{M_i}{M^{(p)}}$. The same apply to values$b_i = \frac{M^{(p)}}{p_i} \left| \frac{M}{M^{(p)}} \left|\frac{x_i}{M_i}\right|_{p_i} \right|_{p_i}$, which can be pre-calculated. The most difficult task is to find conditions under which $e = 1$.

If we set $T = \left|\sum_{i=1}^{n} a_{i,p}\right|_{\frac{M}{M^{(p)}}}$formula (1) can be re-written as:

$$x_{n+1} = \left| M^{(p)} T + \sum_{i=1}^{p} b_i - eM \right|_{p_{n+1}}$$
$$= \left| \left|M^{(p)} T\right|_{p_{n+1}} + \left|\sum_{i=1}^{p} b_i\right|_{p_{n+1}} - e|M|_{p_{n+1}} \right|_{p_{n+1}},$$

and condition (2) as:

$$\begin{cases} T \ge \frac{M}{M^{(p)}} - p + 1 \\ \left| M^{(p)} T + \sum_{i=1}^{p} b_i \right|_M \le (p-1)M^{(p)} - M^{(p)} \sum_{i=1}^{p} \frac{1}{p_i} \end{cases}$$

The proposed method allows us to quickly perform the base extension. Now consider a method for scaling based on the extension of modular basis proposed in [20]. That is, we should find such a value of$Y$that$Y = \left\lfloor \frac{X}{K} \right\rfloor$. This formula can be re-written as follows: $Y = \left\lfloor \frac{X - |X|_K}{K} \right\rfloor$or in modular form:

$$y_i = |Y|_{p_i} = \left| \frac{X - |X|_K}{K} \right|_{p_i} = \left| |x_i - |X|_K|_{p_i} \cdot |K^{-1}|_{p_i} \right|_{p_i}$$

If we choose $K$to be co-prime to all base modules, there necessarily exists$|K^{-1}|_{p_i}$This is well suited for our method, since we can choose$K = 2^L$if all base modules are prime numbers greater than 2. If $K$is already fixed, then values$|K^{-1}|_{p_i}$can be pre-calculated and saved to tables. For calculation, only modular subtraction and multiplication by a constant will be required, for which efficient implementations are well-known.

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

VI. Experimental results forscaling in residue number system

To verify the operation of the proposed quick method of scaling numbers, a series of experiments was carried out. In the experiments, the proposed method was compared with the conventional method based on reverse transformation of numbers from modular representation, scaling operation in positional number system, and then forward conversion from positional number system to residue number system.

Experiment 1: In the first experiment, we checked how device performance and area depend on the size of the base and the chosen modules. Checked bases: {3, 5, 7, 11, 13}, {3, 5, 7, 11, 13, 17}, {3, 5, 7, 11, 13, 17, 23}, {3, 5, 7, 11, 13, 17, 23, 29}, {3, 5, 7, 11, 13, 17, 23, 29}, {3, 5, 7, 11, 13, 17, 23, 29, 31}, {3, 5, 7, 11, 13, 17, 23, 29, 31, 37}, {3, 5, 7, 11, 13, 17, 23, 29, 31, 37, 41}. The value for scaling K was chosen equal to 2^8=256. The results are shown in Fig. 4.

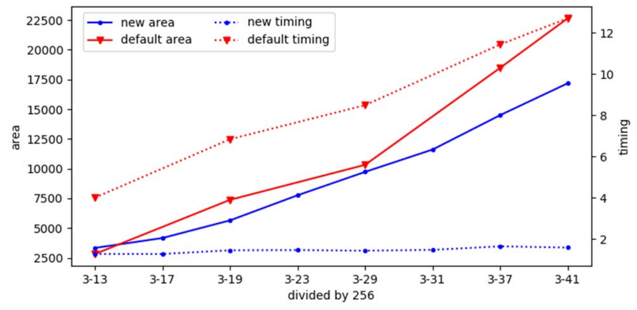

Fig. 4. Signal delay and device area for the first experiment

As can be seen from the graphs, performance of the proposed method is almost independent of the base size and is several times faster than for the conventional scaling method. In addition, area for the same bases is also slightly reduced.

Experiment 2: In the second experiment, we checked how device performance and area depend on the value of the scaling factor *K* for a fixed base {3, 5, 7, 11, 13, 17, 23}. The results are shown in Fig. 5.

As can be seen from the graphs, the proposed method is sensitive to the value of K: the larger it is, the greater the device area. Presumably, this is because of the use of lookup tables. Device performance is 3-4 times higher than for the classical method, which is significant improvement. Delay increases slowly with the growth of scaling factor.

Experiment 3: In the final experiment, we checked how device performance and area depend on the dimension of modules making up the base. We run tests for three different modular bases: {3, 5, 7, 11, 13, 17, 23, 29}, {71, 73, 79, 83, 89} and {233, 239, 241, 251}; and three values of K: 256, 512 and 1024. The results are shown in Fig. 6 and 7.

As can be seen from the graphs, delay of scaling devices built by the new method is almost independent of the base size and dimension of the modules. As to area, it is the smaller, the smaller are the modules dimensions.

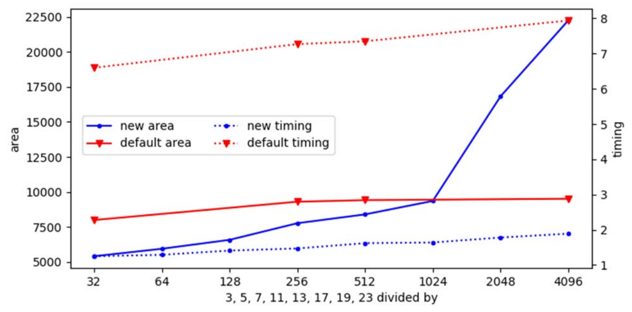

Fig. 5. Signal delay and device area for base {3, 5, 7, 11, 13, 17, 23} and different values of scaling factor

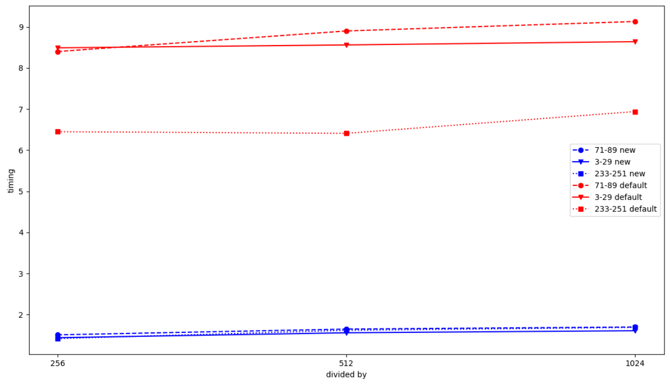

Fig. 6. Signal delayfor three bases of different sizes and different values of scaling factor

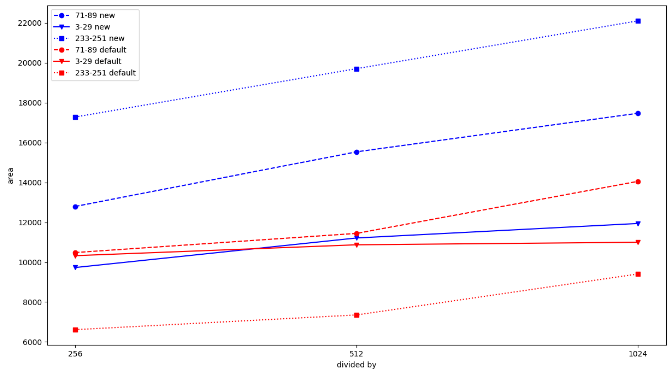

Fig. 7. Device area for three bases of different sizes and different values of scaling factor

VII. Conclusion

In this paper, we have proposed a method for design of hardware implementation of neural networks based on residue number system. Implementation flow is proposed for the well-known mobile neural network MobileNet. The bottlenecks of this implementation related to operations of comparison and scaling in residue number system are marked out. Efficient scaling algorithm is proposed for implementation in hardware

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.

module, as well as comparison-free activation function. As experiments have shown, the new scaling method in residue number system is much faster than the conventional method, and comparable with it in area. The experiment results allow us to draw conclusions about which set of modules is best suited for the problems under consideration. The new scaling method allows solving the bottleneck problem of residue number system as applied to hardware implementation of neural networks.

REFERENCES

[1] Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556, 2014.

[2] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A. Going deeper with convolutions, Cvpr, 2015.

[4] He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770-778.

[5] Ronneberger O., Fischer P., Brox T. U-net: Convolutional networks for biomedical image segmentation, International Conference on Medical image computing and computer-assisted intervention, Springer, Cham, 2015, pp. 234-241.

[6] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Kudlur, M. TensorFlow: A System for Large-Scale Machine Learning, OSDI, 2016, vol. 16, pp. 265-283.

[7] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861, 2017.

[8] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size, arXiv preprint arXiv:1602.07360, 2016.

[9] Han S., Mao H., Dally W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149, 2015.

[10] Jouppi N. Google supercharges machine learning tasks with TPU custom chip, Google Blog, May, 2016, vol. 18.

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

[12] Arrigoni V., Rossi B., Fragneto P., Desoli G. Approximate operations in Convolutional Neural Networks with RNS data representation, European Symposium on Artificial Neural Networks (ESANN 2017).

[13] Chervyakov N.I., Lyakhov P.A., Kalita D.I., Valueva M.V. Convolutional neural network architecture using computations in residue number system with specific module set, Neirocomputery: Razrabotka, Primenenie, 2017, no. 1, pp. 3-15 (in Russian).

[14] Solovyev R., Kustov A., Telpukhov D., Rukhlov V., Kalinin A. Fixed-Point Convolutional Neural Network for Real-Time Video Processing in FPGA. In 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), IEEE, 2019, January, pp. 1605-1611).

[15] Solovyev R. A., Kalinin A. A., Kustov A. G., Telpukhov D. V., Ruhlov V. S. FPGA implementation of convolutional neural networks with fixed-point calculations, 2018, arXiv preprint arXiv:1808.09945.

[16] Cong J., Xiao B. Minimizing computation in convolutional neural networks, International conference on artificial neural networks, Springer, Cham, 2014, pp. 281-290

[17] Brönnimann H., Emiris I., Pan V. Y., Pion S. Sign determination in residue number systems, Theoretical Computer Science, 1999, vol.210, pp. 173-197.

[18] Van Vu T. Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding, IEEE Transactions on Computers, 1985, vol.100, no.7, pp. 646-651.

[19] Hitz M. A., Kaltofen E. Integer division in residue number systems, IEEE transactions on computers, 1995, vol. 44, no. 8, pp. 983-989.

[20] Kong Y., Phillips B. Fast scaling in the residue number system, IEEE transactions on very large scale integration (VLSI) systems, 2009, vol. 17, no. 3, pp. 443-447.

[21] Low J. Y. S., Chang C. H. A VLSI Efficient Programmable Power-of-Two Scaler for {2^{n}-1, 2^{n}, 2^{n}+ 1} RNS, IEEE Transactions on Circuits and Systems I: Regular Papers, 2012, vol. 59, no.. 12, pp. 2911-2919.

[22] Smyk R., Czyżak M., Ulman Z. Scaling of signed residue numbers with mixed-radix conversion in FPGA with extended scaling factor selection, Computer Applications in Electrical Engineering, 2013, vol. 11.

[23] Barsi F., Pinotti M. C. Fast base extension and precise scaling in RNS for look-up table implementations, IEEE transactions on signal processing,1995, vol. 43, no. 10, pp. 2427-2430.

[24] Kuznetsova A., Rom H., Alldrin N., Uijlings J., Krasin I., Pont-Tuset J., Ferrari V. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, 2018, arXiv preprint arXiv:1811.00982.

[25] Goncharenko A., Denisov A., Alyamkin S., Terentev E. Fast adjustable threshold for uniform neural network quantization, 2018, arXiv preprint arXiv:1812.07872.

[26] Stempkovsky A.L., Amerbaev V. M., Solovyev R.A.. Principles of Recursive Modular Arithmetic, Informacionnye tekhnologii, 2013, no.2, pp.22-27 (in Russian).

[27] Amerbaev V.M., Solovyev R.A., Telpukhov D.V. Library implementation of modular arithmetic operations based on logic functions minimization algorithms, Izvestiya Yughnogo Federalnogo Universiteta. Technicheskie Nauki, 2013, vol.7, no.144 (in Russian).

[28] Amerbaev V.M., Solovyev R.A., Telpukhov D.V., Balaka E.S. Construction of residue number system reverse converters with error correction, based on mixed-number system, Neirocomputery: Razrabotka, Primenenie, 2014, no.9, pp.30-35 (in Russian).

[29] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2820-2828).

[30] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).

[31] Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Le, Q. V. (2019). Searching for mobilenetv3. arXiv preprint arXiv:1905.02244.

[32] Omondi A. R., Premkumar B. Residue number systems: theory and implementation. – World Scientific, 2007. – T. 2.

[33] https://github.com/ZFTurbo/MobileNet-in-FPGA

[34] https://www.youtube.com/watch?v=EQ9MJnWeHlo

[35] Solovyev, R., Kustov, A., Telpukhov, D., Rukhlov, V., & Kalinin, A. (2019, January). Fixed-Point Convolutional Neural Network for Real-Time Video Processing in FPGA. In 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus) (pp. 1605-1611). IEEE.

[36] Low precision inference on GPU, Nvidia. URL: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf

Authorized licensed use limited to: University of Cincinnati. Downloaded on February 03,2021 at 14:42:27 UTC from IEEE Xplore. Restrictions apply.