IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 1, 2021, accepted June 25, 2021, date of publication July 5, 2021, date of current version July 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094484

Bipolar Morphological Neural Networks: Gate-Efficient Architecture for Computer Vision

ELENA E. LIMONOVA[1,2], (Member, IEEE), DANIIL M. ALFONSO[3], DMITRY P. NIKOLAEV[2,4], (Member, IEEE), AND VLADIMIR V. ARLAZAROV[1,2], (Member, IEEE)

[1]FRC CSC RAS, 119333 Moscow, Russia
[2]Smart Engines Service LLC, 117312 Moscow, Russia
[3]JSC MCST, 117105 Moscow, Russia
[4]IITP RAS, 127051 Moscow, Russia

Corresponding author: Elena E. Limonova (elena.e.limonova@gmail.com)

This work was supported in part by the Russian Foundation for Basic Research under Project 18-29-26017 and Project 18-29-26020.

ABSTRACT The priority of building hardware-oriented neural network models is growing steadily. The target goals for their development are the performance and energy efficiency of promising hardware-software solutions. Simultaneously, for different classes of computing architectures of the computer, the optimal neural network models will differ. The most interesting from a practical point of view are application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs) and central processing units (CPUs). We have recently proposed a bipolar morphological network as a hardware-oriented model for these computer types, the computationally intensive parts of which use only maximum and addition. In this work, we present for the first time a theoretical assessment of the expressive power of a neural network consisting of BM neurons and show that it corresponds to the expressive power of the classical multilayer perceptron. In addition, we summarize the current results on the use of the bipolar morphological model in typical tasks of technical vision: image classification and semantic segmentation. We consider simple LeNet-5-like neural networks, as well as deeper ResNet and UNet architectures. We show that BM networks demonstrate accuracy that allows their practical use, with significantly higher performance in terms of a transistor budget for two (ASIC, FPGA) of the three architectures under consideration. The source code of the model and ResNet experiments are available at https://github.com/SmartEngines/bipolar-morphological-resnet.

INDEX TERMS ASIC, bipolar morphological networks, computational complexity, expressive power, FPGA.

I. INTRODUCTION

Neural network recognition has gone far beyond academic tasks and is widely used in end users' products [1]–[4]. The rising importance of the edge computing concept forces us to move calculations closer to the end-user. Therefore, in today's world, execution speed, required memory amount, and energy efficiency have become very important parameters as well as recognition quality. These parameters are often decisive when choosing a specific solution, for example, in the case of unmanned vehicles and embedded systems, such as smart homes. Moreover, energy efficiency is also important for environmental protection, so more and more attention is paid to it [5], [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Ali.

Thus, to assess a particular solution's applicability, we need to select a neural network model that provides sufficient accuracy and analyze its inference on the target device. At the same time, there are many modern computer types and various opportunities for the development of specialized devices. For example, central (CPU) and graphic processing units (GPU) are widely available for software projects. However, they have a fixed architecture, which is often not optimized for neural networks. On the other hand, developers of hardware platforms spend a lot of resources to create an optimal low-level device for solving a specific problem, often without being able to change the solution itself. Application-Specific Integrated Circuits (ASICs, for example, Google TPU or Intel VPU) for neural network processing is a compromise, as well as specialized neuromorphic processors. They are optimized for the fast inference of certain

classes of existing models, which have proven themselves well both in terms of quality and performance. Developers can easily use them within their existing infrastructure.

The next step in developing neural network technologies is creating a model, allowing efficient and easily scalable hardware implementation still maintaining sufficient quality. This task requires integrating knowledge from technical vision and hardware engineering with a certain amount of luck and inspiration. The bipolar morphological (BM) neuron model is an attempt to create such a device-oriented neural network model.

In this paper, we describe current research on the bipolar morphological neuron model. Our contribution is as follows:

- we summarize the accuracy study of convolutional BM networks in technical vision problems;
- for the first time, we analyze the expressive power of a neural network consisting of BM neurons;
- we estimate the complexity of the BM network implementation on CPU and FPGA.

The paper is organized as follows: Section 2 is devoted to computer models, the evolution of an artificial neuron model, and an artificial neural network to show the BM neuron's origin. In Section 3, we present the BM neuron, the expressive power of BM networks, their computational complexity for various computers, and the accuracy of BM networks in several technical vision tasks. Section 4 contains an overview of related works devoted to creating efficient neural network models, and Section 5 concludes the work. In Section 6, we discuss the further development of the proposed model.

II. THE EVOLUTION OF COMPUTING AND ARTIFICIAL NEURAL NETWORKS

Modern computing devices have a rather complex architecture. For example, they support pipelining, out-of-order instructions execution, a hierarchical data caching system, etc. Thus, the actual inference time of a particular neural network depends on the number of parameters and its architecture inextricably for each device. Let us consider the most relevant computer models.

A. COMPUTER MODELS

1) LOGIC CIRCUITS

A logic gate is a physical device that realizes a Boolean function. A logic circuit is a directed acyclic graph in which all vertices except input vertices carry the labels of logic gates [7]. It means that the logic circuit computes a binary function $f : B_n \to B_m, B \in 0, 1$ that is a mapping from the values of its n input variables to the values of its m outputs. Using logic circuits one can execute programs without loops and branches (straight-line programs).

The main characteristics of the logic circuits are size and depth. The size is the number of vertices in the circuit graph, while the depth is the number of edges in the longest path from the input to the output. The size defines the required number of gates to implement the logic circuit and therefore is connected to energy-efficiency and complexity of the resulting chip. The depth characterizes latency, i.e. time required to obtain correct output for a new supplied input.

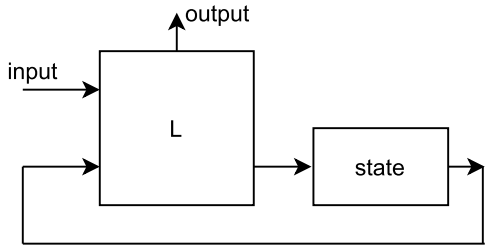

FIGURE 1. The finite-state machine scheme.

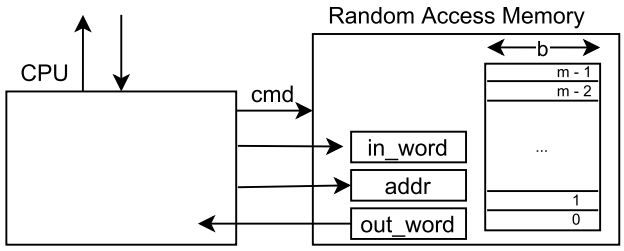

FIGURE 2. The random-access machine scheme.

2) FINITE-STATE MACHINE

The finite-state machine (FSM) is a machine with memory, where its state is stored. It always acts taking into account the state. It can be seen as a logic circuit which takes current state as one of inputs and outputs a target function and a new state (see Fig. 1).

3) RANDOM-ACCESS MACHINES

The random-access machines (RAM) model is the most close to the traditional computer. The RAM is modeled by two synchronous interconnected FSMs, a central processing unit (CPU) and a random-access memory (see Fig. 2) [7]. The CPU has its own data storage places called registers. The number of registers is usually small. The CPU is able to perform operations only on the data stored in registers. The random-access memory is a large data storage.

The CPU works in a fetch-and-execute cycle, i.e. reads instructions from random-access memory and executes them. A CPU typically has five basic kinds of instruction: a) arithmetic and logical instructions, b) memory load and store instructions for moving data between random-access memory and registers, c) jump instructions for breaking out of the current program sequence, d) input and output instructions, and e) a halt instruction.

The random-access memory operates three input words and one output word. These three words are an address, a data word, a command. The command can be reading, writing or doing nothing. While reading the memory returns a word (`out_word`) from the address location (`addr`) and while writing it stores a data word (`in_word`) to the address

location (addr). The memory is called random-access because it has the same time to access a word for all words. The RAM model can have bounded-memory, which means that the storage amount is limited to some value $m = 2^r$, data words have fixed size of 2^b bytes and addressed with r-bit variables.

4) CORRESPONDENCE TO REAL COMPUTERS

In practice, performance-demanding mobile and embedded systems typically use FPGAs, specialized hardware, and central processors. In the case of an FPGA or embedded device, the primary importance have the size and depth of the logic circuit. For central processing units, internal parallelism must also be considered. The data-level parallelism is often supported through Single Instruction Multiple Data (SIMD) extensions that allow one to perform the same arithmetic operation on several data elements of a vector simultaneously. We will consider SIMD instructions as ordinary arithmetic operations, remembering that it is necessary to load and prepare several data values before an operation. Other types of parallelism will remain outside the scope of this paper.

Another CPU difference is the hierarchical memory structure. Generally, coefficients, input data, and intermediate values do not fit into registers and even L1 cache during neural networks inference. Therefore, memory access must be paid attention to when creating an optimal implementation.

B. NEURON MODELS IN COMPUTER VISION

In living systems, signals from the nervous system and the brain control all processes. Modeling the nervous system and its parts is an area of interest for many researchers. Most models describing the state of a neuron use differential equations – for example, the integrate-and-fire model and its generalizations, the Hodgkin-Huxley model, etc. [8].

Physically, these signals represent electric potential differences, so it is not surprising that people have tried to simulate them using electrical circuits. Such devices were called neuromorphic devices or neuromims. Their creation began in the 60s of the 20th century by L. Harmon [9] and E. Lewis [10], who proposed electrical circuits to simulate the Hodgkin-Huxley neuron. However, the production of devices with a large number of neurons to solve meaningful tasks was impossible at that time. Therefore, the studies were focused on simplified models of neurons and neural structures reproducing information processing in living systems.

1) CLASSICAL MATHEMATICAL NEURON MODEL

In 1943 Warren McCulloch and Walter Pitts published an article «A logical calculus of the ideas immanent in nervous activity» [11]. They studied neurons in the brain to create their mathematical model, which could become the basis of artificial intelligence. Their artificial neuron was binary, i.e., depending on the input signals, it could only be in an excited or unexcited state. Like biological neurons, it had a body connected by synapses to several dendrites, which received input signals, and one axon, which served as an output. Dendrite synapses attenuate or amplify input signals by multiplying by a weighting factor and transmit the result to the neuron body represented by an adder. When this sum exceeds a certain threshold value, the neuron moves to an excited state. The following expression describes the McCulloch-Pitts neuron:

$$f(x) = \theta\left(\sum_{i=1}^{N} w_i x_i + w_0\right), \tag{1}$$

where x is the vector of input signals, w is the weight vector of the neuron, θ is a threshold activation function:

$$\theta(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{2}$$

McCulloch and Pitts built the simplest neural network from such mathematical neurons and showed that it could calculate various mathematical functions. They also proposed simulating the self-learning phenomenon observed in natural neural networks by changing the weight coefficients in response to specific sequences of input signals.

This model was later generalized to an artificial neuron using an arbitrary activation function φ. It is this model that is often called the classical mathematical neuron model:

$$f(x) = \varphi\left(\sum_{i=1}^{N} w_i x_i + w_0\right). \tag{3}$$

2) MORPHOLOGICAL NEURON MODEL

In 1990 G. Ritter and his colleagues proposed an alternative neuron model and the neural network structure, which he called the morphological model of the neuron and the morphological neural network, respectively [12]. In the morphological model, the neuron's body does not add but takes the maximum or minimum. Weighting factors affect input signals additively, not multiplicatively. Further, the found value of the maximum or minimum is compared with the threshold, and, thus, the output of the morphological neuron is binary. The following formula can describe a morphological neuron:

$$f(x) = \theta\left(p \max_{i=1}^{N} r_i(x_i + w_i) - w_0\right), \tag{4}$$

where x is the vector of input signals, w is the weight vector of the neuron, θ is a threshold activation function, $r_i \in \{-1, 1\}$ are responsible for the effect of the i^{th} input signal (excitation or inhibition), and $p \in \{-1, 1\}$ manipulates the output sign.

Such a neuron turns out to be more computationally efficient than a classical mathematical neuron since the operations of addition/subtraction and taking the maximum/minimum require fewer logical gates for implementation than the multiplication operation. Therefore it is more energy-efficient and needs significantly less time to compute.

Further development of the morphological neuron was the model of a morphological neuron with dendrites proposed by Ritter in 2003 [13]:

$$f(x) = \theta\left(\min_{k=1}^{K} p_k \left(\min_{i,l}(-1)^{1-l}(x_i + w_{ik}^l)\right) - b\right), \tag{5}$$

where x is the vector of input signals, w is the weight vector of the neuron, θ is a threshold activation function, $l \in \{0, 1\}$, and $p \in \{-1, 1\}$ manipulates the output sign.

Dendrites made it possible to control the neuron's excitation and inhibition flexibly since the neuron state depended on several input signals' combinations with different signs and weights. A feature of the model is that the required number of dendrites was obtained during training the model, i.e., dendrites «grow» as needed. Also, in this work, the authors proved that a morphological perceptron with dendrites can solve any classification problem with any given accuracy. However, the number of neurons required can be extremely large, which negates the advantages of each neuron's simplicity.

Nevertheless, these morphological models were unable to provide sufficient recognition quality compared to the quality of classical perceptron networks and the quality of convolutional neural networks in typical problems. Only recently, the studies of morphological networks have been resumed: new training methods are being developed [14], [15], and ways of using morphological models in real problems are considered (for example, electroencephalogram decoding [16]). Also, layers of morphological neurons with dendrites show promising results in hybrid models where morphological neurons are used to extract features in some network layers [17].

3) SPIKING NEURON MODELS

With the increase in the computing power of microchips, the interest in constructing devices simulating the natural neural network has increased. A feature of the transmission of real nerve impulses is their spike nature; that is, neurons' input and output signals are sequences of impulses. The first model of a spiking neuron was the model by Alan Hodgkin and Andrew Huxley, proposed in 1952. They described the mechanisms underlying the origin and propagation of signals along an axon. This model is one of the most biologically accurate models, and therefore it is pretty complex: it contains four differential equations, and a lot of parameters [8].

However, the most famous impulse neuron model was proposed by the French physiologist Louis Lapicque in 1907. This model is called the «integrate-and-fire» neuron. The idea behind it is that when there is some signal at the neuron's input, its output rises until a threshold. Then the output drops drastically, and the process starts over. The disadvantage of this model is that with a linear increase in the input signal, the firing frequency of the neuron grows indefinitely. To eliminate this drawback, too frequent neuron triggering was prohibited. The second drawback is that the output signal will remain intermediate value between the maximum and minimum if the neuron did not trigger. To fix it, a new «leaky integrate-and-fire» neuron was developed.

The leaky integrate-and-fire model introduces the «resistance» of the neuron, which causes a progressive leakage, so the output gradually resets even if the input did not reach the threshold [8].

In 2003 Evgene Izhikevich developed a new class of spiking neuron models described by just two simple differential equations. On the one hand, such neurons are quite simple and computationally efficient, and on the other hand, they retain the biological accuracy of the Hodgkin-Huxley model [18].

Today, spiking neural networks are developing as mathematical models and as special neuromorphic chips. These chips are processors that effectively implement spiking neural networks and are characterized by extremely low power consumption. For example, in 2014, IBM released the TrueNorth neuromorphic chip, which consists of a million neurons that can form 256 million connections. In 2017, Intel introduced the Loihi neuromorphic chip that supports training neural network models. In 2020, the Akida neural processor was created with approximately 1.2 million neurons that can form 10 billion connections.

However, despite the variety of models and active research of spiking neural networks, there are still no stable training methods that would allow obtaining sufficiently accurate solutions to modern recognition problems [19]. Simultaneously, work in this area continues and shows promising results [20]–[22].

4) ARTIFICIAL NEURON MODELS IN STATE-OF-THE-ART NETWORKS

After creating the McCulloch-Pitts neuron, the following fundamental issue was the organization of neurons into a network. One option was the perceptron, in which individual neurons were organized into layers, and each neuron of the next layer was connected to some neurons of the previous one. At first, the training was performed with an error-correction method (F. Rosenblatt, [23]), and later with an error back-propagation method using various metrics (D. Rumelhart, [24]). Despite criticism and limitations of such a model (for example, image analysis was non-invariant to spatial shifts), this model solved a number of simple problems.

Another option for organizing a neural network was the cognitron (1975), and the neocognitron (1980) by K. Fukushima [25]. The neocognitron allowed for invariant analysis of the input image. It used neurons of two types for this purpose: S- («simple») and C- («complex»). «Simple» neurons were connected with some fixed area of input sensors and selected some specific feature. «Complex» neurons received the outputs of «simple» ones and located features in the entire image. The neocognitron had a hierarchical structure and could include several layers of «simple» and «complex» neurons. As a result, it was able to recognize objects in the image, regardless of their position and small deformations. Although neocognitron applications were limited only to image recognition tasks, it demonstrated impressive results in these tasks.

Convolutional neural network architectures we know, starting with LeNet-5 [26] and AlexNet [27], are based on the neocognitron and the classical mathematical neuron. The combination of a relatively simple convolutional layer structure with a small number of weights and the increased

computing power of computers made it possible to solve practical problems. At the same time, researchers began to study not individual neurons but entire layers with special properties, which were then combined to create a network. Let us list the main layers of state-of-the-art networks:

- fully-connected layer;
- convolutional layer;
- pooling layer;
- normalizing layer;
- long short-term memory layer;
- and etc.

Layers can be combined in complex ways to form a graph with layers as vertices and edges representing data flow.

In our work, we propose a new bipolar morphological neuron and suggest to modify neuron type keeping layer types and overall graph structure of the network unchanged. Currently we recommend to use BM neuron in convolutional layers only.

III. BIPOLAR MORPHOLOGICAL NEURON

A. THE BIPOLAR MORPHOLOGICAL NEURON

The idea behind the proposed neuron is to simplify computations in the classical mathematical neuron (3) to get maximum and addition as the main neuron operations.

To construct such a neuron we use the approximation of the classical one. At first we consider the case $x_i \geq 0$, $w_i \geq 0$:

$$\begin{aligned}\sum_{i=1}^{N} x_i w_i &= \exp\{\ln \sum_{i=1}^{N} x_i w_i\} \\ &= (1+k)\exp \max_i (\ln x_i + \ln w_i) \\ &= (1+k)\exp \max_i (y_i + v_i) \\ &\approx \exp \max_i (y_i + v_i), \end{aligned} \tag{6}$$

where $y_i = \ln x_i$ are new inputs, $v_i = \ln w_i$ are new weights, and

$$k = \frac{\sum_{j=1}^{N} x_j w_j}{M} - 1, \tag{7}$$

$$M = \max_j (x_j w_j). \tag{8}$$

Then we consider a full classical neuron:

$$\begin{aligned}\sum_{i=1}^{N} x_i w_i = &\sum_{i=1}^{N} p_i^{++} x_i |w_i| - \sum_{i=1}^{N} p_i^{+-} x_i |w_i| \\ &- \sum_{i=1}^{N} p_i^{-+} |x_i| w_i + \sum_{i=1}^{N} p_i^{--} |x_i||w_i|, \end{aligned} \tag{9}$$

where

$$p_i^{++} = \begin{cases} 1, & \text{if } x_i \geq 0 \text{ and } w_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$p_i^{-+} = \begin{cases} 1, & \text{if } x_i < 0 \text{ and } w_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$p_i^{+-} = \begin{cases} 1, & \text{if } x_i \geq 0 \text{ and } w_i < 0 \\ 0, & \text{otherwise} \end{cases}$$

$$p_i^{--} = \begin{cases} 1, & \text{if } x_i < 0 \text{ and } w_i < 0 \\ 0, & \text{otherwise} \end{cases}$$

Each term here deals with non-negative inputs and weights and can be approximated using (6).

Finally we obtain a new neuron:

$$\begin{aligned} y_{BM}(\mathbf{x}, V, v) = \varphi \Big(&\exp \max_{j=1}^{N} (\ln x_j^+ + v_j^+) \\ &- \exp \max_{j=1}^{N} (\ln x_j^+ + v_j^-) \\ &- \exp \max_{j=1}^{N} (\ln x_j^- + v_j^+) \\ &+ \exp \max_{j=1}^{N} (\ln x_j^- + v_j^-) + v_0 \Big), \end{aligned} \tag{10}$$

$$x_j^+ = \begin{cases} x_j, & x_j \geq 0, \\ 0, & x_j < 0, \end{cases} \tag{11}$$

$$x_j^- = \begin{cases} -x_j, & x_j < 0, \\ 0, & x_j \geq 0, \end{cases} \tag{12}$$

where $\mathbf{x}$ is an input vector of length N, v^+, v^- are weight vectors of size N, v_0 is bias, $\varphi(\cdot)$ is a nonlinear activation function. We define $\ln 0 = -\infty$ and replace it by a big enough negative value for actual computations.

We call it bipolar morphological (BM) neuron. The term bipolar means that they participate in two separate processes: excitation and inhibition. In the equation (10) each pair of input and weight is included in one sum only depending on the sign combination. In real life bipolar neurons are specialized neurons for sensory perception and can be found, for example, in the retina.

The BM neuron is not computationally efficient yet, because of logarithm and exponent operations. However, neurons in the neural network are organized into layers. In the BM layer consisting of BM neurons (fully-connected or convolutional layer) ln and exp operations are performed on the activation (the signal transmitted between network layers) and can be considered as a part of an activation function. Normally the activation functions do not make significant contributions to the computational complexity of a neural network, so the increase of computational complexity should be of little consequence.

In the Fig. 3 we illustrate the BM layer structure. The rectifier (ReLU) allows us to take values above zero and create four computational path for positive and negative input or weights. Then we take the logarithm of rectified input (x) and perform essential morphological operation of the layer. The results are passed to the exponential unit and subtracted to get the output (y).

B. BM NEURON ACCURACY AND EXPRESSIVE POWER

In the above Subsection we considered the BM neuron as the approximate classical mathematical neuron. However,

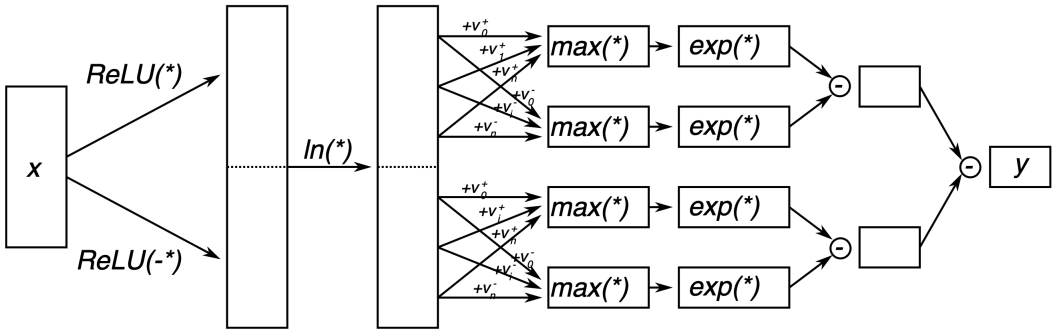

FIGURE 3. The structure of a BM neuron with input vector x and weights v_i^+, v_i^-.

the approximation (6) is good only in case $k \ll 1$. For a real neuron $0 \leq k \leq N-1$, and the best case is the sum containing only one non-zero term ($k = 0$) and the worst case is the sum with all equal terms ($k = N-1$). In this case the real value for the sum will be N times more than approximated. On one hand, we can use additional coefficients to improve the approximation accuracy, on the other hand, BM networks already have the same expressive power as traditional multilayer perceptrons have.

Though we are going to mix up BM and other more traditional neurons in a single network, here we consider networks of BM neurons only (pure BM networks). Namely, each continuous function f on any compact set in the feature space $\mathbb{R}^n$ can be approximated uniformly with any precision by some pure BM network.

The idea of the proof is the same as that for 3-layer perceptrons. Here are some hints how to construct such an approximating BM-network.

Let us consider identity activation function $\varphi(\cdot)$. The first two layers of the approximating BM network will depend only on the bounding box of the domain of the function f to be approximated.

The first layer consists of a number of neurons with single input (any of the features x_j), computing functions of shape $x_j + b_i^+$, $-x_j + b_i^-$ for some set of biases b_i^*, and one neuron with bias -1 and no input. The second layer consists of neurons with $2n+1$ inputs and computes indicator functions of a number of parallelepipeds in the feature space. At last, the only neuron of the third layer has a huge number of inputs and computes a piecewise-linear (and almost piecewise-constant) function approximating f.

C. COMPUTATIONAL EFFICIENCY

1) COMPUTATIONAL COMPLEXITY

At first it should be noted that although computing the positive and negative part of input (x^+ and x^- in (10)) doubles its length, there are exactly half of zero elements in them in total. These terms do not require logarithm computation.

The standard convolutional layer with input $I_{L\times M\times C}$ and output $O_{L\times M\times F}$ does the following:

$$O(l,m,f) = \varphi\left(\sum_{c=1}^{C}\sum_{\Delta l=0}^{K-1}\sum_{\Delta m=0}^{K-1} I(l+\Delta l, m+\Delta m, c)\cdot \right.$$
$$\left. \cdot w(\Delta l, \Delta m, c, f) + b(f)\right),$$
$$f = \overline{1,F}, l = \overline{1,L}, m = \overline{1,M} \quad (13)$$

Here F is the number of filters, C is the number of input channels, $K \times K$ is the spatial dimensions of the filter, input image size is $L \times M \times C$, w is a set of convolutional filters, b is the bias. We suppose I is padded properly for the result to be of the same size.

The BM convolutional layer with the same input and output sizes:

$$J = \varphi\left(\sum_{\alpha}\sum_{\beta} p^\alpha p^\beta \exp(\ln I^\alpha \odot v^\beta) + \mathbf{b}\right),$$

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

$$(I \odot v)_{n,m,c} = \max_{c=1}^{C}\max_{\Delta n=0}^{K-1}\max_{\Delta m=0}^{K-1}\left(I_{n+\Delta n, m+\Delta m, c} + v_{\Delta k, \Delta m, c, f}\right)$$

The standard fully-connected layer with input I_P and output O_Q does:

$$O(q) = \sigma\left(\sum_{p=1}^{P} I(p)\cdot w(p,q) + b(q)\right),\ q = \overline{1,Q} \quad (14)$$

Here P is the number of inputs, Q is the number of neurons in the layer, w is a set of fully-connected weights, b is set of biases.

The BM fully-layer with the same input and output sizes:

$$O(q) = \sigma\left(\sum_{\alpha,\beta} p^\alpha p^\beta \exp\max_{p=1}^{P}(\ln I^\alpha(p) + v^\beta(p,q) + b(q)\right),$$
$$q = \overline{1,Q} \quad (15)$$

TABLE 1. The number of operations in the convolutional (conv) layer of BM and standard models. F is the number of filters, C is the number of input channels, $K \times K$ is the spatial dimensions of the filter, input image size is $L \times M \times C$.

Op	Standard conv	BM conv
$\sigma(\cdot)$	FLM	FLM
Exp	0	$4FLM$
Log	0	CLM
Add	FK^2CLM	$2F(K^2C+2)LM$
Max	0	$2F(K^2C-1)LM$
Mul	FK^2CLM	0

TABLE 2. The number of operations in the fully-connected (fc) layer of BM and standard models. P is the number of inputs, Q is the number of neurons in the layer.

Op	Standard fc	BM fc
$\sigma(\cdot)$	Q	Q
Exp	0	$4Q$
Log	0	P
Add	QP	$2Q(P+2)$
Max	0	$2Q(P-1)$
Mul	QP	0

The number of operations for the standard and BM convolutional layers is shown in Table 1. In Table 2, we show the number of operations for standard and BM fully-connected layers.

2) THE BM NETWORKS FOR CPUs

Modern CPUs use general-purpose Arithmetic Logic Units (ALUs) for computations like addition, multiplication and maximum. They also include SIMD-extensions for arithmetics that operate 128-512 bit vector data. In the Table 3 we show operating time and throughput for these operations for several x86_64 and ARM processors. There is no difference for single-precision floating-point data and a little difference for integer data. It means that the BM networks have practically no advantage on CPUs over classical networks.

3) FPGA COMPLEXITY

The most promising application of BM networks is FPGA and ASIC devices, because specialized addition and maximum blocks are able to perform faster than general-purpose ALUs. Moreover, BM neuron/layer can be constructed using 4 sets of parallel units to speed up execution (one for each computational branch).

Firstly, we estimate the number of logic gates and clock cycle latency required for the arithmetical operations involved in the computations. We obtain register transfer level description of addition, multiplication and maximum computation arithmetic units using Verilog HDL conforming to IEEE 754 single-precision floating-point standard, and Synopsys Design Compiler to implement them at gate-level and get logic gate complexity and latency characteristics.

TABLE 3. The latency and average throughput of arithmetic operations for scalar and vector data types [28], [29].

Op	latency	throughput
Intel Skylake-X, floating-point 128-bit vector		
add	4	0.5
max	4	0.5
mul	4	0.5
Intel Skylake-X, integer 128-bit vector		
add	1	0.33
max	1	0.5
mul	5	0.5
mul+add	5	0.5
Intel Skylake-X, single-precision floating-point		
add	3	1
max	4	0.5
mul	5	1
Intel Coffee Lake, floating-point 128-bit vector		
add	4	0.5
max	4	0.5
mul	4	0.5
Intel Coffee Lake, integer 128-bit vector		
add	1	0.33
max	1	0.5
mul	5	0.5
mul+add	5	0.5
Intel Coffee Lake, single-precision floating-point		
add	3	1
max	4	0.5
mul	5	1
ARM Cortex-A57, floating-point 128-bit vector		
add	5	2
max	5	2
mul	5	2
ARM Cortex-A57, integer 128-bit vector		
add	3	2
max	3	2
mul	5	1
mul+add	5	1
ARM Cortex-A57, floating-point 128-bit vector		
add	5	2
max	5	2
mul	5	2

We show the results for 65 and 16 nm technologic libraries in Table 4.

We use software approximation for logarithm and exponent. The exponent is implemented as in a reference implementation from Intel [30]. For a logarithm we construct our own implementation, which gives the precision of 4 decimal digits and uses only 5 multiplications and 6 additions [31].

We estimate the benefits from BM neuron for convolutional layers in Table 5. We use the number of operations from Tables 1, 2 and their characteristics from Table 4. We also suppose that four terms in (10) are computed in parallel (twice less operation for `add` and `max` for one thread and 4 times less for `exp`).

These ratios demonstrate that for core layers inside the network with quite a large number of input channels, we can get 2.1–2.9 times fewer gates for 65 nm technology or

TABLE 4. The estimate number of gates and latency for arithmetical operations.

Op	Gates (65 nm)	Gates (16 nm)	Latency, clock cycles
add	16048	2659	3
max	1464	563	2
mul	35345	3247	4
log	154179	32189	35
exp	256965	17718	21

TABLE 5. The approximate gate number and latency ratios for standard and BM convolutional layers.

F	C	K	Gates standard/BM (65 nm)	Gates standard/BM (16 nm)	Latency standard/BM
16	1	1	0.16	0.21	0.22
16	16	1	1.14	0.89	0.80
32	1	1	0.17	0.22	0.23
32	32	1	1.64	1.20	1.02
64	1	1	0.17	0.23	0.23
64	64	1	2.11	1.45	1.18
128	1	1	0.17	0.23	0,23
128	128	1	2.45	1.62	1.28
256	1	1	0.17	0.23	0.23
256	256	1	2.67	1.72	1.34
512	1	1	0.17	0.23	0.23
512	512	1	2.80	1.77	1.37
16	1	3	1.02	0.99	0.87
16	16	3	2.50	1.64	1.29
32	1	3	1.03	1.01	0.89
32	32	3	2.70	1.73	1.34
64	1	3	1.03	1.02	0.89
64	64	3	2.81	1.78	1.37
128	1	3	1.04	1.03	0.91
128	128	3	2.87	1.81	1.39
256	1	3	1.04	1.03	0.90
256	256	3	2.9	1.82	1.39
512	1	3	1.04	1.03	0.90
512	512	3	2.92	1.83	1.40

1.4–1.8 times fewer gates for 16 nm, and 15-30% lower latency for the BM layer. As the real FPGAs and ASICs do not use separate implementation for each layer, these values are shown for demonstration only.

D. TRAINING

The BM networks need special approaches for training, because there is only one non-zero gradient element due to max operation and only one weight is updated at each iteration on the back-propagation. A lot of weights are never updated and hold inefficient values. The approach to training them was introduced in [32].

The idea is to train a neural network with classical neurons, modify desired layers sequentially from the first to the last using classical weights and fine-tune the resulting network. This approach is widely used with low-bit neural networks for lossless quantization [33], [34].

Algorithm 1: Training of BM Network

Data: Training data
Result: BM neural network
1. Train classical neural network by standard methods;
foreach *conv and fc layers* **do**
2. Replace neurons of type (3) with weights w by the BM-neurons with weights $\{v^+, v^-, v_0\}$, where:

$$v_j^+ = \begin{cases} \ln w_j, & \text{if } w_j > 0, \\ -\infty, & \text{otherwise}, \end{cases}$$
$$v_j^- = \begin{cases} \ln |w_j|, & \text{if } w_j < 0, \\ -\infty, & \text{otherwise}, \end{cases}$$
$$v_0 = w_0. \tag{16}$$

3. Fine-tune the resulting network by standard methods;

TABLE 6. Accuracy for different training methods.

Dataset	Network	Test accuracy,%	
		From scratch	Ours
MNIST	CNN_1 (conv)	98.48	98.76
MNIST	CNN_2 (conv)	98.70	99.37

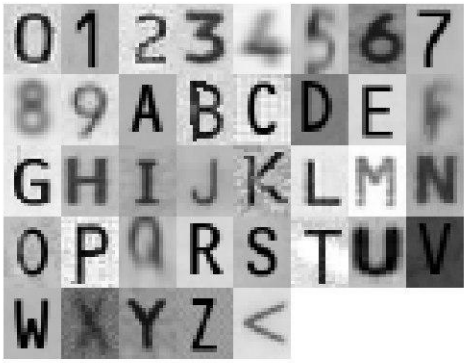

FIGURE 4. Sample images of MRZ characters.

The method of training can be summarized as:

One may perform steps 1-3 several times with different initial conditions and choose the best result.

In the Table 6 we compare this method with training from scratch (see Section IV for experimental setup and network structure). It shows that even for simple neural network architectures introduced training approach works significantly better.

IV. EXPERIMENTS

In this Section we summarize the results from [31], [32], [35] on BM network training for various problems.[1]

[1]The Tensorflow code for training is available at https://github.com/SmartEngines/bipolar-morphological-resnet

A. IMAGE CLASSIFICATION

We consider image classification problem on 3 different datasets:

1) Private dataset with symbols extracted from real documents with machine-readable zone (MRZ) [36].
   Classes: 37.
   Image size: 21 × 17 pixels, gray.
   Full dataset size: about 3, 7×10^5.
   Validation size: 2, 8×10^4.
   Test size: 9, 4×10^4.
2) MNIST [37].
   Classes: 10.
   Image size: 28 × 28 pixels, gray.
   Full dataset size: 60000.
   Validation size: 6000.
   Test size: 10000.
3) CIFAR-10 [38]
   Classes: 10.
   Image size: 32 × 32 pixels, 3 channels.
   Full dataset size: 60000.
   Validation size: 6000.
   Test size: 10000.

We test simple LeNet-5-like networks with custom architectures and ResNet-like architectures. The training is performed via layer-by-layer conversion from the Section III-D.

We use the following notation:

- conv(n, w_x, w_y) — convolutional layer with n filters of size $w_x \times w_y$;
- fc(n) — fully-connected layer with n neurons;
- maxpool(w_x, w_y) — max-pooling layer with the window of size $w_x \times w_y$;
- dropout(p) — dropout the input signals with the probability p;
- relu — rectifier activation function ReLU(x) = max (x, 0);
- softmax — standard softmax activation function.

The CNN_1 architecture is: conv1(30, 5, 5) - relu1 - dropout1(0,2) - fc1(10) - softmax1.

The CNN_2 architecture is: conv1(40, 5, 5) - relu1 - maxpool1(2, 2) - conv2(40, 5, 5) - relu2 - fc1(200) - relu3 - dropout1(0,3) - fc2(10) - softmax1.

The CNN_3 architecture is: conv1(8, 3, 3) - relu1 - conv2(30, 5, 5) - relu2 - conv3(30, 5, 5) - relu3 - dropout1(0,25) - fc1(37) - softmax1.

The CNN_4 architecture is: conv1(8, 3, 3) - relu1 - conv2(8, 5, 5) - relu2 - conv3(8, 3, 3) - relu3 - dropout1(0,25) - conv4(12, 5, 5) - relu4 - conv5(12, 3, 3) - relu5 - conv6(12, 1, 1) - relu6 - fc1(37) - softmax1.

The obtained accuracies are shown in Table 7. We can see that using BM fully-connected layers dramatically reduces accuracy, BM convolutional layers are almost lossless for CNN* and cause noticeable accuracy decrease for ResNet with 22 converted layers and almost no descend for partly-converted network.

TABLE 7. Classification accuracy for BM neural networks. The *orig acc* refers to the accuracy of standard network, the *BM part* denotes converted to BM layers and *BM acc* shows the resulting accuracy.

Dataset	Network	Orig acc, %	BM part	BM acc, %
MNIST	CNN_1	98,72	1 conv	98,76
			1 conv + fc	94,00
	CNN_2	99,45	2 conv	99,37
			2 conv + fc	97,86
	ResNet-22	99.3	22 conv	99.1
MRZ	CNN_3	99,63	2 conv	99,57
			3 conv + fc	93,38
	CNN_4	99,67	6 conv	99,61
			6 conv + fc	95,46
CIFAR-10	ResNet-22	85.3	22 conv	77.7
			16 conv	85.1

This problem can be solved by using hybrid networks with BM and standard convolutional layers. Such networks will still benefit from BM layers because normally convolutions take major time at inference stage.

B. SEMANTIC SEGMENTATION

We consider the Document Image Binarization Competition (DIBCO) 2017. It is a challenge on the binarization of historical documents.The organizers provided 86 training and 20 testing images with ideal binary markup (see example in Fig. 6a, b), and evaluation tools.

The winning solution of the DIBCO 2017 was a U-Net convolutional network [39]. It is shown in Fig. 5. It includes 10 convolutional layers with 3 × 3 kernel size and ReLU activation, and final convolution with 1×1 kernel and sigmoid activation.

The images were made grayscale and split into 128 × 128 patched. The data was divided into train and validation in the 80/20 ratio so that all patches from one image were either in the train or the validation sets. Data augmentation was performed, adding shifts, noise, contrasting, scaling, and lines. The network was trained with Adam optimizer using binary cross-entropy as a loss function and mean Intersection over Union (mIoU) to estimate binarization accuracy. To obtain binary output, output values are rounded. We use this solution as a baseline. In Fig. 6 we show the output of the U-Net in comparison to the ideal markup.

The metrics to evaluate binarization quality were presented in [40]:

- F-Measure (FM)
- pseudo F-Measure (Fps)
- PSNR
- Distance Reciprocal Distortion Metric (DRD)

The results of the fine-tuning of BM U-Net are shown in Table 8. There is no accuracy degradation in mIoU value for 2 BM layers, a small decrease for 7 layers, and a 1.9-time error increase for the whole network. Full evaluation shows that BM U-Net outperforms Otsu and Sauvola methods but is inferior to standard U-Net. However, the accuracy decrease in this problem is also increasing with the number of BM layers,

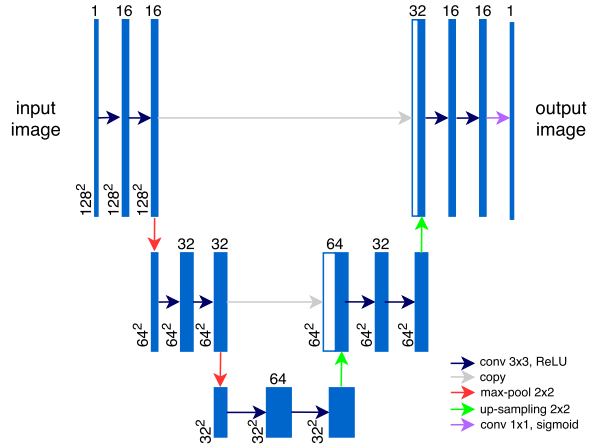

FIGURE 5. The U-Net network architecture.

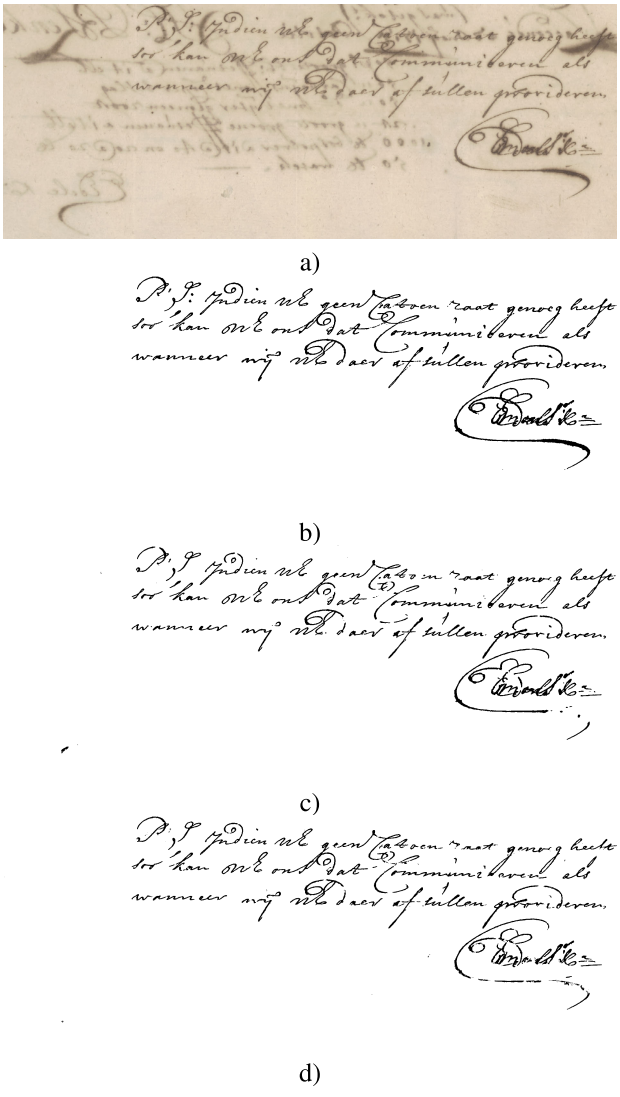

FIGURE 6. The example of binarization: a) input image, b) ideal markup, c) standard U-Net, d) BM U-Net.

so that one can choose the desired balance between quality and network complexity.

TABLE 8. Evaluation results for different methods on DIBCO 2017.

Method	mIoU	FM	Fps	PSNR	DRD
Otsu [39]	-	77.73	77.89	13.85	15.5
Sauvola [39]	-	77.11	84.10	14.25	8.85
UNet [39]	-	91.01	92.86	18.28	3.40
UNet (ours)	99.36	90.89	92.77	18.15	3.31
BM-UNet (10 layers)	98.79	85.82	88.02	17.00	5.13
BM-UNet (7 layers)	99.08	88.97	90.59	17.49	4.19
BM-UNet (2 layers)	99.33	90.35	92.56	18.00	3.50

V. RELATED WORK

Neural network usage on end devices is growing steadily. On the one hand, this leads to the development of methods for increasing the computational efficiency of neural network models, and on the other, to the development of specialized devices capable of performing the necessary calculations quickly.

Today, there are many deep learning ASICs that have a predefined architecture and cannot be customized. They are already specialized for executing modern models and supporting current trends like compact low-bit models. Such devices include Google Tensor Processing Unit [41], Intel Vision Processing Unit [42], Huawei Ascend [43], etc.

Alternatively, the use of FPGA gives developers a lot of freedom in implementing a calculator for a specific neural network model and in optimizing hardware architecture. Works on this topic are usually aimed at improving the efficiency of utilizing logic gates without modifying neural network architecture and training methods [44]–[47].

A. METHODS FOR IMPROVING NEURAL NETWORK EFFICIENCY

There are many different methods for inference time reduction. They can be roughly divided into two groups. The methods of the first group reduce the number of weight coefficients and computational operations in the network, so it works faster. These include tensor decompositions of convolutions, model compression, and neural network architectures search. Methods of the second group are oriented to hardware characteristics and replace some computational operations and structures with others that and require less time to calculate. For example, these are low-precision computations or efficiently computable neuron/layer models that replace traditional ones.

Low-precision neural networks store coefficients and use low-bit data types for computations (typically integers). Such computations are faster than in single-precision floating-point data types. Neural networks with 8-bit coefficients are widely used; there are a number of libraries developed for various device architectures [48]–[50]. There are both end-to-end systems and systems of mixed precision that use low-precision quantization, and integer computations [51]. For example, Z. Cai and N. Vasconcelos [52] note that using the same quantization method for all filters is not optimal and suggest performing a search for sufficient accuracy

quantization parameters. They show that it can significantly improve the classification results compared to uniformly quantized networks.

Also, many researchers try to reduce the bit depth further. Since standard approaches to weight quantizing are not differentiable and direct conversion does not provide sufficient accuracy, accuracy improvement methods are desired. In [53], the authors propose an approach to finding weights, which improved the accuracy for a large set of bit-widths (1, 2, 4 bits for weights and 1, 2, 4, 8, 32 bits for activations). Zhuang *et al.* [54] add a full-precision auxiliary module to the network during the training stage. They demonstrate noticeable accuracy improvement on ImageNet, CIFAR-100, and COCO datasets. In [55], a model ternary weights and a novel training method for it are proposed. Their approach allows to create a network without multiplication and control its sparseness, i.e., the number of zero weights.

Binary neural network models are also very popular because they can be implemented with fewer multiplications (or no multiplications at all). Their usage does not lead to recognition accuracy loss in many practical problems compared to models with floating-point coefficients [56]–[59]. Such solutions can provide a high inference speed but are limited to products with specially designed hardware.

At the same time, there are also works devoted to the use of low-precision floating-point data types in neural network models [60]–[62]. Their main advantage is improving the computation accuracy because the floating-point has limited relative error. Such models even allow training in an 8-bit floating-point data type, but they also need hardware support for practical use.

B. ALTERNATIVE NEURONS/LAYERS

Modifications of neural network layers are also actively developed to reduce their computational complexity or improve recognition quality. For example, in the DeepShift model [63], the authors trained the network to use a bitwise shift instead of multiplication. In [64], the authors introduce MConv layers, which compute erosion/dilatation operations in a sliding window. Since min / max operations are not differentiable, PConv, LMorph and SMorph layers with continuous structure were proposed [65], [66]. Chen *et al.* [67] modify only convolutional layers to use the L_1-norm instead of covariance. It dramatically simplifies convolutional layers; however, other layers still use multiplication. In [68], the authors proposed a knowledge distillation method to train convolutional layers with L_1-norm without accuracy loss on CIFAR-10, CIFAR-100, and ImageNet.

VI. CONCLUSION

In this paper, we summarized our work on the new BM neuron and BM network models. The BM layers are aimed to replace multiplications with additions and thus demonstrate lower hardware complexity and latency. We proposed to use BM convolutional layers in standard neural network architectures instead of conventional convolutional layers, as they are normally most computationally intensive. We showed that BM networks do not give advantages on CPUs but can significantly improve gate number and latency on FPGAs.

In this paper, we prove that the representational power of BM networks is not less than those of classical neural networks. We demonstrate and compare training approaches for the BM networks. Training from scratch shows poor resulting accuracy, so we propose the layer-by-layer technique. We used it for image classification (MNIST, CIFAR10) with LeNet-5 and ResNet architectures showing no accuracy decrease for LeNet and some accuracy degradation for ResNet, which increases with the number of BM layers. We also considered the semantic segmentation problem (DIBCO 2017) with UNet with pretty good results. However, as the accuracy is not perfect, we look forward to further research on training methods and propose BM networks with hybrid structure (some BM and some standard convolutional layers) as a quick workaround.

FUTURE WORK

The need for highly efficient devices capable of executing neural networks will only grow over time. As far as we know, two hardware-based neural network models potentially reach classical models in terms of accuracy. These are BM networks and AdderNet. AdderNet shows the recognition accuracy practically no worse than that of classical models. It is also efficient for hardware implementation [69]. BM networks also demonstrate promising results when evaluating hardware implementation, but they are still inferior in accuracy to classical networks. Thus, one of the leading research areas of BM networks should be the development of training methods. We consider approximation and additional training-based methods to be promising, as well as knowledge distillation methods. We should also consider modifying activation functions (using more simple trainable functions instead of `ln` and `exp` construction) to minimize the layer approximation error.

Another essential aspect unconsidered at the moment is the quantization of BM networks. The use of integer coefficients is important for hardware implementation and will significantly speed up and simplify it. An interesting point here will be the integer approximation of the activation function since they use exponent and logarithm in the current model. We believe that the next step towards quantizing the BM network will be simplifying the activation functions, especially when the BM layers are stacked sequentially. In this case, sequences of layers with operations `exp` - `combine` - `relu` - `ln` appear in the network, which we would like to replace with some nonlinear function of a more straightforward structure.

And finally, the concluding stage will be the creation of a prototype device for fast inference of the BM network. It is an engineering problem that can be solved based on existing FPGAs or by creating a unique, specialized device. On the one hand, the first solution is limited by the available FPGA configurations, and on the other hand, it will allow one to rapidly implement BM networks in real-world tasks.

Specialized devices will be able to demonstrate maximum performance but will require separate design and manufacturing.

REFERENCES

[1] Z. Zhang and H. Peng, "Deeper and wider siamese networks for real-time visual tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4591–4600.

[2] X.-H. Li, F. Yin, and C.-L. Liu, "Page object detection from PDF document images by deep structured prediction and supervised clustering," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Aug. 2018, pp. 3627–3632.

[3] A. Bokovoy, "Automatic control system's architecture for group of small unmanned aerial vehicles," *Informatsionnye Tekhnologii i Vychslitel'nye Sistemy*, vol. 2018, no. 1, pp. 68–77, 2018.

[4] K. Bulatov and V. V. Arlazarov, "Determining optimal frame processing strategies for real-time document recognition systems," in *Proc. ICDAR*, to be published.

[5] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," 2019, *arXiv:1907.10597*. [Online]. Available: http://arxiv.org/abs/1907.10597

[6] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*. [Online]. Available: http://arxiv.org/abs/1906.02243

[7] J. E. Savage, *Models of Computation: Exploring the Power of Computing*, 1st ed. Reading, MA, USA: Addison-Wesley, 1997.

[8] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[9] L. D. Harmon, "Studies with artificial neurons, I: Properties and functions of an artificial neuron," *Kybernetik*, vol. 1, no. 3, pp. 89–101, Dec. 1961, doi: 10.1007/BF00290179.

[10] E. R. Lewis, "The locus concept and its application to neural analogs," *IRE Trans. Bio-Med. Electron.*, vol. 10, no. 4, pp. 130–137, 1963.

[11] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[12] G. X. Ritter and P. Sussner, "An introduction to morphological neural networks," in *Proc. 13th Int. Conf. Pattern Recognit. (ICPR)*, vol. 4, 1996, pp. 709–717.

[13] G. X. Ritter, L. Iancu, and G. Urcid, "Morphological perceptrons with dendritic structure," in *Proc. 12th IEEE Int. Conf. Fuzzy Syst. (FUZZ)*, vol. 2, May 2003, pp. 1296–1301.

[14] F. Arce, E. Zamora, H. Sossa, and R. Barrón, "Differential evolution training algorithm for dendrite morphological neural networks," *Appl. Soft Comput.*, vol. 68, pp. 303–313, Jul. 2018.

[15] N. Dimitriadis and P. Maragos, "Advances in the training, pruning and enforcement of shape constraints of morphological neural networks using tropical algebra," 2020, *arXiv:2011.07643*. [Online]. Available: http://arxiv.org/abs/2011.07643

[16] J. M. Antelis, B. Gudiño-Mendoza, L. E. Falcón, G. Sanchez-Ante, and H. Sossa, "Dendrite morphological neural networks for motor task recognition from electroencephalographic signals," *Biomed. Signal Process. Control*, vol. 44, pp. 12–24, Jul. 2018.

[17] G. Hernández, E. Zamora, H. Sossa, G. Téllez, and F. Furlán, "Hybrid neural networks for big data classification," *Neurocomputing*, vol. 390, pp. 327–340, May 2020.

[18] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[19] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, Mar. 2019.

[20] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 1311–1318.

[21] S. Lu and A. Sengupta, "Exploring the connection between binary and spiking neural networks," *Frontiers Neurosci.*, vol. 14, p. 535, Jun. 2020.

[22] X. Wang, X. Lin, and X. Dang, "Supervised learning in spiking neural networks: A review of algorithms and evaluations," *Neural Netw.*, vol. 125, pp. 258–280, May 2020.

[23] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[24] G. Sampson, "Parallel distributed processing: Explorations in the microstructures of cognition," *JSTOR*, vol. 63, no. 4, pp. 871–886, 1987.

[25] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 826–834, Sep. 1983.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[28] A. Fog. (2017). Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, AMD and via CPUs. Technical University of Denmark. [Online]. Available: http://www.agner.org/optimize/instruction_tables.pdf

[29] *Cortex-A57 Software Optimization Guide*. Accessed: Jul. 7, 2021. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.uan0015b/Cortex_A57_Software_Optimization_Guide_external.pdf

[30] *Reference Implementations for Intel Architecture Approximation Instructions VRCP14, VRSQRT14, VRCP28, VRSQRT28, and VEXP2*. Accessed: Jul. 7, 2021. [Online]. Available: https://software.intel.com/en-us/articles/reference-implementations-for-IA-approximation-instructions-vrcp14-vrsqrt14-vrcp28-vrsqrt28-vexp2

[31] E. Limonova, D. Alfonso, D. Nikolaev, and V. V. Arlazarov, "ResNet-like architecture with low hardware requirements," 2020, *arXiv:2009.07190*. [Online]. Available: http://arxiv.org/abs/2009.07190

[32] E. Limonova, D. Matveev, D. Nikolaev, and V. V. Arlazarov, "Bipolar morphological neural networks: Convolution without multiplication," *Proc. SPIE*, vol. 11433, pp. 1–8, Jan. 2020, doi: 10.1117/12.2559299.

[33] D. Ilin, E. Limonova, V. Arlazarov, and D. Nikolaev, "Fast integer approximations in convolutional neural networks using layer-by-layer training," *Proc. SPIE*, vol. 10341, Mar. 2017, Art. no. 103410Q, doi: 10.1117/12.2268722.

[34] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," Feb. 2017, *arXiv:1702.03044*. [Online]. Available: https://arxiv.org/abs/1702.03044

[35] E. Limonova, D. Nikolaev, and V. V. Arlazarov, "Bipolar morphological u-net for document binarization," *Proc. SPIE*, vol. 11605, Jan. 2021, Art. no. 116050P, doi: 10.1117/12.2587174.

[36] *Machine-Readable Passport*. Accessed: Jul. 7, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Machine-readable_passport

[37] *THE MNIST DATABASE of Handwritten Digits*. Accessed: Jul. 7, 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[38] *CIFAR-10 Dataset*. Accessed: Jul. 7, 2021. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[39] P. V. Bezmaternykh, D. A. Ilin, and D. P. Nikolaev, "U-Net-bin: Hacking the document image binarization contest," *Comput. Opt.*, vol. 43, no. 5, pp. 825–832, Oct. 2019, doi: 10.18287/2412-6179-2019-43-5-825-832.

[40] I. Pratikakis, K. Zagoris, G. Barlas, and B. Gatos, "ICDAR2017 competition on document image binarization (DIBCO 2017)," in *Proc. 14th IAPR Int. Conf. Document Anal. Recognit. (ICDAR)*, Nov. 2017, pp. 1395–1403.

[41] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, May 2018.

[42] *Intel Movidius™ Vision Processing Units*. Accessed: Jul. 7, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/movidius-vpu.html

[43] C. Yeo, D. M. H. Kee, X. Y. Mo, H. E. Ang, S. M. Chua, S. Agnihotri, and S. Pandey, "Technology advancement and growth: A case study of Huawei," *J. Community Develop. Asia*, vol. 3, no. 1, pp. 82–91, Jan. 2020.

[44] K. Khalil, O. Eldash, B. Dey, A. Kumar, and M. Bayoumi, "Architecture of a novel low-cost hardware neural network," in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2020, pp. 1060–1063.

[45] D. Wang, K. Xu, J. Guo, and S. Ghiasi, "DSP-efficient hardware acceleration of convolutional neural network inference on FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4867–4880, Dec. 2020.

[46] X. Jun, "FPGA deep learning acceleration based on convolutional neural network," 2020, *arXiv:2012.03672*. [Online]. Available: http://arxiv.org/abs/2012.03672

[47] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, 2019.

[48] *gemmlowp: A Small Self-Contained Low-Precision GEMM Library*. Accessed: Jul. 7, 2021. [Online]. Available: https://github.com/google/gemmlowp

[49] (2020). *The Ruy Matrix Multiplication Library*. [Online]. Available: https://github.com/google/ruy
[50] M. Dukhan, Y. Wu, and H. Lu. (2018). *QNNPACK: Open Source Library for Optimized Mobile Deep Learning*. [Online]. Available: https://code.fb.com/ml-applications/qnnpack/
[51] Y. Yao, B. Dong, Y. Li, W. Yang, and H. Zhu, "Efficient implementation of convolutional neural networks with end to end integer-only dataflow," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2019, pp. 1780–1785.
[52] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 2349–2358.
[53] Z. Yang, Y. Wang, K. Han, C. Xu, C. Xu, D. Tao, and C. Xu, "Searching for low-bit weights in quantized neural networks," 2020, *arXiv:2009.08695*. [Online]. Available: http://arxiv.org/abs/2009.08695
[54] B. Zhuang, L. Liu, M. Tan, C. Shen, and I. Reid, "Training quantized neural networks with a full-precision auxiliary module," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 1488–1497.
[55] X. Deng and Z. Zhang, "An embarrassingly simple approach to training ternary weight networks," 2020, *arXiv:2011.00580*. [Online]. Available: http://arxiv.org/abs/2011.00580
[56] J. Li, Y. Wang, B. Liu, Y. Han, and X. Li, "Simulate-the-hardware: Training accurate binarized neural networks for low-precision neural accelerators," in *Proc. 24th Asia South Pacific Design Autom. Conf.*, Jan. 2019, pp. 323–328.
[57] M. D. McDonnell, H. Mostafa, R. Wang, and A. Schaik, "Single-bit-per-weight deep convolutional neural networks without batch-normalization layers for embedded systems," in *Proc. 4th Asia–Pacific Conf. Intell. Robot Syst. (ACIRS)*, Jul. 2019, pp. 197–204.
[58] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song, "Forward and backward information retention for accurate binary neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 2250–2259.
[59] Y. Li, Y. Bao, and W. Chen, "Fixed-sign binary neural network: An efficient design of neural network for Internet-of-Things devices," *IEEE Access*, vol. 8, pp. 164858–164863, 2020.
[60] L. Cambier, A. Bhiwandiwalla, T. Gong, M. Nekuii, O. H Elibol, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," 2020, *arXiv:2001.05674*. [Online]. Available: http://arxiv.org/abs/2001.05674
[61] J. Johnson, "Rethinking floating point for deep learning," 2018, *arXiv:1811.01721*. [Online]. Available: http://arxiv.org/abs/1811.01721
[62] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 4900–4909.
[63] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "DeepShift: Towards multiplication-less neural networks," 2019, *arXiv:1905.13298*. [Online]. Available: http://arxiv.org/abs/1905.13298
[64] D. Mellouli, T. M. Hamdani, J. J. Sanchez-Medina, M. B. Ayed, and A. M. Alimi, "Morphological convolutional neural network architecture for digit recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2876–2885, Sep. 2019.
[65] J. Masci, J. Angulo, and J. Schmidhuber, "A learning framework for morphological operators using counter–harmonic mean," in *Proc. Int. Symp. Math. Morphol. Appl. Signal Image Process.* Berlin, Germany: Springer, 2013, pp. 329–340.
[66] A. Kirszenberg, G. Tochon, E. Puybareau, and J. Angulo, "Going beyond p-convolutions to learn grayscale morphological operators," 2021, *arXiv:2102.10038*. [Online]. Available: http://arxiv.org/abs/2102.10038
[67] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "AdderNet: Do we really need multiplications in deep learning?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 1468–1477.
[68] Y. Xu, C. Xu, X. Chen, W. Zhang, C. Xu, and Y. Wang, "Kernel based progressive distillation for adder neural networks," 2020, *arXiv:2009.13044*. [Online]. Available: http://arxiv.org/abs/2009.13044
[69] Y. Wang, M. Huang, K. Han, H. Chen, W. Zhang, C. Xu, and D. Tao, "AdderNet and its minimalist hardware design for energy-efficient artificial intelligence," 2021, *arXiv:2101.10015*. [Online]. Available: http://arxiv.org/abs/2101.10015

ELENA E. LIMONOVA (Member, IEEE) was born in Dolgoprudny, Moscow, Russia. She received the master's degree in physics, mathematics, and computer science from the Moscow Institute of Physics and Technology, in 2017. She is currently pursuing the Ph.D. degree with the FRC CSC RAS. Since 2016, she has been a Programmer and a Technician with Smart Engines Service LLC. Her research interests include neural network compression and image recognition on mobile devices.

DANIIL M. ALFONSO was born in Moscow, Russia. He received the master's degree in physics, mathematics, and computer science from the Moscow Institute of Physics and Technology, in 2015. Since 2012, he has been a Hardware Engineer with JSC MCST, being involved in development of microprocessors Elbrus-8C and Elbrus-16C. His research interests include built-in self-test systems and arithmetic hardware engines.

DMITRY P. NIKOLAEV (Member, IEEE) was born in Moscow, Russia, in 1978. He received the master's degree in physics and the Ph.D. degree in computer science from Moscow State University, Moscow, in 2000 and 2004, respectively. Since 2007, he has been the Head of the Vision Systems Laboratory, Institute for Information Transmission Problems, Russian Academy of Sciences, Moscow. Since 2016, he has been the CTO of Smart Engines Service LLC., Moscow. Since 2016, he has also been an Associate Professor with the Moscow Institute of Physics and Technology (State University), Moscow, teaching the "Image Processing and Analysis" course. He is an author of over 220 articles and six patents. His research interest includes computer vision with primary application to color image understanding.

VLADIMIR V. ARLAZAROV (Member, IEEE) was born in Moscow, Russia, in 1976. He received a Specialist degree in applied mathematics from the Moscow Institute of Steel and Alloys, in 1999, and the Ph.D. degree in computer science, in 2005. Since 1999, he has been working with the Institute for Systems Analysis of Russian Academy of Sciences (currently Federal Research Center "Computer Science and Control," Russian Academy of Sciences), Moscow, as a Researcher, Senior Researcher, and the Head of the Laboratory. Since 2012, he has been an Associate Professor with the Moscow Institute of Physics and Technology (State University), Moscow. Since 2016, he has been the General Director of Smart Engines Service LLC., Moscow. Since 2018, he has been a Senior Researcher with the Institute for Information Transmission Problems, Russian Academy of Sciences. He has published more than 90 articles and authored seven patents. His research interests include computer vision and document analysis systems.