Fast and Gate-Efficient Approximated Activations for Bipolar Morphological Neural Networks*

E. E. Limonova[I, II]

[I] Federal Research Center “Computer Science and Control” of Russian Academy of Sciences, Moscow, Russia
[II] Smart Engines Service LLC, Moscow, Russia

Abstract. Bipolar morphological neural networks are aimed at efficient hardware implementation without multiplications inside the convolutional layers. However, they use resource-demanding activation functions based on binary logarithm and exponent. In this paper, the computationally efficient approximations for activation functions of bipolar morphological neural networks are considered. Mitchell's approximation is used for binary logarithm and demonstrates 12 times decrease in the estimated logic gate number and latency. Schraudolph's approximation used for exponent has 3 times lower logic gates complexity and latency. The usage of approximate activation functions provides a 12-40% latency decrease for the BM convolutional layers with a small number of input channels and 3x3 filters compared to standard ones. The experiments show that these approximations can be used in the BM ResNet trained for classification task with a reasonable recognition accuracy decreasing from 99.08% to 98.90%.

Keywords: bipolar morphological networks, approximations, computational efficiency.

DOI 10.14357/20718632220201

Introduction

Artificial neural networks are a part of modern recognition technologies and are widely used in practical tasks [1-4]. More and more end devices with mobile processors, ASICs, or FPGAs become target devices for the inference of neural networks. Neural network models for efficient execution on such devices are actively developed. The main research directions are the development of quantized networks that operate low-bit integer coefficients [5, 6] and the development of models with alternative operations within the layers or individual neurons of the network [7-11].

An example of such a model is a bipolar morphological (BM) network [10, 11]. The main computations in such networks fall on the addition and maximum operations. These operations require fewer logic gates for implementation than the multiplication, so BM networks are potentially more energy-efficient and fast than classical ones. However, the original BM networks include complex activation functions based on the logarithm and exponent operations. In this paper, these functions are approximated to reduce their computational complexity. The resulting operation number and gate complexity are estimated. The accuracy of the BM networks with considered approximations is evaluated experimentally.

1. Bipolar Morphological Networks

Let us define *exp2* and *log2* operations as:

$$exp2(x) = 2^x$$
$$log2(x) = \log_2 x$$

* This work is partially supported by Russian Foundation for Basic Research (project 19-29-09066)

Then one BM neuron can be defined as follows:

$$y_{BM}(x, v^+, v^-, v_0) = = \varphi\big(exp2\, max_{j=1}^{R}\big(log2\, x_j^+ + v_j^+\big) - exp2\, max_{j=1}^{R}\big(log2\, x_j^+ + v_j^-\big) - exp2\, max_{j=1}^{R}\big(log2\, x_j^- + v_j^+\big) + exp2\, max_{j=1}^{R}\big(log2\, x_j^- + v_j^-\big) + v_0\big),$$

$$x_j^+ = \begin{cases} x_j, & x_j \geq 0 \\ 0, & x_j < 0 \end{cases}',$$

$$x_j^- = \begin{cases} -x_j, & x_j < 0 \\ 0, & x_j \geq 0 \end{cases}',$$

where x is an input vector of length R, v^+, v^- are weight vectors of size R, v_0 is bias, $\varphi(\cdot)$ is a nonlinear activation function. It is assumed that $log2(0) = -\infty$, which is replaced by a big enough negative value for actual computations.

BM convolutional layer, which uses BM neurons instead of classical ones, can be defined as:

$$J = \varphi\Big(\sum_{\alpha}\sum_{\beta} p^\alpha p^\beta \exp2\big(\log2\, I^\alpha \odot v^\beta\big) + v_0\Big),$$

where $\alpha \in -, +$, $\beta \in -, +$, $p^+ = +1$, $p^- = -1$, $\odot$ is a BM convolution operation:

$$(\mathrm{I} \odot v)_{\mathrm{n,m,f}} = \max_{\mathrm{c=1}}^{\mathrm{C}} \max_{\Delta n=0}^{\mathrm{K-1}} \max_{\Delta m=0}^{\mathrm{K-1}} \big(\mathrm{I}_{\mathrm{n+\Delta \mathit{n},\, m+\Delta \mathit{m},\, c}} + \mathrm{v}_{\Delta n,\, \Delta m,\, \mathrm{c,\, f}}\big),$$
$$f = \overline{1, F}, n = \overline{1, N}, m = \overline{1, M},$$

where F is the number of filters, C is the number of input channels, KxK is the spatial dimensions of the filter, input image size is NxMxC, v is a set of convolutional filters, v_0 is the bias. It is assumed that I is padded properly for the result to be of the same size.

The operations of *log2* and *exp2* process activation vectors only, which means they are less computationally intensive than the BM convolution. However, they still require time and hardware to be performed.

2. Floating-Point Arithmetic

Computations in classical neural network models are performed in floating-point numbers. The most widely used data formats for representing such values in computers are defined by IEEE 754 floating-point arithmetic standard. This standard was developed by the Institute of Electrical and Electronics Engineers (IEEE) in 1985 and then updated in 2008 and 2019. The latest version of the standard includes [12]:

1. Binary and decimal floating-point data formats.
2. Definitions of arithmetic operations: addition, subtraction, multiplication, division, fused multiply-add, square root, compare, and other operations.
3. Rules for converting between integer and floating-point formats.
4. Rules for converting between different floating-point formats.
5. Rules for converting floating-point data to external representations (for example, strings).
6. Formats and rules for handling floating-point exceptions, including the handling of non-numeric data.

This standard ensures the same calculation results for software, hardware, or combined implementations of floating-point arithmetic, and provides a unified error format that is not limited to a specific implementation.

Floating-point data in binary formats of IEEE 754 consists of 3 ordered fields:

a) 1-bit sign S.

b) w-bit biased exponent $E = e + bias$

c) $(t = p - 1)$-bit trailing significand field digit string $T = d_1 d_2 ... d_{p-1}$; the leading bit of the significand, d_0, is implicitly encoded in the biased exponent E.

The value v of the floating-point datum is obtained from these fields as follows:

a) If $E = 2^w - 1$ and $T \neq 0$, then v is NaN and d_1 allows one to distinguish between quiet NaN and signaling NaN.

b) If $E = 2^w - 1$ and $T = 0$, then $v = (-1)^S \cdot (+\infty)$.

c) If $1 \leq E \leq 2^w - 2$, then the number is considered normal and $v = (-1)^S \cdot 2^{E-bias} \cdot (1 + 2^{1-p} \cdot T)$.

d) If $E = 0$ and $T \neq 0$, then the number is considered subnormal and $v = (-1)^S \cdot 2^{emin} \cdot (0 + 2^{1-p} \cdot T)$, where *emin* is a minimal value of e.

e) If $E = 0$ and $T = 0$, then $v = (-1)^S \cdot (+0)$.

Machine learning tasks most commonly use floating-point format *binary32* with binary encoding and 32-bit width. It uses $w = 8$, $p = 24$, $bias = 127$, $emin = -127$.

Note that the input of a neural network usually consists of normal floating-point values. If the input data and weight coefficients are correctly set, all the computations inside the network results are finite and non-NaN values. Subnormal values can appear as a result of multiplication or subtraction. However, these are very small numbers around zero, which can be assumed to be zero for a neural network. Therefore, in this paper only normal real values are considered.

3. Binary Logarithm Approximation

In computing systems, the binary logarithm is calculated via different approximations. The complexity of the approximation depends on the desired accuracy. Let us use a floating-point representation of the real numbers to decompose the *log2* calculation:

$$log2(x) = \log_2\left(2^e(1+y)\right) = e + \log_2(1+y),$$

where *e* is an integer, and *y* is a real value in the [0, 1) range. Now the problem of binary logarithm approximation is reduced to the approximation of *log2(1 + y)*.

3.1. Polynomial Approximation

In [13] a polynomial approximation of the 5th order was introduced to use in BM networks:

$$f(y) = \log_2(1+y) \to g(y) = \sum_{i=0}^{5} C_i y^i,$$

where *y* is from [0, 1). Then *f(y)* and *g(y)* were equated at three equally spaced points as well as the values of *f'(y)* and *g'(y)*. The resulting coefficients were *C* = {0, 1.44269504, -0.71249131, 0.42046732, -0.1955884, 0.04491735} with a maximum approximation error of about $7 \cdot 10^{-5}$ in [0, 1) range (estimated numerically). The approximation is shown in Fig. 1. It takes 5 multiplications and 6 additions (including the one to get e, which can actually require fewer computations depending on hardware implementation) for calculation using Horner's method, and bit manipulations to get the sign, significand, and exponent, which are free for hardware.

3.2. Mitchell's Approximation

An extremely most low-cost approximation of the binary logarithm is the one proposed by J. N. Mitchell [14]. He simply uses the first term of the Maclaurin series to approximate

$$\log_2(1+y) \approx y$$

Then the

$$\log_2(x) = e + y,$$

where e and y are obtained from the exponent and significand of *x* correspondingly. Bit manipulations to obtain e and y are considered to be free for hardware. So, Mitchell's binary logarithm requires only one addition for computation and thus is extremely efficient. At the same time, it has quite a large error of about 0.08639 in the range [0, 1) which was found analytically. However, neural networks allow us to approximate the computations in a wide range and the experimental section of the paper shows the performance of Mitchell's approximation for BM networks.

The approximation is shown in Fig. 1. It can be seen that it is piecewise linear with the end of each interval at the integer powers of two.

3.3. Approximation Efficiency

In order to estimate the efficiency of the considered approximations the following metrics were computed:

a) The number of elementary mathematical operations required for computations. In this case, those are addition and multiplication.

b) The number of logic gates required for hardware implementations of the *log2* module. To estimate it, the numbers of gates for 16 nm addition and multiplication modules from [11] were used. Those values were obtained by describing modules on Verilog HDL and using Synopsys Design Compiler to implement them at gate-level and get logic gate complexity and latency characteristics. The latency of the resulting module was estimated as an accumulated latency for all underlying modules and will be less in actual implementation.

The properties of considered approximations are summarized in Table 1. One can see that the Mitchell's logarithm requires 6 times fewer additions and no multiplications compared to the polynomial approximation. It has about 12 times fewer logic gates and 12.7 times lower latency.

4. Exponent Approximations

4.1. Basic Exponent Approximation

Modern Intel's processors include a special instruction for fast exponent computation [15]. It is

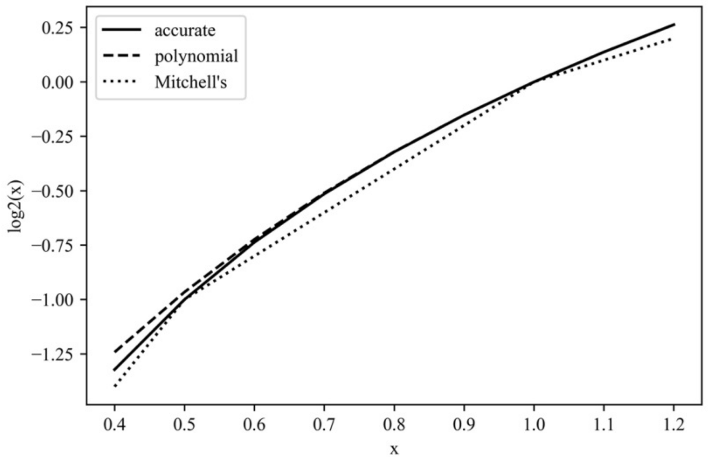

Fig. 1. The comparison of binary logarithm approximations

Table 1. The comparison of logarithm and exponent approximations

Module (16 nm)	add	Mul	gates	latency	accuracy in (0, 1]
addition	1	0	2659	3	-
multiplication	0	1	3247	4	-
polynomial *log2*	6	5	32189	38	$7 \cdot 10^{-5}$
Mitchell's *log2*	1	0	2659	3	0.08639
basic *exp2*	3	3	17718	21	-
Schraudolph's *exp2*	1	1	5906	7	0.05798

based on a look-up table and the polynomial approximation of the 2nd order. Its relative error is less than 2^{-23}, which means that it provides a precise result for *binary32* data. This approximation requires 3 addition and multiplication operations.

4.2. Schraudolph's Approximation

In 1999 N. N. Schraudolph proposed an extremely fast approximation of exponential function, which was based on the structure of IEEE 754 format for floating-point numbers [16]. He considered *binary64* or *double* data, but his approach can be directly expanded to *binary32* data.

Since $v = (-1)^S \cdot 2^{E-bias} \cdot (1 + 2^{1-p} \cdot T)$, to calculate 2^x with an integer x one needs to write $x+bias$ to the exponent field of the value:

$$i = 2^{p-1}(x + bias),$$

where i is an integer representation of the result. Using non-integer x in such an operation modifies the highest bits of significand. This modification provides a linear interpolation between adjacent integer exponents. So, Schraudolph's approximation for *binary32* data is:

$$i = ax + (b - c),$$

where i is an integer representation of the resulting value, $a = 2^{23}$, $b = 127 \cdot 2^{23}$, and c is an adjustment parameter, which was set to 486411 in this work.

So, this approximation uses one integer addition and one floating-point multiplication. Its accuracy in the [0, 1) range is 0.05798 and was estimated numerically. The approximation is shown in Fig. 2.

4.3 Approximation Efficiency

In order to estimate the efficiency of the considered approximations, we used the same metrics as in Section 3.3 The properties of considered approximations are summarized in Table 1. One can see that the Schraudolph's exponent requires 3 times fewer additions and multiplications compared to accurate approximation and has 3 times fewer logic gates and latency.

5. BM Layer Complexity with Approximate Activations

The BM neurons are mostly used in convolutional layers of neural networks. So, let us estimate the number of logic gates and clock cycle latency for a convolutional BM layer with approximate

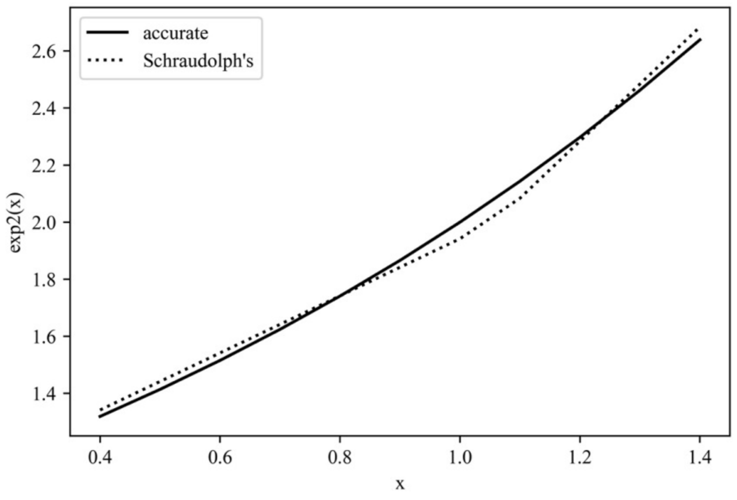

Fig. 2. The comparison of accurate and approximated exp2 functions

activations and compare them to the ones for a standard convolutional layer. The layer's gate number is accumulated for all modules used to perform necessary computations to obtain the total gate number. The latency is accumulated for all modules of one pathway (for positive input and weights) because pathways can be computed in parallel [13]. The 2-pathway model of the BM layer corresponding to a preceding ReLU activation is evaluated. The result is shown in Table 2.

Table 2. The assessment of the gate number and latency ratios for standard, BM, and BM with fast activation convolutional layers
F is a channel number, *C* is a number of channels, *K*x*K* is a kernel size

F	*C*	*K*	Gates, (std/BM)	Latency, (std/BM)	Gates, (std/BM approx)	Latency, (std/BM approx)
16	1	1	0.11	0.22	0.21	0.43
16	16	1	0.52	0.78	0.74	1.19
32	1	1	0.11	0.22	0.21	0.43
32	32	1	0.68	1.00	0.82	1.29
64	1	1	0.11	0.23	0.21	0.44
64	64	1	0.77	1.17	0.87	1.34
128	1	1	0.11	0.23	0.21	0.44
128	128	1	0.84	1.27	0.89	1.37
256	1	1	0.12	0.23	0.21	0.44
256	256	1	0.88	1.33	0.90	1.38
512	1	1	0.12	0.23	0.21	0.44
512	512	1	0.90	1.37	0.91	1.39
16	1	3	0.51	0.87	0.67	1.12
16	16	3	0.85	1.29	0.89	1.37
32	1	3	0.51	0.88	0.67	1.12
32	32	3	0.88	1.34	0.90	1.39
64	1	3	0.51	0.89	0.67	1.12
64	64	3	0.90	1.37	0.91	1.39
128	1	3	0.52	0.90	0.67	1.12
128	128	3	0.91	1.38	0.91	1.40
256	1	3	0.52	0.90	0.67	1.12
256	256	3	0.91	1.39	0.92	1.40
512	1	3	0.52	0.90	0.67	1.12

One can see that the gate efficiency of the BM convolutional layer decreases slightly because activation functions have much less contribution than the operations inside the layer. The latency for convolutions with a big number of input channels and filters is almost not affected for the same reason. However, the assessment shows that fast activations fix the potential slowdown for a small input channel number with 3x3 and bigger filters. For 3x3 kernels, the BM convolutional layer demonstrates 12-40% lower latency than the standard one.

6. Experimental Evaluation

The approximations considered above provide fast inference and a low number of logic gates but alter the results of computations. In order to test their performance, an image classification problem of MNIST dataset was considered. MNIST demonstrates 70000 gray handwritten digit images of size 28x28 pixels, 60000 images for the training set, and 10000 for the test set [17].

The BM network of ResNet-22 architecture was trained according to [13]. Such an architecture contains 22 BM convolutional layers (Fig. 3). Then 1) each BM convolutional layer from the first to the last was converted to use custom *log2* and *exp2*; 2) converted layer was trained to fine-tune the classification quality. To use error backpropagation, gradient of Mitchell's logarithm was implemented. Since the Mitchell's logarithm is continuous, it was simply differentiated to obtain a piecewise-constant gradient. Schraudolph's exponent uses bit representations and is non-differentiable, so the gradient of non-approximate function was used. The accuracy after layer-by-layer conversion and after fine-tuning is shown in Fig. 4.

This experiment demonstrates how weights of the layers can adapt to approximate activations and restore the previous precision, and therefore characterized the expressive power of the BM layer with custom *log2* and *exp2*.

It can be seen that classification accuracy on the testing set decreased from 99.08% to 98.90% in total. The resulting error rate increased from 0.92% to 1.10%, which is a 1.2-time increase. However, the 17th layer still preserved the original accuracy, and the accuracy drop happened between 17 and

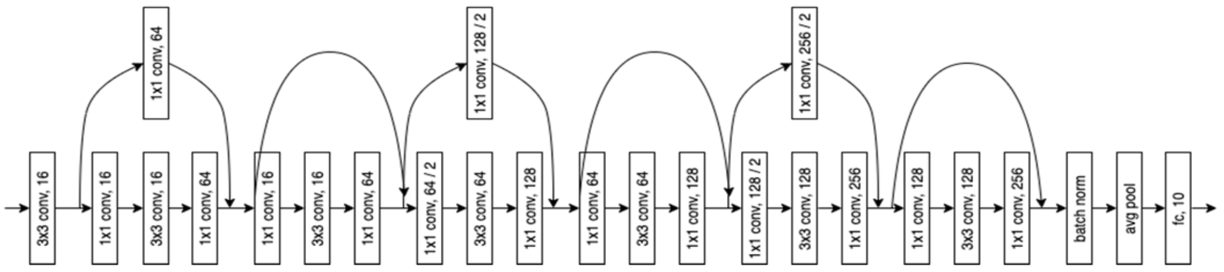

Fig. 3. The ResNet architecture with 22 convolutional layers used in experiments. Batch normalization and activations are omitted for simplicity

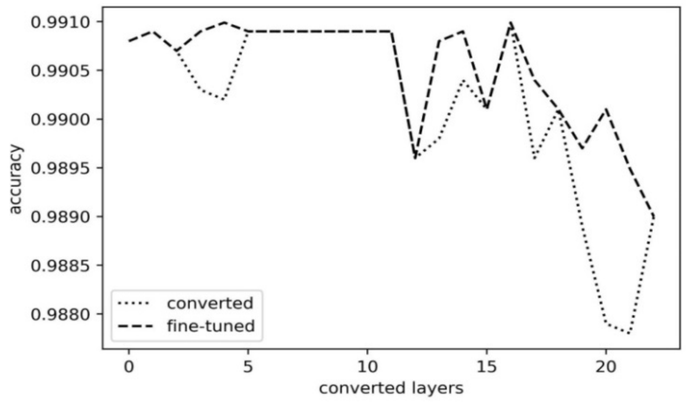

Fig. 4. MNIST classification accuracy for BM network with approximate activations after conversion and after fine-tuning

22 layers. Moreover, this decrease is not steady and depends on the layer number. It means that it depends on the weight values and layer sizes and training from scratch with approximate activations is reasonable.

Conclusion

This paper considers activation approximations for convolutional layers of BM networks. BM activations use binary logarithm and 2-based exponent operations. In this work, Mitchell's logarithm and Schraudolph's exponent approximations were considered. They are extremely fast and allow efficient implementations in hardware. An approximate logarithm requires 6 times fewer additions and no multiplications compared to the polynomial approximation used in previous works. It has about 12 times fewer logic gates and almost 13 times lower latency. The exponent was approximated with Schraudolph's method and used 3 times fewer additions and multiplications compared to accurate approximation and has 3 times fewer logic gates and latency as well. The usage of approximate activations noticeably decreases the latency of the BM layer for a small number of input channels. For 3x3 kernels, the BM convolutional layer demonstrates 12-40% lower latency than the standard one.

Experimental evaluation showed that these approximations can be used in a real BM network. Although the recognition accuracy dropped from 99.08% to 98.90%, it is still high enough for practical usage considering the efficiency increase.

References

1. Chernyshova Y. S., Sheshkus A. V., Arlazarov V. V. Two-step CNN framework for text line recognition in camera-captured images // IEEE Access. — 2020. — vol. 8. — pp. 32587-32600. — DOI: 10.1109/ACCESS.2020.2974051.
2. Kanaeva I. A., Ivanova Y. A., Spitsyn V. G. Deep convolutional generative adversarial network-based synthesis of datasets for road pavement distress segmentation // Computer Optics. — 2021. — vol. 45. — №. 6 — pp. 907-916.
3. Lobanov M.G., Sholomov D. L. Application of shared backbone DNNs in ADAS perception systems // In Thirteenth International Conference on Machine Vision. — vol. 11605. — 2021. — p. 1160525.
4. E. I. Andreeva, V. V. Arlazarov, A. V. Gayer, E. P. Dorokhov, A. V. Sheshkus and O. A. Slavin, "Document Recognition Method Based on Convolutional Neural Network Invariant to 180 Degree Rotation Angle," ITiVS, no 4, pp. 87-93, 2019, DOI: 10.14357/20718632190408.
5. Jacob B., Kligys S., Chen B., Zhu M., Tang M., Howard A.G., Adam H., Kalenichenko D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference // 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. — 2018. — pp.2704-2713.
6. Liu J., Zhuang B., Chen P., Tan M., Shen C. AQD: Towards Accurate Quantized Object Detection // 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), — 2021. — pp. 104-113.
7. Prazeres M., Li X., Oberman A., Nia V.. EuclidNets: Combining Hardware and Architecture Design for Efficient Training and Inference. // In Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods - ICPRAM, — 2022. — ISBN 978-989-758-549-4 — pp 141-151. — DOI: 10.5220/0010988500003122
8. You H., Chen X., Zhang Y., Li C., Li S., Liu Z., Wang Z., Lin Y. ShiftAddNet: A Hardware-Inspired Deep Network. ArXiv, abs/2010.12785. // Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020. — 2020.
9. Chen H., Wang Y., Xu C., Shi B., Xu C., Tian Q., Xu C. AdderNet: Do We Really Need Multiplications in Deep Learning? // 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). — 2020. — pp.1465-1474.
10. Limonova E., Matveev D., Nikolaev D., Arlazarov V. V. Bipolar morphological neural networks: convolution without multiplication // ICMV 2019. — 2020. — T. 11433. — 11433 3J. — C. 1-8. — DOI: 10.1117/12.2559299.
11. Limonova E. E., Alfonso D. M., Nikolaev D. P., Arlazarov V. V. Bipolar Morphological Neural Networks: Gate-Efficient Architecture for Computer Vision // IEEE Access. — 2021. — T. 9. — C. 97569-97581. — DOI: 10.1109/ACCESS.2021.3094484.
12. IEEE Standard for Floating-Point Arithmetic — 2019. — pp.1-84. — DOI: 10.1109/IEEESTD.2019.8766229.
13. Limonova E. E., Alfonso D. M., Nikolaev D. P., Arlazarov V. V. ResNet-like Architecture with Low Hardware Requirements // Processings of ICPR 2020. — May 2021. — ISSN 1051-4651. — ISBN 978-17-28188-09-6. — C. 6204-6211. — DOI: 10.1109/ICPR48806.2021.9413186.
14. Mitchell J.N. Computer Multiplication and Division Using Binary Logarithms // IRE Trans. Electron. Comput. — 1962. — Vol. 11. — pp. 512-517.
15. Reference Implementations for Intel Architecture Approximation Instructions VRCP14, VRSQRT14, VRCP28, VRSQRT28, and VEXP2, https://www.intel.com/content/www/us/en/developer/articles/code-sample/reference-implementations-for-ia-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2.html (accessed April 5, 2022)
16. Schraudolph N. N. A fast, compact approximation of the exponential function // Neural Computation. — 1999. — vol. 11. — no. 4. — pp. 853-862.
17. The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/ (accessed April 5, 2022).

Limonova Elena Evgenievna. Federal Research Center “Computer Science and Control” of Russian Academy of Sciences, Moscow, Russia, researcher; LLC “Smart Engines Service”, programmer. Master’s degree of Moscow Institute of Physics and Technology, 2017. Number of publications: 37. Research interests: neural networks, image processing, pattern recognition on mobile devices. E-mail: limonova@smartengines.com