E. E. Limonova, M. I. Neiman-zade, V. L. Arlazarov, Особенности реализации матричных операций в малобитных нейросетевых моделях на платформе Эльбрус, *Вестн. ЮУрГУ. Сер. Матем. моделирование и программирование*, 2020, том 13, выпуск 1, 118–128

DOI: 10.14529/mmp200109

MSC 68T10 DOI: 10.14529/mmp200109

SPECIAL ASPECTS OF MATRIX OPERATION IMPLEMENTATIONS FOR LOW-PRECISION NEURAL NETWORK MODEL ON THE ELBRUS PLATFORM

***E.E. Limonova*[1,2], *M.I. Neiman-zade*[3], *V.L. Arlazarov*[1]**
[1]Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russian Federation
[2]Smart Engines Service LLC, Moscow, Russian Federation
[3]JSC "MCST", Moscow, Russian Federation
E-mails: elena.e.limonova@gmail.com, muradnz@mcst.ru, vladimir.arlazarov@gmail.com

This paper investigates the possibility of effective implementation of calculations in low-precision neural network models on the Elbrus platform with the VLIW architecture. Such models are widely used in practice to increase the computational efficiency of recognition and well suit computers with the x86 and ARM architectures. In this paper, we consider an 8-bit neural network model, in which matrix multiplication is the most resource-intensive part of the implementation. This paper presents an effective implementation of matrix multiplication that takes into account the features of the Elbrus architecture: the presence of several computational channels with various arithmetic and logic devices, an array prefetch buffer, and its own SIMD extension. We carry out theoretical and experimental comparisons of the computational efficiency of low-precision and classical neural network models, which show that Elbrus processors have much more capabilities for performing fast floating point calculations and require the development of new approaches to increase the computational efficiency of neural network models.

Keywords: low-precision neural networks; computational efficiency; Elbrus architecture; matrix operations.

Introduction

Artificial neural networks are widely used for solving pattern recognition problems. Many modern recognition systems use neural network models, for example, when recognizing documents or license plates [1–5]. Improving the accuracy of the models often requires complication of their structure and leads to a decrease in computational efficiency. At the same time, the scope of recognition systems is steadily growing and the requirements for their computational efficiency are increasing, therefore the problem of improving the inference speed of neural network models is extremely urgent for mobile or embedded devices and solutions that use complex high quality neural network models [6–8].

One of the latest approaches to the development of highly efficient neural network models is the quantization of weights and intermediate calculation results with a moderate decrease in recognition accuracy. Such methods allow to use small-sized integer types for the calculations and increase the computational performance on a number of x86, x86_64 and ARM processors. For example, the gemmlowp [9] and qnnpack [10] software packages provide efficient implementation of matrix operations on 8-bit integers for various processor architectures. This operation takes the majority of computational time for the most neural networks. Further studies show that it is possible to use 4-bit models without significant recognition accuracy loss in a number of cases [8, 11] and speed-up recognition inference even more.

The research in the area of low-precision (with 1-8 bit coefficients) neural networks is actively conducted and it seems to be a promising area of machine learning. However, the main goal of these studies is to create a computationally efficient neural network model with an accuracy comparable to that of the original single precision model. The majority of the studies are implicitly based on some assumptions about the computing device architecture. For example, operations on integer data are considered to be more productive than operations on floating-point numbers. Therefore, it must be taken into account that the effectiveness of such methods strongly depends on the architecture of the device. In this paper, we consider processors with very long instruction word (VLIW) architecture Elbrus [12], that have special features and are well optimized for floating-point computations. We have to design low-precision matrix multiplication, because existing optimized packages do not suit the Elbrus architecture. In this paper we create and investigate the efficient implementation of matrix operations for a 8-bit neural network model for Elbrus processors and compare the computational efficiency with optimized floating-point implementation.

1. Matrix Operations in Artificial Neural Networks

Modern models of artificial neural networks vary considerably: from the relatively simple AlexNet [13, 14] to variations of the Inception architecture [15] and deep recurrent neural networks [16, 17]. However, despite noticeable differences, modern neural network architectures massively use convolutional layers of various sizes and spent most of the time into convolution calculation.

Let us inspect a convolutional layer of a neural network. Each convolutional layer performs a convolution of an input image with a set of filters, adds bias and applies a nonlinear activation function. The input of the layer can be a multichannel image, such as a color image. Filters can also be multichannel, because they contain coefficients for each input channel. The output of convolutional layer can be described by the following expression:

$$O(x, y) = \sum_{c} \sum_{\Delta x} \sum_{\Delta y} I(c, x + \Delta x, y + \Delta y) w(c, \Delta x, \Delta y), \tag{1}$$

where (x, y) is the point of the output, O is the output of convolution, c is the channel number, I is the input image, w is the matrix described the filter (see Fig. 1).

After the convolution, the bias b is added and the non-linear activation function φ is applied:

$$O'(x, y) = \varphi(O(x, y) + b), \tag{2}$$

where O' is the final output of the convolutional layer and φ is an activation function, such as a rectifier (ReLU), hyperbolic tangent, etc.

Normally the architecture of a neural network does not involve complex activation functions, and the main computational complexity is connected with the convolution of the input image with a set of filters.

In [18] a computationally efficient implementation of the convolution is presented as a single operation of matrix-matrix multiplication. The authors present convolutional filters in the form of a matrix in which each row corresponds to one filter, all of its coefficients are placed sequentially. Then a matrix is formed from the input image. In the matrix one

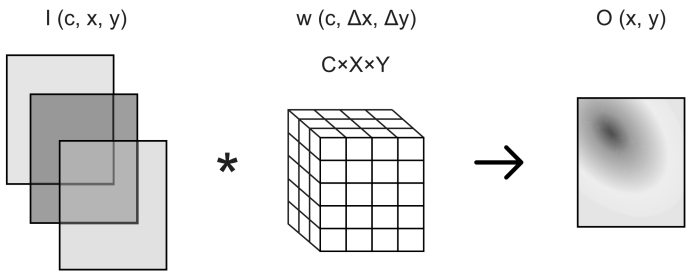

Fig. 1. Convolution of a multichannel image with one $C \times X \times Y$ filter

filter application corresponds to one column i.e. input image coefficients are multiplied by the filter coefficients. The number of columns is equal to the number of outputs of one convolution filter, i.e. the number of terms in the sum (1). Here the authors use interleaved order of elements, i.e. write the first element of the first channel of the filter (or image), then the first element of the second channel of the filter (or image) and etc.

Such implementation allows to reduce execution time by using various optimizations of the matrix multiplication operation, for example, efficient memory access, Single Instruction Multiple Data extensions (SIMD-extensions), which can perform an operation on the vector of packed numbers simultaneously, and other parallelization capabilities of modern processors.

The model under consideration uses 8-bit unsigned integer coefficients, which are converted to 16-bit ones before multiplication. A 32-bit accumulator is used for addition. Therefore, overflows in the calculations are excluded and the resulting sum of products is correct for any input data.

2. The Elbrus Architecture

Elbrus is a microprocessor architecture with a very long instruction word. It means that the Elbrus processor is able to execute several elementary commands in one cycle and these commands together form one very long instruction word. Their formation takes place exclusively at the compilation stage of the program is performed by the optimizing compiler. It is able to perform comprehensive analysis of the source code and schedule instructions for computing devices more efficiently than a hardware instruction scheduler, much more time and compiling resources are available since at the compilation stage [19–21].

In the architecture VLIW principle is supported by using 6 sets of arithmetic logic units (ALU) for each processing core (each with its own instruction pipeline). We refer to each set of units as computing channel. These channels are not identical, because each channel contains arithmetic logic units supporting only some of the available operations. For example, the addition of integer values is supported by all channels, while the majority of vector operations performed on numbers packed in wide registers (SIMD commands) are available only on two channels. It means that the use of vector operations is not always justified in terms of computational efficiency.

With the development of the Elbrus architecture, ALU capabilities were sligntly changed. Table 1 shows an information on basic computing operations for various versions of the instruction set (IS). The length of a machine word is assumed to be 32 bits.

Table 1

Arithmetic logic devices parameters depending on the instruction set version

	Instruction set		
	3	4	5
Processors	Elbrus-4S	Elbrus-8S, Elbrus-1S+	Elbrus-8SV
SIMD register width, bits	64	64	128
Number of channels supporting the instruction / latency in cycles			
Integer add (64 bits)	6/1	6/1	6/1
Floating point add (64 bits)	4/4	6/4	6/4
Integer multy (32x32 $\rightarrow$ 64 bits)	4/4	4/4	4/4
Floating multiply add (64 bits)	4/8	6/8	6/8
Integer add (register)	2/1	2/1	2/1
Integer multiple with pairwise add (register)	2/2	2/2	2/2
Byte shuffle (64 bits)	4/1	4/1	4/1
Unpack (register)	2/1	2/1	2/1
APB load data	4/5	4/5	4/5

In addition, the Elbrus architecture provides hardware support for fast memory access via a hierarchical cache system and an array prefetch buffer (APB). APB provides fast loading of n-linear arrays stored in the main memory of a device. Unlike a cache, which is aimed at prefetching frequently used data, APB optimizes access time of arrays that are used a small number of times and are processed sequentially. APB usage has the following limitations:

- APB can be used in the absence of function calls during its utilization;
- in IS of versions 3 and 4 APB can only be used when working with aligned data, however, this requirement is relaxed in IS 5 and will be completely removed in IS 6.

Note that APB is effective

- with a sufficiently large number of iterations of the cycle in which APB is accessed;
- when accessing array elements spaced apart are no more than 32 bits;
- in the absence of memory conflicts of write operations between iterations, or with a sufficiently large distance of such dependencies.

3. An Approach to Efficient Matrix Multiplication Implementation for the Elbrus Architecture

Computationally efficient implementation of matrix multiplication should provide the maximum load of the processor ALUs, as well as the loading of the necessary data from the

a) b)

Fig. 2. a) Left operand matrix packing, b) right operand matrix packing

memory before the computation, since this time can be by times longer than the execution of elementary arithmetic operations. A distinctive feature of matrix multiplication is the fact that the same matrix element can be used to calculate several elements of the result. Therefore, [22] proposes an approach to loading matrix elements depending on matrix sizes and parameters of the memory subsystem. The main idea is to process the matrices by blocks, and the sizes of the blocks are selected such that one block fits in the level 1 (L1) and level 2 (L2) caches, respectively. This idea allows to provide fast access to the elements of the matrices and use the elements repeatedly to calculate the contribution of these elements to the whole block of the result.

In this paper, we use this algorithm with the only difference being that the elements were packed inside the matrix block (see Fig. 2). The goal of the packing operation is to ensure the optimal order of elements for their subsequent loading into registers and using vector commands from SIMD-extension. Packing needed to be done only once for each block, therefore its complexity is linearly depended on the number of matrix elements. In addition, the packing solves the problem on unavailability of APB due to arbitrary alignment of the addresses of matrix elements. To this end, we place additional zero elements to reach the nearest multiple of 8 (for 64-bit registers) for multiplication operands and the multiple of 2 for the result.

The proposed algorithm can be written in pseudocode as follows:

```
for bl_r in block_r(rhs):
    packed_r ← pack_rhs(bl_r)
    for bl_l in block_l(lhs):
        packed_l ← pack_lhs(bl_l)
        packed_res ← pack_res(bl_res)
        kernel(packed_res, packed_l, packed_r)
        bl_res ← unpack_res(packed_res).
```

Here `lhs` and `rhs` are the left and right operands of matrix multiplication, respectively, $\texttt{block}_l\texttt{(.)}$ and $\texttt{block}_r\texttt{(.)}$ are the methods to separate matrices `lhs` and `rhs` into blocks. The methods `pack_rhs` and `pack_lhs` implement block packing, `pack_res` implements result packing, `unpack_res` implements result unpacking. The method `kernel` performs computations on packed blockes and writes the result in a packed result block.

The pseudocode of the `kernel` method is given below:

```
for j in 0, ..., cols / nr
    {dst0, dst1} ← next result block
    for i in 0, ..., rows / mr
        for k in 0, ..., depth / 2
```

```
            bl_r ← next block of rhs
            bl_l ← next block of lhs
            lhs ← pshufb(zero, bl_l, 0x09010800009010800LL);
            rhs0 ← punpcklbh(zero, bl_r);
            rhs1 ← punpckhbh(zero, bl_r);
            dst0 ← dst0 + pmaddh(rhs0, lhs);
            dst1 ← dst1 + pmaddh(rhs1, lhs);
    // Handle the resting elements.
```

Here `pshufb` is an instruction of shuffling elements according to the rule, `punpckhbh` is an instruction of packing two high parts of 16-bit registers, `punpcklbh` is an instruction of packing two low parts of 16-bit registers, `pmaddh` is an instruction of long multiplication of 16-bit registers followed by pairwise addition of the products.

In the main cycle of the program, we perform loop unrolling in order to minimize memory accesses through the use of registers. The size of the processed result block is $nr \times mr = 12 \times 8$.

As a result, one iteration of the main loop take 8 cycles and processes 48 elements of the result. For real multiplication the duration of one iteration of the cycle is 8 clock cycles for IS 3, since the data is loaded by 14 APB commands, and the calculations are performed for 48 `fmadd` instructions, which are distributed over 8 wide commands.

Table 2 shows time measurements for multiplication of two 8-bit matrices using the proposed method and optimized floating-point multiplication on Elbrus-4S. Each measurement is averaged over $N = 10^6$ iterations.

Table 2

Multiplication time for A and B the matrices under the proposed method and optimized EML package

A size	B size	Time of the proposed method, ms	EML time, ms
16x9	9x100	0,04	0,01
16x9	9x400	0,15	0,04
16x25	25x400	0,18	0,06
16x144	144x400	0,29	0,20
16x400	400x400	0,62	0,50
16x400	400x1600	2,57	2,07
32x400	400x1600	4,57	3,94
32x800	800x1600	13,91	11,88
32x800	800x2500	14,05	11,90

It is obvious that, for Elbrus-4S, the implementation of integer matrix multiplication does not exceed the highly optimized material implementation in speed. It is happened due to the additional cost of data packaging. For subsequent generations of Elbrus processors, this gap will only worsen due to an increase in the number of ALUs working with floating point numbers.

Conclusion

The paper considers the widely used modern approach to reducing the inference time of neural network models, more specifically, the use of integers to store weights and intermediate results. However, despite the fact that this approach is effective for a number of modern devices, the approach is based on the properties of a specific microprocessor architectures and is not general. Therefore, the Elbrus architecture is specially optimized for floating-point computing and provides up to 6 64-bit floating-point ALUs, while ALU for vector integer operations (64 or 128 bit) is available only on 2 of the 6 processor channels. We propose, the implementation of integer matrix operations, which is used in neural network models with integer 8-bit coefficients and 32-bit intermediate calculation results. This implementation turned out to be comparable in computational efficiency with an optimized floating-point implementation for processors with IS 3 (Elbrus-4S), but does not exceed such an implementation for processors with IS 4 and 5 (Elbrus-8S and Elbrus-8SV) based on a theoretical assessment of their performance. Therefore, Elbrus architecture devices perform well in problems involving floating-point data, but may require special approaches when it comes to integer calculations. It means that in order to increase the computational efficiency of neural network models on such computers, approaches that use floating-point data in low-precision types [7] or reduce the structural complexity of neural network models [6] are more relevant. However, at the same time, the use of integer calculations can be justified to reduce the amount of memory required to store a neural network model, as well as to improve performance by placing all model coefficients and the input image in the cache (in case of small models) or random-access memory (in the case of complex models with hundreds of millions of parameters). Therefore, the given implementation, takes into account the features of the Elbrus platform and can find application in embedded systems with hard-fixed parameters of the calculator or in server applications that need to process a lot of data efficiently using various neural network models.

References

1. Limonova E.E., Bocharov N.A., Paramonov N.B., Bogdanov D.S., Arlazarov V.V., Slavin O.A., Nikolaev D.P. Recognition System Efficiency Evaluation on VLIW Architecture on the Example of Elbrus Platform. *Programming and Computer Software*, 2019, vol. 45, no. 1, pp. 15–21. DOI: 10.1134/S0132347419010047

2. Bulatov K.B., Arlazarov V.V., Chernov T.S., Slavin O.A., Nikolaev D.P. Smart IDReader: Document Recognition in Video Stream. *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 9-12 November, Kyoto, 2017, pp. 39–44. DOI: 10.1109/ICDAR.2017.347

3. Lynchenko A., Sheshkus A., Arlazarov V.L. Document Image Recognition Algorithm Based on Similarity Metric Robust to Projective Distortions for Mobile Devices. *International Conference on Machine Vision (ICMV 2018)*, 1-3 November, Munich, Germany, vol. 11041, article ID: 110411K, 7 p. DOI: 10.1117/12.2523152

4. Islam N., Islam Z., Noor N. A Survey on Optical Character Recognition System. *Journal of Information and Communication Technology*, 2016, vol. 10, no. 2, article ID: 18302720, 11 p. DOI: 10.1109/ICEDSS.2018.8544323

5. Bolotova Y.U., Spitsyn V.G., Rudometkina M.N License Plate Recognition Algorithm on the Basis of a Connected Components Method and a Hierarchical Temporal Memory Model. *Computer Optics,* 2015, vol. 39, no. 2, pp. 275–280. DOI: 10.18287/0134-2452-2015-39-2-275-280
6. Limonova E.E., Sheshkus A.V., Ivanova A.A., Nikolaev D.P. Convolutional Neural Network Structure Transformations for Complexity Reduction and Speed Improvement. *Pattern Recognition and Image Analysis*, 2018, vol. 28, no. 1, pp. 24–33. DOI: 10.1134/S105466181801011X
7. Johnson J. *Rethinking Floating Point for Deep Learning*, 2018. Available at: https://arxiv.org/abs/1811.01721 [accessed 01.10.2019]
8. Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, Yurong Chen. *Incremental Network Quantization: Towards Lossless CNNS with Low-Precision Weights*, 2017. Available at: https://arxiv.org/abs/1702.03044 [accessed 01.10.2019]
9. *Low-Precision Matrix Multiplication*. Available at: https://github.com/google/gemmlowp [accessed 01.10.2019]
10. *QNNPACK: Open Source Library for Optimized Mobile Deep Learning*. Available at: https://code.fb.com/ml-applications/qnnpack [accessed 01.10.2019]
11. Choukroun Y., Kravchik E, Kisilev P. *Low-Bit Quantization of Neural Networks for Efficient Inference*, 2019. Available at: https://arxiv.org/abs/1902.06822 [accessed 01.10.2019]
12. Prokhorov, N.L.,Kim A.K., Egorov G.A. To the 60th Anniversary of the I.S. Brook Institute of Electronic Control Computers. *Journal of Information Technologies and Computing Systems*, 2018, no. 3, pp. 1–13. DOI: 10.14357/20718632180301
13. Krizhevsky, A., Sutskever I., Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 2017, vol. 60, no. 6, pp. 84–90. DOI: 10.1145/3065386
14. Toshev A., Szegedy C. Deeppose: Human Pose Estimation Via Deep Neural Networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 17–19 June, Washington, 2014, pp. 1653–1660. DOI: 10.1109/CVPR.2014.214
15. Szegedy C., Wei Liu, Yangqing Jia, Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. Going Deeper with Convolutions. *IEEE Conference on Computer Vision and Pattern Recognition*, 7–12 June, Boston, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594
16. Bashivan P., Rish I., Yeasin M., Codella N. *Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks*, 2015. Available at: https://arxiv.org/abs/1511.06448 [accessed 01.10.2019]
17. Brahimi S., Aoun N.B., Amar C.B. Very Deep Recurrent Convolutional Neural Network for Object Recognition. *International Conference on Machine Vision*, 18–20 November, Nice, 2017, vol. 10341, article ID: 1034107.
18. Chellapilla K., Puri S., Simard P. High Performance Convolutional Neural Networks for Document Processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 23–26 October, La Baule, France, 2006, pp. 1237–1242.
19. Kim A.K., Perekatov V.I., Ermakov S.G. *Mikroprocessory i vychislitel'nye kompleksy semejstva Jel'brus* [Microprocessors and Computing Systems of the Elbrus Family]. Saint-Petersburg, Piter, 2013. (in Russian)
20. Ishin P.A., Loginov V.E., Vasilyev P.P. [Acceleration of Computations Using High-Performance Mathematical and Multimedia Libraries for the Architecture of Elbrus]. *Bulletin of Aerospace Defense*, 2015, no. 4 (8), pp. 64–68. (in Russian)

21. Limonova E.E., Skoryukina N.S., Neyman-Zade M.I. Fast Hamming Distance Computation for 2D Art Recognition on VLIW-Architecture in Case of Elbrus Platform. *International Conference on Machine Vision*, 16–18 November, Amsterdam, The Netherlands, 2019, vol. 11041, article ID: 110411N, 10 p. DOI: 10.1117/12.2523101

22. Goto K., Geijn R.A. Anatomy of High-Performance Matrix Multiplication. *Transactions on Mathematical Software*, 2008, vol. 34, no. 3, pp. 12. DOI: 10.1145/1356052.1356053

Received October 7, 2019

УДК 004.93 **DOI: 10.14529/mmp200109**

ОСОБЕННОСТИ РЕАЛИЗАЦИИ МАТРИЧНЫХ ОПЕРАЦИЙ В МАЛОБИТНЫХ НЕЙРОСЕТЕВЫХ МОДЕЛЯХ НА ПЛАТФОРМЕ ЭЛЬБРУС

***Е.Е. Лимонова*[1,2], *М.И. Нейман-заде*[3], *В.Л. Арлазаров*[1]**
[1]Федеральный исследовательский центр «Информатика и управление» РАН, г. Москва, Российская Федерация
[2]ООО «Смарт Энджинс Сервис», г. Москва, Российская Федерация
[3]АО МЦСТ, г. Москва, Российская Федерация

В работе исследуется возможность эффективной реализации вычислений в малобитных нейросетевых моделях на платформе с VLIW архитектурой Эльбрус. Такие модели широко применяются на практике для повышения вычислительной эффективности распознавания и хорошо подходят для вычислителей таких архитектур, как x86 и ARM. В данной работе была рассмотрена 8-битная нейросетевых модель, в которой наиболее ресурсоемкой частью реализации является матричное умножение. В данной работе приведена эффективная реализация матричного умножения, учитывающая особенности архитектуры Эльбрус: наличие нескольких вычислительных каналов с различными арифметико-логическими устройствами, буфера предварительной подкачки массивов и собственного SIMD-расширения. Проведено теоретическое и экспериментальное сравнение вычислительной производительности малобитной и классической нейросетевых моделей, показавшее, что процессоры Эльбрус имеют гораздо больше возможностей для выполнения оптимальных вещественных вычислений и требуют разработки новых подходов к повышению вычислительной эффективности нейросетевых моделей.

Ключевые слова: малобитные нейронные сети; вычислительная эффективность; архитектура Эльбрус; матричные операции.

Литература

1. Лимонова, Е.Е. Оценка быстродействия системы распознавания на VLIW архитектуре на примере платформы Эльбрус / Е.Е. Лимонова, Н.А. Бочаров, Н.Б. Парамонов, Д.С. Богданов, В.В. Арлазаров, О.А. Славин, Д.П. Николаев // Программирование. – 2019. – № 1. – С. 15–21.

2. Bulatov, K.B. Smart IDReader: Document Recognition in Video Stream / K.B. Bulatov, V.V. Arlazarov, T.S. Chernov, O.A. Slavin, D.P. Nikolaev // IAPR International Conference on Document Analysis and Recognition (ICDAR), 9–12 November. – Kyoto, 2017. – P. 39–44.

3. Lynchenko, A. Document Image Recognition Algorithm Based on Similarity Metric Robust to Projective Distortions for Mobile Devices / A. Lynchenko, A. Sheshkus, V.L. Arlazarov // International Conference on Machine Vision (ICMV 2018), 1–3 November. – Munich, 2019. – V. 11041. – Article ID: 110411K. – 7 p.
4. Islam, N.A Survey on Optical Character Recognition System / N. Islam, Z. Islam, N. Noor // Journal of Information and Communication Technology. – 2016. – V. 10, № 2. – Article ID: 18302720. – 11 p.
5. Болотова, Ю.А. Распознавание автомобильных номеров на основе метода связных компонент и иерархической временной сети / Ю.А. Болотова, В.Г. Спицын, М.Н. Рудометкина // Компьютерная оптика. – 2015. – V. 39, № 2. – С. 275–280.
6. Limonova, E.E. Convolutional Neural Network Structure Transformations for Complexity Reduction and Speed Improvement / E.E. Limonova, A.V. Sheshkus, A.A. Ivanova, D.P. Nikolaev // Pattern Recognition and Image Analysis. – 2018. – V. 28. – № 1. – P. 24–33.
7. Johnson J. Rethinking Floating Point for Deep Learning / J. Johnson. – 2018. – URL: https://arxiv.org/abs/1811.01721 [дата обращения 01.10.2019]
8. Aojun Zhou. Incremental Network Quantization: Towards Lossless CNNS with Low-Precision Weights / Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, Yurong Chen. – 2017. – URL: https://arxiv.org/abs/1702.03044 [дата обращения 01.10.2019]
9. Low-Precision Matrix Multiplication. – URL: https://github.com/google/gemmlowp [дата обращения 01.10.2019]
10. QNNPACK: Open Source Library for Optimized Mobile Deep Learning. – URL: https://code.fb.com/ml-applications/qnnpack [дата обращения 01.10.2019]
11. Choukroun, Y. Low-Bit Quantization of Neural Networks for Efficient Inference / Y. Choukroun, E. Kravchik, P. Kisilev. – 2019. – URL: https://arxiv.org/abs/1902.06822 [дата обращения 01.10.2019]
12. Прохоров, Н.Л. К 60-летию Института электронных управляющих машин им. И.С. Брука / Н.Л. Прохоров, А.К. Ким, Г.А. Егоров // Информационные технологии и вычислительные системы. – 2018. – № 3. – С. 1–13.
13. Krizhevsky, A. ImageNet Classification with Deep Convolutional Neural Networks / A. Krizhevsky, I. Sutskever I., G.E. Hinton // Communications of the ACM. – 2017. – V. 60, № 6. – P. 84–90.
14. Toshev, A. Deeppose: Human Pose Estimation Via Deep Neural Networks / A. Toshev, C. Szegedy // IIEEE Conference on Computer Vision and Pattern Recognition, 17–19 June. – Washington, 2014. – P. 1653–1660.
15. Szegedy, C. Going Deeper with Convolutions / C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovic // IEEE Conference on Computer Vision and Pattern Recognition, 7–12 June. – Boston, 2015. – P. 1–9.
16. Bashivan, P. Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks / P. Bashivan, I. Rish, M. Yeasin, N. Codella. – 2015. – URL: https://arxiv.org/abs/1511.06448.
17. Brahimi, S. Very Deep Recurrent Convolutional Neural Network for Object Recognition / S. Brahimi, N.B. Aoun, C.B. Amar // International Conference on Machine Vision, 18–20 November. – Nice, 2017. – V. 10341. – Article ID: 1034107.
18. Chellapilla, K. High Performance Convolutional Neural Networks for Document Processing / K. Chellapilla, S. Puri, P. Simard // Tenth International Workshop on Frontiers in Handwriting Recognition, 23–26 October. – La Baule, 2006. – P. 1237–1242.

19. Ким, А.К. Микропроцессоры и вычислительные комплексы семейства Эльбрус / А.К. Ким, В.И. Перекатов, С.Г. Ермаков – СПб.: Питер, 2013.

20. Ишин, П.А. Ускорение вычислений с использованием высокопроизводительных математических и мультимедийных библиотек для архитектуры Эльбрус / П.А. Ишин, В.Е. Логинов, П.П. Васильев // Вестник воздушно-космической обороны. – 2015. – № 4 (8). – С. 64–68.

21. Limonova, E.E. Fast Hamming Distance Computation for 2D Art Recognition on VLIW-Architecture in Case of Elbrus Platform / E.E. Limonova, N.S. Skoryukina, M.I. Neyman-zade // International Conference on Machine Vision, 16–18 November. – Amsterdam, 2019. – V. 11041. – Article ID: 110411N. – 10 p.

22. Goto, K. Anatomy of High-Performance Matrix Multiplication / K. Goto, R.A. Geijn // Transactions on Mathematical Software. – 2008. – V. 34, № 3. – P. 12.

Елена Евгеньевна Лимонова, магистр, Федеральный исследовательский центр «Информатика и управление» РАН, Институт системного анализа (г. Москва, Российская Федерация); ООО «Смарт Энджинс Сервис» (г. Москва, Российская Федерация), elena.e.limonova@gmail.com.

Мурад Искендер-оглы Нейман-заде, кандидат физико-математических наук, начальник отделения систем программирования, АО МЦСТ (г. Москва, Российская Федерация), muradnz@mcst.ru.

Владимир Львович Арлазаров, доктор технических наук, профессор, член-корреспондент РАН, Федеральный исследовательский центр «Информатика и управление» РАН (г. Москва, Российская Федерация), vladimir.arlazarov@gmail.com.

Поступила в редакцию 7 октября 2019 г.