Method for fast classification of MNIST digits on Arduino UNO board using LogNNet and linear congruent generator

Y A Izotov, A A Velichko* and P P Boriskov
Institute of Physics and Technology, Petrozavodsk State University, 33 Lenin str., 185910, Petrozavodsk, Russia
*E-mail: velichko@petrsu.ru

Abstract. The paper presents a method for forming a reservoir of a neural network LogNNet using a linear congruent pseudo-random number generator. This method made it possible to reduce the MNIST handwritten digit recognition time on the low-memory Arduino Uno board to 0.28 s for the LogNNet 784:20:10 configurations, with a classification accuracy of ~ 82%. It was found that the computations with integers gives an increase in the speed of the algorithm by more than 2 times in comparison with the algorithm using the real type when generating a chaotic time series. The developed method can be used to accelerate the calculations of edge devices in the field of "Internet of Things", for example, for mobile medical devices, autonomous vehicle control systems and bionic suit control.

1. Introduction
Currently, neural networks are being actively implemented in many branches of science and technology. Neural networks intelligently process large amounts of information [1], identify threats in the network infrastructure [2], are used in clinical decision support systems [3,4], as well as in mobile medical devices [5], autonomous transport control systems [6] and bionic suits (soft wearable exosuit) [7,8]. Artificial intelligence based on neural networks finds application in the concept of "Internet of Things" (IoT) [9,10] not only at the upper levels of ecosystems (servers, cloud services), but also at the local level of IoT for computing on edge devices (microcontrollers, sensors) [11]. Limited resources of edge devices, their low computing power create difficulties for the implementation of artificial intelligence on low-level platforms and require the development of special architectures and algorithms of neural networks.

In previous works [12,13], we developed the architecture of the LogNNet neural network and tested its operation on the Arduino UNO board with a low ~ 2kB RAM. LogNNet performed well on MNIST handwritten digit recognition with ~ 82% accuracy. A distinctive feature of the LogNNet architecture is the presence of a reservoir where the input information is transformed due to the influence of a recurrent chaotic mapping. Such a chaotic transformation transforms information from one multidimensional space into another, after which an efficient classification of the result by the output linear classifier is possible. The neural network LogNNet is a feedforward network, where signals propagate exclusively from input to output, has a simple architecture that is easily adaptable to various platforms.

The output values of the reservoir are calculated in the course of the program's operation and this process takes up a small amount of RAM memory, but this requires significant computational and time
expenditures of the processor to calculate the sums and products of floating-point values. For the LogNNet 784:20:10 configuration, the recognition time for one image on the Arduino UNO was about 7 s [13]. In this regard, the transition from a chaotic logistic mapping, which has a time series of real numbers, to a chaotic series of integers will lead to a significant acceleration in the calculation of the output values of the reservoir.

Generation of pseudo-random integers is a widely studied scientific problem, for example, in the field of information encryption [14]. There are several basic methods for generating the Mersenne Twister [15], Linear Congruential Generator [16], Xorshift [17]. The Mersenne Twister method is the most widely used for generating pseudo-random numbers, but in terms of implementation time and simplicity it is inferior to the Linear Congruential Generators method. The Xorshift method is an extremely fast number generator and its application may be the subject of future research. There are other types of generators adapted for digital platforms, for example, one-dimensional discrete-space chaotic map based on the multiplication of integer numbers and circular shift [18], composition of permutations [14], logistic map over integer [19].

The transition from real to integers in the calculations of neural networks was used in a number of works to reduce the consumed RAM: algorithms Bonsai [20], ProtoNN [21], CNN [22], FastGRNN [23], Spectral-RNN [24], and NeuralNet Pruning [25]. However, the use of a pseudo-random integer generator for calculating neural networks is used for the first time in this work, in addition, this approach is the basis for saving RAM in the LogNNet network.

In the present study, a method for generating pseudo-random integers is proposed to form a chaotic reservoir of the LogNNet network, and a study of the speed of LogNNet for various reservoir configurations is carried out. This paper has the following structure: Section 2 describes LogNNet structure, architecture optimization techniques, formulas used. Section 3 describes the implementation features of the program code on the Arduino platform. Section 4 describes results for measuring accuracy and characteristic times. In the Conclusions, a general description of the study and its scientific significance is given.

2. Architecture of LogNNet

The architecture of the neural network LogNNet is shown in figure 1. The architecture parameters are indicated as LogNNet S:P:M, where S = 784 is the size of the input data array Y, the value of P determines the number of neurons in the hidden layer Sh, and the number M = 10 is the number of neurons in the output layer Sout.

![Figure 1. LogNNet neural network architecture.](image)

The input of the neural network LogNNet is a handwritten image with a size of 28 × 28 pixels from the MNIST database [26]. The image is presented in grayscale (8 bits), black corresponds to a pixel value of 255, and white corresponds to 0. The MNIST base contains a training set consisting of 60,000 images and a test set consisting of 10,000 images. The digit image is converted to a linear array Y using the T-pattern transformation [12] (figure 1), normalized to 1 by dividing the pixel value by 255.
The zero element of the $Y[0] = 1$ array is the bias neuron. Vector $Y$ is fed to the reservoir, which converts information according to formula (1) and then the linear classifier classifies the result according to formula (2).

\[ S_h = f_h(W_1 \cdot Y), \]
\[ S_{out} = f_{out}(W_2 \cdot S_h). \]

In formulas (1) and (2) $S_h$ is the vector of neurons of the hidden layer located in the reservoir, $S_{out}$ is the vector of neurons of the output layer, $Y$ is the vector of input data, $W_1$, $W_2$ are the matrix of weight coefficients, $f_h$ is the identical activation function, with subsequent normalization of the values in the range from -0.5 to 0.5, and $f_{out}$ is the logistic activation function. The maximum, minimum and weighted average values of $S_h$ are normalized values of the function $f_h$ and are calculated from the first 1000 elements of the training set [12].

The weight coefficients $W_1$ are set using a chaotic mapping, the numbering of the elements of the matrix $W_1[j,i]$ has the limits $j=1…P$, $i=0…784$.

The weight coefficients $W_2$ are trained by the backpropagation method using the training many MNIST images.

Testing the accuracy of LogNNNet recognition is performed by running the test set MNIST through the neural network.

2.1. Optimization of the LogNNNet architecture using the logistic equation

In our previous study [13], the reservoir matrix $W_1$ was filled column by columns with a time series of the logistic mapping, the initial values for which were determined by the sine function according to the formula:

\[ W_1[1,i] = A \cdot \sin\left(\frac{i \pi}{784} \right), \]
\[ W_1[j,i] = 1 - r \cdot (W_1[j-1,i])^2, \text{ for } j > 1 \]

where $r$, $A$ and $B$ are constants.

Formula (3) has a drawback that the order of calculation of the elements of the matrix $W_1[j,i]$ does not correspond to the order of their use in formula (1). This, in turn, significantly increases the calculation time for formula (1).

To eliminate this drawback, we proposed to use row-by-row filling of the matrix $W_1$ with a time series based on the logistic mapping:

\[ x_{n+1} = 1 - R \cdot (x_n)^2, \]
\[ x_1 = V \]

where $R$ and $V$ are constants, $V$ defines the initial value of the time series.

Matrix $W_1$ is filled row by row as shown in figure 2a, according to the formula

\[ n = (j-1) \cdot 785 + i + 1, \]
\[ W_1[j,i] = x_n \]
The number of elements $N$ in the time series is determined as $N=P \times 785$.

The selection of constants $R$ and $V$ is carried out by the optimization method of a swarm of particles, where the fitness function is the LogNNet recognition accuracy function.

In operation the values of the constants $R=1.99999$ and $V=0.95861558034584$ were used.

The trained neural network was programmed on the Arduino UNO board.

2.2. Optimization of the LogNNet architecture using the congruent generator

Reducing the filling time of the matrix $W_1$ and the calculation time of formula (1) can be achieved if, instead of the logistic equation (4) generating chaotic floating-point numbers, a pseudo-random integer number generator is used. In this work, we propose to use a linear congruent integer generator:

$$\begin{align*}
  x_{n+1} &= (K \cdot x_n + D) \mod L \\
  x_i &= C
\end{align*}$$

(6)

where $K$, $D$, $L$ and $C$ are constants of integer type, $n$ is the numbering of the elements of the series ($n>0$), and the function mod is the remainder of the integer division. Constant $C$ defines the initial value of the series $x_n$.

The chaotic integers generated by formula (6) range from $-(L-1)$ to $+(L-1)$.

Matrix $W_1$ is filled with time series $x_n$ line by line, as shown in figure 2b, normalized to 1 according to the formula:

$$\begin{align*}
  n &= (j-1) \cdot 785 + i + 1 \\
  W_{1}[j,i] &= \frac{x_n}{L}
\end{align*}$$

(7)

The maximum number of elements $N$ of the time series $x_n$ is $N=P \times 785$.

The selection of constants $K$, $D$, $L$ and $C$ was carried out by the optimization method of a swarm of particles, where the fitness function is the LogNNet recognition accuracy function.

The values of the constants $K=2$, $D=0$, $L=5534$ and $C=18$ were used in the work.

The trained neural network was programmed on the Arduino UNO board.

---

**Figure 2.** Scheme of row-by-row filling of matrix $W_1$ using logistic mapping (a) and congruent generator (b).
3. Implementation of the LogNNet network on the Arduino UNO board

In this study, four architectures were used: LogNNet 784:20:10, LogNNet 784:18:10, LogNNet 784:15:10 and LogNNet 784:10:10, with parameters similar to work [13], which was done in order to compare the results. The maximum value of $P = 20$ was determined by the RAM limit of the Arduino UNO board. The program code is presented in the appendix in figure 4; it differs from the code from [13] by the “Hidden_Layer_Calculating” procedure, which is responsible for the reservoir function. In this procedure, the elements of the matrix $W_1$ are formed and multiplied with the vector $Y$.

Three versions of the “Hidden_Layer_Calculating” procedure were compared: Version 1 - filling in $W_1$ according to formula (3), as in [13]; Version 2 - filling $W_1$ line by line with logistic map according to formulas (4-5); Version 3 - filling $W_1$ line by line with a congruent generator according to formulas (6-7).

The code corresponding to each version is shown in figure 3.

Version 1 has three loops and its calculations are not optimized. Version 2 has code optimization, implementing line-by-line filling of the matrix and the removal of the normalization operation $Y$ outside the inner loop. Version 3 differs from Version 2 by switching to integer calculations using the congruent generator, with the normalization operation being moved outside the inner loop. Performing mathematical operations with the “int” type takes significantly less time than similar mathematical operations with the “float” type. When executing the congruent generator function (6) on Arduino, it is necessary to take into account the limits of the range of the integer type “int” (from -32768 to 32767), and avoid overflow of variables of the integer type. Overflow of variables can lead to a difference in the result from model calculations on a computer. The $W_2$ matrix is trained on a high-performance computer in a model program written in the Delphi language. When optimizing the constants $K$, $D$, $L$, and $C$ in the model program, we used artificial checks for the time series to go beyond the permissible limits of integers.

![Code snippets for the "Hidden_Layer_Calculating" procedure](image)

Figure 3. Code of the "Hidden_Layer_Calculating" procedure for Arduino UNO Version 1 (a), Version 2 (b) with a logistic equation, and Version 3 (c) with a congruent generator.
It should be noted that simulating Version 2 on a computer is faced with the difficulty of distinguishing between "float" type on Arduino with 6 decimal places and "float" type on Delphi compiler with 15 decimal places, this leads to long time series. Calculated on different platforms start to differ significantly after the 20th element. Therefore, in the Delphi model program, the $W_1$ matrix was filled with a time series generated on the Arduino. Solving the problem of mismatching long time series of "float" type on different platforms is a topic for further research. For time series of an integer type, this problem does not arise, which is an additional advantage of using pseudo-random integer generators.

All three versions of the "Hidden_Layer_Calculating" procedure provide low RAM consumption using only a few auxiliary variables, and provide significant memory savings compared to the code when the $W_1$ array is stored entirely in RAM.

Testing the classification accuracy of the MNIST test base by the executable program on the Arduino Uno board was carried out through the data exchange module between the computer and the COM port of the board. Images of 10,000 handwritten MNIST digits were sequentially transmitted through the COM port to the ATmega 328P microcontroller, and the Digit classification result was transmitted back to the computer (see the program code in Appendix A).

4. Results and discussion

Table 1 shows the results of the MNIST-10 image recognition accuracy on the Arduino Uno board, obtained using the LogNNet network with various versions of the "Hidden_Layer_Calculating" procedure. It can be seen that Versions 1 and 3 have a similar result in terms of accuracy, with a maximum value of $\sim 82\%$ for the LogNNet 784:20:10 configuration. Version 2 has a lower recognition accuracy, which may be due to a smaller number of variable parameters of the logistic mapping and its properties. Version 1 has 3 variable parameters ($A, B, r$), Version 2 has 2 parameters ($V, R$) and Version 3 has 4 parameters ($K, D, L, C$).

The time spent on the execution of the "Hidden_Layer_Calculating" procedure is shown in Table 2. The time was counted on the Arduino board through the "millis" function, which returns the value of the time interval in milliseconds. As you can see from Table 2, the "Hidden_Layer_Calculating" procedure based on a linear congruential generator (Version 3) is performed with the least amount of time. Using integer variables in Version 3 allows you to speed up calculations relative to Version 2 floating point by more than 2 times. A significant difference in execution time of $\sim 25$ times between Versions 1 and 3 is achieved by optimizing the code for calculating the values of the neurons of the hidden layer $S_h$. Version 3 first calculates the sum of the products ($\text{long}W_1 Y[i]$) in integer format, and then normalizes by dividing by the product of the constants $L \cdot 255L$. The operation with the "long" type are performed much faster than with the "float" type. The constant $L$ is a parameter in Equation 6, and the constant 255L represents the number 255 in the "long" type representation. The use of the "long" type is also necessary to avoid overflow, since the result of $L \cdot 255L$ exceeds the limits of the "int" type.

Table 3 shows the recognition times of one image by the LogNNet network for different versions of the "Hidden_Layer_Calculating" procedure. The values of table 3 differ from the corresponding values of Table 2 by a small-time interval of the order of 10-15 ms, spent on normalization (Hidden_Layer_Normalization procedure) and processing of the output classifier of the neural network (Output_Layer function). Obviously, most of the time spent on recognizing one image falls on the "Hidden_Layer_Calculating" procedure.

Table 1. Accuracy of MNIST recognition by the LogNNet network for various versions of the procedure "Hidden_Layer_Calculating.

| Network configuration | Version 1 | Version 2 | Version 3 |
|-----------------------|-----------|-----------|-----------|
| LogNNet 784:20:10     | 82.03%    | 65.04%    | 82.18%    |
| LogNNet 784:18:10     | 80.45%    | 63.06%    | 79.79%    |
LogNNet 784:15:10  76.94%  59.80%  75.96%
LogNNet 784:10:10  70.01%  52.30%  68.92%

Table 2. Working hours of different versions of the Hidden_Layer_Calculating” procedure.

| Network configuration | Version 1 | Version 2 | Version 3 |
|-----------------------|-----------|-----------|-----------|
| LogNNet 784:20:10     | 7.096 s   | 0.592 s   | 0.269 s   |
| LogNNet 784:18:10     | 5.997 s   | 0.533 s   | 0.241 s   |
| LogNNet 784:15:10     | 4.049 s   | 0.444 s   | 0.201 s   |
| LogNNet 784:10:10     | 2.054 s   | 0.296 s   | 0.133 s   |

Table 3. Recognition time of one image by the LogNNet network when using different versions of the "Hidden_Layer_Calculating” procedure.

| Network configuration | Version 1 | Version 2 | Version 3 |
|-----------------------|-----------|-----------|-----------|
| LogNNet 784:20:10     | 7.111 s   | 0.607 s   | 0.284 s   |
| LogNNet 784:18:10     | 6.013 s   | 0.547 s   | 0.256 s   |
| LogNNet 784:15:10     | 4.062 s   | 0.457 s   | 0.213 s   |
| LogNNet 784:10:10     | 2.065 s   | 0.305 s   | 0.143 s   |

An increase in accuracy can be achieved by increasing the number of neurons in the hidden layer or using several hidden layers, but the use of both methods requires an increase in RAM memory, and is a subject for further research.

The results of the work showed an increase in the speed of the LogNNet neural network by more than two times when changing from chaotic map with elements of "float" type to type "int", and more than 25 times when optimizing reservoir computation. In Version 3, the digit image is processed in a fraction of a second. The high computing speed of LogNNet can be in demand when processing medical data [4] in medical decision support systems, as well as for new applications, for example, in autonomous motion systems or when creating intelligent miniature wearable devices. When creating autonomous transport systems, the letter image recognition algorithm can be used to recognize road signs. In papers [7,8], a machine learning algorithm quickly determined the best parameters for controlling a soft exosuit in order to minimize the amount of energy that a person spends while walking. The neural network used respiration rate measurements as input parameters, and at the output indicated to the exosuit when and where to apply the auxiliary force. The maximum operating frequencies of the registered dynamics are within 1 Hz, and correspond to the operating speed of LogNNet on the Arduino. Solving exoskeleton control problems using LogNNet may be the subject of further research.

5. Conclusions
In the course of the work, three versions of the procedure for generating a chaotic reservoir of LogNNet networks were investigated. All three versions of the procedure were tested on an Arduino Uno board with a small amount of RAM.

The use of the linear congruent generator method allowed us to speed up the execution of the entire algorithm by more than 25 times relative to the version of the algorithm presented in [13], with a classification value of ~ 82% for the LogNNet 784:20:10 configuration and the RAM used no more than 2kB. It was found that the transition to computations with integers gives an increase in the speed
of the algorithm by more than 2 times in comparison with the algorithm using the real type when generating a chaotic time series.

The use of the congruent generator method in LogNNet expands the range of research areas devoted to the implementation of artificial intelligence on peripheral devices "Internet of Things", contributes to the miniaturization of the technological base and increases the performance of edge devices. With the appropriate training, LogNNet can be used for a wide range of tasks, on most digital-to-analog platforms. The executable program has a compact form, and can be used for educational purposes when working with boards of the Arduino family. With appropriate training, the presented LogNNet algorithm can be adapted for wide practical applications, for example, for mobile medical devices, autonomous vehicle control systems and bionic suit control.

Acknowledgment

This work is implemented in Petrozavodsk State University (PetrSU) with financial support by the Ministry of Science and Higher Education of Russia within Agreement no. 075-15-2021-1007 on the topic "Software and hardware methods of sensorics and machine perception for robotic systems with autonomous movement".

In part of data computing infrastructure, this study was performed using the Unique Scientific Unit (UNU)-Multicomponent software and hardware system for automated collection, storage, markup of research and clinical biomedical data, their unification and analysis based on Data Center with Artificial Intelligence technologies (UNU for support of medical decision-making) (reg. number: 2075518).

Appendix A

(a)

```c
1 #include "LogNet.h"
2
3 #define PI 3.141592653
4 #define MAX 100
5 #define MIN 0
6
7 void Hidden_Layer_Calculating(double *Y) {
8     int Wl = Cl;
9     long Sh_f = 0;
10    Sh[0] = 1;
11    for (int j = 1; j <= P; j++) {
12        Sh[j] = Sh[j-1];
13        Wl = Wl + Dl*Y[j];
14        Sh_f = Sh_f + (long)(Wl + Y[j-1]);
15    }
16    Sh[1] = (float)Sh_f / (L+205d);
17}
```

(b)

```c
31    SOUR[1] = 0;
32    for (int i = 0; i <= P; i++)
33        SOUR[1] = SOUR[1] + Sh[i];
34
35    byte digit = 0;
36    for (int i = 0; i < MAX; i++) {
37        if (Sour[1] > SOUR[1] + i) {
38            digit = i;
39            return digit;
40        }
41    }
42}
43
44 void setup() {
45    Serial.begin(115200);
46}
47
48 void loop() {
49    if (Serial.available() > 0) {
50        Y[i] = Serial.read();
51    } else if (i < 51) {
52        i = 0;
53    }
54    Hidden_Layer_Calculating(Y);
55    float Sh0 = Hidden_Layer_Normalization();
56    float Output[51];
57    for (int j = 0; j < MAX; j++) {
58        Outpu
```

Figure 4. Executable program for LogNNet network on Arduino Uno board using linear congruential generator.
Figure 5. LogNNet.h library source code for LogNet 784:20:10 configuration using linear congruential generator.

The source code version is available in the GitHub repository: https://github.com/izotov93/LogNNet_LCG

References

[1] Baliarsingh S K, Vipsita S, Gandomi A H, Panda A, Bakshi S and Ramasubbareddy S 2020 Analysis of high-dimensional genomic data using MapReduce based probabilistic neural network Comput. Methods Programs Biomed. 195 105625

[2] Sarker I H 2021 Deep cybersecurity: a comprehensive overview from neural network and deep learning perspective SN Comput. Sci. 2 154

[3] Kwan J L, Lo L, Ferguson J, Goldberg H, Diaz-Martinez J P, Tomlinson G, Grimshaw J M and Shojania K G 2020 Computerised clinical decision support systems and absolute improvements in care: meta-analysis of controlled clinical trials BMJ 370 m3216

[4] Velichko A 2021 A Method for medical data analysis using the LogNNet for clinical decision support systems and edge computing in healthcare Sensors 21

[5] Obermeyer Z and Emanuel E J 2016 Predicting the future — big data, machine learning, and clinical medicine N. Engl. J. Med. 375 1216–9

[6] Erdmann J G, Hummela V, Leipzig K von and Schuhmacher J 2020 Development and implementation of an autonomous control system for target-optimised use of intralogistics transport systems in the Learning Factory Werk 150 at Reutlingen University Procedia Manuf. 45 405–10

[7] Ye D, Myunghkee K, Scott K and J. W C 2018 Human-in-the-loop optimization of hip assistance with a soft exosuit during walking Sci. Robot. 3 eaar5438

[8] Xiloyannis M, Annese E, Canesi M, Kodiyan A, Bicchi A, Micera S, Ajoudani A and Masia L
2019 Design and validation of a modular one-to-many actuator for a soft wearable exosuit Front. Neurorobot. 13 39

[9] Hamdan S, Ayyash M and Almajali S 2020 Edge-computing architectures for Internet of Things Applications: A Survey Sensors 20

[10] Korzun D, Balandina E, Kashevnik A, Balandin S and Viola F 2019 Ambient Intelligence Services in IoT Environments (IGI Global)

[11] Perminov V, Ermakov V and Korzun D 2020 Fault Diagnosis for Industrial Rotary Machinery based on Edge Computing and Neural Networking UBICOMM 2020, Proc. of the 14th Int. Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (Nice, France) pp 1–6

[12] Velichko A 2020 Neural network for low-memory IoT devices and MNIST image recognition using kernels based on logistic map Electronics 9 1432

[13] Izotov Y A, Velichko A A, Ivshin A A and Novitskiy R E 2021 Recognition of handwritten MNIST digits on low-memory 2 Kb RAM Arduino board using LogNNet reservoir neural network IOP Conf. Ser. Mater. Sci. Eng. 1155 12056

[14] Lambić D 2015 A new discrete chaotic map based on the composition of permutations Chaos, Solitons & Fractals 78 245–8

[15] Abdelfatah R I 2020 A color image authenticated encryption using conic curve and Mersenne twister Multimed. Tools Appl. 79 24731–56

[16] de Oliveira T and Marranghello N 2011 Design of a reconfigurable pseudorandom number generator for use in intelligent systems Neurocomputing 74 1510–9

[17] Billaudelle S, Cramer B, Petrovici M A, Schreiber K, Kappel D, Schemmel J and Meier K 2021 Structural plasticity on an accelerated analog neuromorphic hardware system Neural Networks 133 11–20

[18] Lambić D 2020 A new discrete-space chaotic map based on the multiplication of integer numbers and its application in S-box design Nonlinear Dyn. 100 699–711

[19] MIYAZAKI T, ARAKI S and UEHARA S 2010 Some Properties of Logistic Maps over Integers IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E93.A 2258–65

[20] Kumar A, Goyal S and Varma M 2017 Resource-efficient machine learning in 2 KB RAM for the Internet of things Proc. of the 34th Int. Conf. on Machine Learning vol 70 ICML’17 (JMLR.org) pp 1935–44

[21] Gupta C, Suggala A S, Goyal A, Simhadri H V, Paranjape B, Kumar A, Goyal S, Udupa R, Varma M and Jain P 2017 ProtoNN: Compressed and accurate kNN for resource-scarce devices Proc. of the 34th Int. Conf. on Machine Learning Proceedings of Machine Learning Research vol 70, ed D Precup and Y W Teh (International Convention Centre, Sydney, Australia: PMLR) pp 1331–40

[22] Gural A and Murmann B 2019 Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications Proc. of the 36th Int. Conf. on Machine Learning Proceedings of Machine Learning Research vol 97, ed K Chaudhuri and R Salakhutdinov (Long Beach, California, USA: PMLR) pp 2515–24

[23] Kusupati A, Singh M, Bhatta K, Kumar A, Jain P and Varma M 2018 FastGRNN: a fast, accurate, stable and tiny kilobyte sized gated recurrent neural network Advances in Neural Information Processing Systems

[24] Zhang J, Lei Q and Dhillon I 2018 Stabilizing gradients for deep neural networks via efficient SVD parameterization Proc. of the 35th Int. Conf. on Machine Learning Proceedings of Machine Learning Research vol 80, ed J Dy and A Krause (Stockholmsmässan, Stockholm Sweden: PMLR) pp 5806–14

[25] Han S, Mao H and Dally W 2016 Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding CoRR abs/1510.0

[26] Lecun Y, Bottou L, Bengio Y and Haffner P 1998 Gradient-based learning applied to document recognition Proc. IEEE 86 2278–324