In this paper, the object of the research is the implementation of artificial neural networks (ANN) on FPGA. The problem to be solved is the construction of a mathematical model used to determine the compliance of FPGA computing resources with the requirements of neural networks, depending on their type, structure, and size. The number of its LUT (Look-up table – the basic FPGA structure that performs logical operations) is considered as a computing resource of the FPGA.

The search for the required mathematical model was carried out using experimental measurements of the required number of LUTs for the implementation on the FPGA of the following types of ANNs:

– MLP (Multilayer Perceptron);
– LSTM (Long Short-Term Memory);
– CNN (Convolutional Neural Network);
– SNN (Spiking Neural Network);
– GAN (Generative Adversarial Network).

Experimental studies were carried out on the FPGA model HAPS-80 S52, during which the required number of LUTs was measured depending on the number of layers and the number of neurons on each layer for the above types of ANNs. As a result of the research, specific types of functions depending on the required number of LUTs on the type, number of layers, and neurons for the most commonly used types of ANNs in practice were determined.

A feature of the results obtained is the fact that with a sufficiently high accuracy, it was possible to determine the analytical form of the functions that describe the dependence of the required number of LUT FPGA for the implementation of various ANNs on it. According to calculations, GAN uses 17 times less LUT compared to CNN. And SNN and MLP use 80 and 14 times less LUT compared to LSTM. The results obtained can be used for practical purposes when it is necessary to make a choice of any FPGA for the implementation of an ANN of a certain type and structure on it

Keywords: FPGA, MLP, LSTM, CNN, SNN, GAN

UDC 004.032.26
DOI: 10.15587/1729-4061.2023.281731

DEVELOPMENT OF A MODEL FOR DETERMINING THE NECESSARY FPGA COMPUTING RESOURCE FOR PLACING A MULTILAYER NEURAL NETWORK ON IT

Bekbolat Medetov
PhD*
Tansaule Serikov
PhD*
Arai Tolegenova
Candidate of Technical Sciences*
Dauren Zhexebay
Corresponding author
PhD, Senior Lecturer**
E-mail: zhexebay92@gmail.com
Asset Yskak
Master
Department of Computer Science
Nazarbayev University
Kabanbay batyr ave., 53,
Astana, Republic of Kazakhstan, 010000
Timur Namazbayev
Master, Senior Lecturer**
Nurtay Albanbay
Master
Department of Cybersecurity, Information Processing and Storage
Satbayev University
Satbaev str., 22, Almaty, Republic of Kazakhstan, 050013
*Department of Radio Engineering, Electronics and Telecommunications
S. Seifullin Kazakh Agro Technical Research University
Zhenis ave., 62, Astana, Republic of Kazakhstan, 010011
**Department of Solid State Physics and Nonlinear Physics
Al-Farabi Kazakh National University
Al-Farabi ave.,71, Almaty, Republic of Kazakhstan, 050040

Received date 12.05.2023
Accepted date 14.07.2023
Published date 31.08.2023

How to Cite: *Medetov, B., Serikov, T., Tolegenova, A., Zhexebay, D., Yskak, A., Namazbayev, T., Albanbay, N. (2023). Development of a model for determining the necessary FPGA computing resource for placing a multilayer neural network on it. Eastern-European Journal of Enterprise Technologies, 4 (4 (124)), 34–45. doi: https://doi.org/10.15587/1729-4061.2023.281731*

1. Introduction

Modern research in the field of artificial neural networks (ANNs) shows that they are very good at solving many problems related to the classification and processing of big data, such as images, audio, and video data. It is known that when solving such problems, it turns out that the dimensionality of data and the computational complexity of neural calculations turn out to be significantly large, that even multi-core, accelerated, powerful general-purpose processors CPUs do not cope well with calculations. For comfortable and efficient work with modern ANNs, as a rule, powerful and rather expensive GPUs are used. This is especially true for processing a large data stream, for example, video information in real-time. Recently, as an alternative to the GPU for the implementation of ANNs,

FPGAs are often paid attention to. This is due to the following factors:

– FPGA consumes much less energy, and this is important when using ANN in devices where there are difficulties in providing external power (for example, spacecraft);

– dimensions and weight of the device;

– requirements for special cooling systems of the device;

– price.

Since the need for equipment for working with artificial neural networks is constantly growing, the task arises to determine all possible technical requirements for FPGAs for the implementation of ANNs of various types and structures with their help. One of these requirements is to determine the sufficiency of FPGA computing resources depending on the type and parameters of the ANN (for example, the number of layers and neurons on the layers).

However, due to the peculiarities of the operation of FPGAs, when implementing ANNs on them, one problem arises related to determining the correspondence between the computing resources of the device and the computational requirements of neural networks. Without knowing this correspondence, it is very difficult to select an FPGA for implementing an ANN of one structure or another on it. In this regard, in this paper, the task was set to build a certain model, with the help of which it would be possible to predict the required computing resources of the FPGA, depending on the type, structure, and size of a particular ANN.

Therefore, studies designed to predict the calculations of FPGA resources for the implementation of ANNs are relevant.

2. Literature review and problem statement

The current deep learning trend has sparked interest in implementing artificial neural networks on FPGAs in order to improve performance. The work [1] presents the results of studies on the influence of the choice of representation of numbers (floating point vs. fixed point) and the choice of the number system both on the use of hardware resources and on the performance of a neural network. The FPGA is shown to be energy efficient, and the data learning rate is slightly slower than that of a desktop computer when using fixed-point data. But questions remained unresolved related to the large-scale assessment of the used FPGA resources for different structures of multilayer neural networks. The reason for this may be the costly part in terms of training each neural network model depending on the number of layers and neurons in the layers.

In [2], the results of studies on the implementation of multilayer neural networks with an accuracy of 24, 20, 16, 12, and 8 bits are presented. The neural network model with 16-bit fixed-point data accuracy is shown to be the most efficient MLP design in terms of classification accuracy, latency, power consumption, and resource usage. Also, issues related to a large-scale assessment of the used FPGA resources are not considered in this paper. The reason for this may be the use of FPGA as an accelerator for the task of classifying human activity in real-time, and not the implementation of different structures of multilayer neural networks in order to determine the optimal FPGA resources for their implementation.

In [3], the results of a study of the acceleration of a multilayer perceptron on an FPGA for the classification of handwritten digits are presented. It is shown that MLP with one hidden layer of 12 neurons, implemented on FPGA, provides a classification accuracy of 93 % and a 10-fold acceleration compared to existing works. But questions remained unresolved related to the assessment of the required amount of FPGA resources for different numbers of layers and neurons in layers. The reason is also the use of FPGAs to accelerate the MLP model, not their resource estimate.

Convolutional Neural Networks (CNNs) are widely used in computer vision and pattern recognition due to their high accuracy. There are various CNN models for pattern recognition such as VGG, ResNet, GoogLeNet, MobileNet, R-CNN, SSD, and YOLO. The paper [4] presents the results of studies of the implementation of MobileNetV2 on the FPGA Arria 10 SoC. The results of these studies show that this accelerator can classify each image from ImageNet in 3.75 ms, which is about 266.6 frames per second. The FPGA design is shown to provide a 20x speedup compared to the processor. The paper [5] presents the results of studies of the hardware implementation of the YOLOv2 model on the Xilinx ZYNQ xc7z035 FPGA for object detection. The design is shown to provide a total throughput of 111.5 GOP/s and a power efficiency of 18.71 GOP/s/W at 200 MHz FPGA operating frequency. Compared to the central processor, the proposed accelerator increases energy efficiency by 33.4 times. But questions remained unresolved related to the assessment of the used FPGA resources for CNN models with different structures. Nevertheless, this paper does not provide an estimate of the required amount of FPGA resources for implementing CNN-type networks on it.

One of the most commonly used artificial neural network architectures in practice is Long Short-Term Memory (LSTM), which belongs to the type of recurrent neural network (RNN). LSTM is characterized by the presence of feedback, which limits the high degree of parallelism of general-purpose processors such as CPUs and GPUs. Also, in terms of the energy efficiency of data center applications, the high consumption of GPU and CPU computing cannot be ignored. An ideal alternative to solve the above problems is the FPGA. The paper [6] presents the results of studies on the implementation of the FPGA-based LSTM network acceleration mechanism. It is shown that compared to CPU and GPU, the FPGA-based acceleration mechanism can provide 8.8 and 2.2 times performance improvement and 16.9 and 9.6 times energy efficiency improvement, respectively, within the framework of Caffe. However, in this work, the exactingness of LSTM-type networks to FPGA resources is not evaluated.

In recent years, there has been significant interest in «generative adversarial networks» (GANs) due to their ability to generate data with high accuracy. Many GAN models have been proposed for a variety of areas, from natural language processing to image processing [7]. For image processing, convolution and deconvolution operations are used. Much of the work has focused on accelerating deconvolutional neural networks (DCNNs) on FPGAs [8]. Because accelerators designed for convolution operations do not play well with the deconvolution operation. Obviously, specialized accelerators are needed to achieve high performance with GANs. The paper [7] provides an overview of GAN acceleration methods and architectures, including those based on FPGA. But in these works, questions are only considered regarding the acceleration of calculations for GAN-type networks on FPGAs, without affecting the problem of choosing an FPGA depending on the computational requirement of a neural network.

Spiking Neural Networks (SNNs) are the most accurate mimics of biological neural networks. In [9], the results of a study of the hardware implementation of SNN on FPGA

are presented. The design has been shown to support up to 16,384 neurons and 16.8 million synapses, but requires minimal hardware resources and provides a very low power consumption of 0.477 W. The evaluation results show an accuracy of 97.06 % and a frame rate of 161 fps. In this paper, there is some attempt to estimate the FPGA hardware resources required to implement an SNN on it. But these results are limited to a particular case and do not have a general character. The paper [10] presents the results of studies of the influence of various coding methods on the accuracy of SNNs and implementation in the Xilinx Zynq ZCU102 hardware architecture. It is shown that the experimental results of the MNIST dataset can achieve an accuracy of 98.94 % with eight-bit quantized weights. In addition, it reaches 164 frames per second (FPS) at 150 MHz and gets 41x faster than the CPU implementation and 22x less power than the GPU implementation. This paper touches upon issues related only to the study of accuracy and acceleration of calculations for SNNs on FPGAs. Also, no attention has been paid to determining the necessary FPGA resource for implementing SNN-type networks on it.

The review of the literature shows that the use of FPGAs for neural computing is highly efficient in terms of providing maximum performance and low power consumption. At present, it is already known that all known types of artificial neural networks with various activation functions can be successfully implemented on FPGAs. It should be noted that FPGAs are used not only to implement neural networks, but also to implement many other tasks, for example, in cryptographic data encryption [11]. However, one problem remains poorly understood, related to the determination of the technical characteristics of an FPGA for the implementation of a certain neural network structure on it. It is known that the main technical characteristic of an FPGA, which describes its computing resource, is LUT (Look-up table – the basic FPGA structure that performs logical operations). The LUT is the basic building block of an FPGA and is capable of implementing any kind of logic.

An analysis of the available scientific papers has shown that there are no detailed studies to determine the compliance of FPGA resources with the computational needs of various neural networks.

3. The aim and objectives of the study

The aim of the study is to build a mathematical model that allows you to estimate approximately the required computing resources of the FPGA to place on them an ANN of a certain type and structure.

To achieve this aim, the following objectives are accomplished:

– to conduct experimental measurements of the calculation of the number of LUTs required to implement various types of ANNs (SNN, MLP, LSTM, CNN, GAN);

– to carry out a comparative analysis of the computational resource intensity of ANN types to determine the efficiency of using FPGAs in their implementation.

4. Materials and methods

The object of study in this work is the implementation of ANN on FPGA. At the same time, the main issue of the study is to determine the compliance of FPGA resources with the computational needs of ANN. It is assumed that there is a close relationship between the required amount of FPGA computing resources and the main ANN parameters such as the number of neurons and the number of layers. Thus, the main hypothesis of the study is the assumption that the required amount of FPGA resources can be represented as a function depending on two arguments – the number of layers and the number of neurons for each type of ANN. The form and numerical characteristics of these functional dependencies can only be obtained experimentally, in which case the result of the research will have to be formalized in the form of empirical formulas. For this reason, experimental studies are required in the context of each type of ANN.

During the experimental measurements of the required number of LUTs, depending on the parameters of the neural network, the following agreements were accepted:

– arithmetic calculations are performed only with the help of LUT;

– data transferred from one layer to another is stored in FF blocks, and weights are stored in LUT;

– data for all types of networks are presented in 24-bit size;

– to ensure maximum performance, neural calculations for each layer of the network are performed strictly in parallel.

The use of only LUT in the calculation is due to the small number of DSP blocks in the FPGA. Using them together with LUT does not result in a significant increase in the productivity of calculations. Because the number of DSPs is directly proportional to the number of multiplication operations, of which there are a huge number of neural networks.

To automatically generate Verilog code that implements a neural network of a certain type and structure, a special Python library was developed. The correctness of the operation of neural networks generated in the Verilog language was verified using the functions of the PyTorch library of the Python language. The check was carried out as follows:

– a certain neural network was designed using the PyTorch library of the Python language (source network);

– the parameters of the original network are transferred to the corresponding functions of a special library written in the Python language, which at the output generates a code in the Verilog language that implements the neural network for its placement on the FPGA;

– the same data is given to the input of the original network on the computer and the version of the same network on the FPGA;

– at the end, the results of the output of these networks are compared, and if they match, it is considered that the neural network implementation code on Verilog was generated correctly.

In our work, we analyzed the correctness of the following types of neural networks generated for FPGAs:

– MLP (Multilayer Perceptron);
– LSTM (Long Short-Term Memory);
– CNN (Convolutional Neural Network);
– SNN (Spiking Neural Network);
– GAN (Generative Adversarial Network).

The measurements performed showed that all the analyzed neural networks at the output gave virtually the same results both in the computer version and in the version on the FPGA. However, there are small deviations, within less than 1 %, between these results. These deviations are not related to the fact that our library incorrectly generates the Verilog code of the neural network, but to the fact that in the

computer version of the neural network, the data is presented in floating-point format, while in the FPGA version, the data is presented in fixed-point format.

In this work, for experimental measurements, the Xilinx Virtex UltraScale XCVU440 FLGA2892 FPGA based on the HAPS-80 S52 system from Synopsys was used, which has the following characteristics:

– CLB LUTs – 2,532,960;
– CLB Flip-Flops – 5,065,920;
– Block RAM – 2,520;
– DSP Slices – 2,880;
– Clock frequency – 300 MHz.

The required number of LUTs to accommodate a neural network of a certain type and structure was calculated from the results of compiling its Verilog code using the HAPS (R) ProtoCompiler S Version T-2022.12 software.

Below are the results of experimental measurements and processing of the obtained measurements.

5. Results of research on the estimation of resources of a field-programmable gate array in the implementation of neural networks

5. 1. Experimental results for different types of artificial neural networks

5. 1. 1. Experiments for Spiking Neural Networks

Spiking Neural Networks (SNN) are designed to take advantage of time-varying data. SNN inputs are encoded in a sequence of spikes and continue for a certain amount of time. The input data is presented as a sequence of zeros and ones, where one corresponds to the appearance of a spike. Fig. 1 shows the block diagram of the SNN that was used in our experiments.

When conducting experiments, the SNN model is first created in Python using the PyTorch and snnTorch libraries. As you can see from Fig. 1, PyTorch is used to form connections between neurons, and snnTorch is used to create neurons. The snnTorch library supports the following Leaky Integrate-and-Fire Neuron Models (the neuron model used in SNN):

– Lapicque – Lapicque's RC model;
– Leaky – 1st-order model;
– Synaptic – Synaptic Conductance-based neuron model.

Leaky Integrate-and-Fire Neuron Model, like a classical neural network, weights the input data, and unlike it, integrates the weighted input data and compares it with a threshold value. If the values exceed the threshold value, then a spike appears at the output.

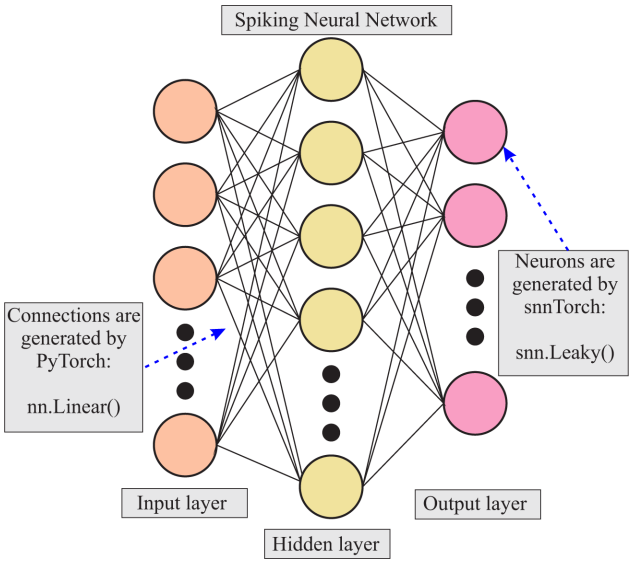

Fig. 1. Spiking neural network structure for experiments

In our experiments, the Leaky neuron model was used, which is more optimal in terms of the number of parameters compared to other models. Since the structure of an SNN is determined by the number of layers and the number of neurons in each layer, experimental measurements of the required number of LUTs were carried out following these rules:

– each layer has the same number of neurons;
– the number of neurons varies from 10 to 50 with a step of 5;
– the number of neurons is fixed, and the number of layers varies in the range from 10 to 100 with a step of 10.

Table 1 shows the data obtained from the results of compiling different SNN structures, where the values of the required LUTs are stored in the cells of the table.

As we can see from Table 1, the required number of LUTs is a function of two variables (the number of layers and the number of neurons in each layer):

$$z = f(x, y), \tag{1}$$

where x is the number of neurons, y is the number of layers, and z is the number of LUTs required to place the neural network on the FPGA. The main task of our research is to determine the type of function (1). To do this, we analyze the data from Table 1.

Table 1

SNN compilation results with different numbers of layers and neurons

neurons \ layers	10 layers	20 layers	30 layers	40 layers	50 layers	60 layers	70 layers	80 layers	90 layers	100 layers
10 neurons	25118	51302	76570	101862	127191	151170	176348	202298	227176	255466
15 neurons	44738	88937	133745	176700	221221	266135	314091	358620	403370	448590
20 neurons	69627	138759	206847	276912	347281	416027	486703	555701	624385	694587
25 neurons	97844	194185	291209	389911	486178	583131	681876	779402	875148	972850
30 neurons	128950	257451	386045	517365	646827	774725	904698	1033669	1163449	1292394
35 neurons	168196	335302	503302	669622	835913	1004848	1171850	1340464	1506208	1673294
40 neurons	209249	417478	625162	835820	1043744	1253409	1460297	1672145	1880339	2088556
45 neurons	254833	506657	762163	1016108	1270655	1524748	1780616	2033327	2286523	–
50 neurons	303876	606803	910207	1213746	1517665	1819097	2124285	2426886	–	–

Fig. 2 shows the dependence of the required number of LUTs on the number of layers for a fixed value of neurons on the layers, obtained on the basis of experimental data (Table 1).

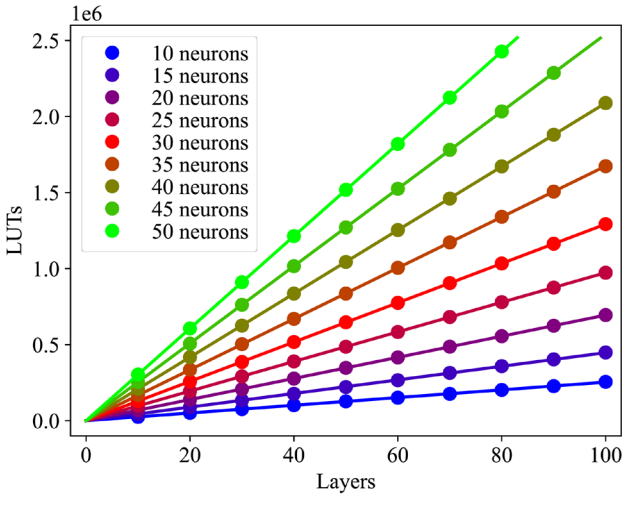

Fig. 2. Dependence of the required number of look-up tables on the number of layers of the neural network for various fixed values of the number of neurons

Fig. 2 shows that the dependence of the required number of LUTs on the number of layers of the neural network at a fixed value of the number of neurons is a linear function, and can be described by a group of the following functions:

$$z_i(y)=k(x_i)\cdot y,\ i=1,2,...n, \tag{2}$$

where n is the number of options for the number of neurons (in this case, n=9), x_i is the number of neurons (takes a fixed value), y is the number of layers (it changes from 10 to 100 with a step of 10), $z_i(y)$ is the number of required LUT depending on the number of layers at a fixed value of the number of neurons.

The set of coefficients $k(x_i)$ in formula (2) can be determined using the least squares method as follows:

$$k(x_i)=\frac{\sum_{j=1}^{m} y_j\cdot z_j}{\sum_{j=1}^{m} y_j^2}, \tag{3}$$

where m is the number of options for the number of layers. Table 1 shows that for the number of neurons 50, m=8, and for the number of neurons 45, m=9, and in all other cases m=10. The limitation of the number of layers is due to the limitation of the LUT resources of the XCVU440 FPGA used in our experiments. Fig. 3 shows the dependence of the coefficient $k(x_i)$ on the number of neurons.

Fig. 3 shows that the dependence of the coefficient $k(x_i)$ on the number of neurons is non-linear. This dependence can be approximated by a power function of the following form:

$$k(x)=c\cdot x^{\alpha}, \tag{4}$$

where c and α are some positive real numbers.

To find the parameters c and α, we can again use the least squares method. For this, equation (4) can be translated into a logarithmic form:

$$\ln(k)=\ln(c)+\alpha\cdot\ln(x). \tag{5}$$

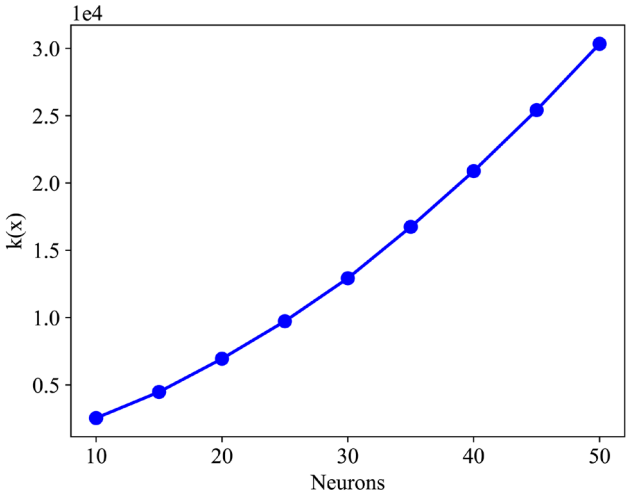

Fig. 3. Dependence of the coefficient $k(x_i)$ on the number of neurons

If we take Y=ln(k), A=ln(c), B=α and X=ln(x), then equation (5) can be rewritten as follows:

$$Y=A+B\cdot X. \tag{6}$$

According to the least squares method, the coefficients A and B in equation (6) are calculated by the formulas:

$$B=\frac{\sum_i\left[\left(X_i-\bar{X}\right)\cdot Y_i\right]}{\sum_i\left(X_i-\bar{X}\right)^2}, \tag{7}$$

$$A=\bar{Y}-B\cdot\bar{X}, \tag{8}$$

where $\bar{X}$ and $\bar{Y}$ are the averages of the argument and functions (6).

Fig. 4 shows a graph of the dependence of the coefficient k on the number of neurons on a logarithmic scale.

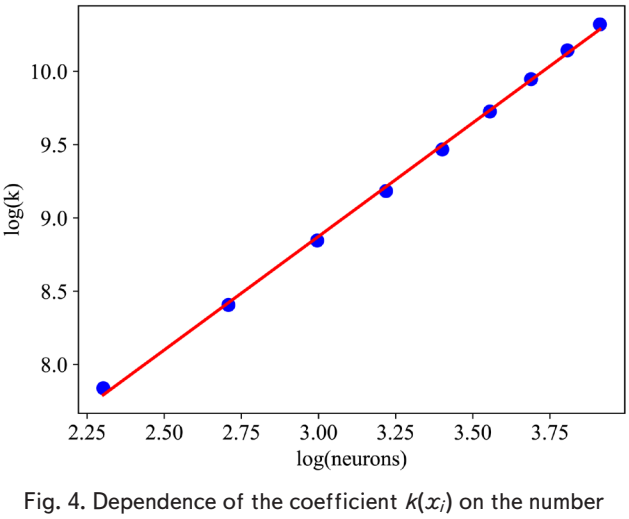

Fig. 4. Dependence of the coefficient $k(x_i)$ on the number of neurons on a logarithmic scale

In Fig. 4, the dots indicate the logarithmic data taken from Fig. 3, and the red line corresponds to the graph of the approximating function according to formula (8). As can be seen from Fig. 4, the dependence of the coefficient $k(x_i)$ on the number of neurons on a logarithmic scale is indeed linear.

Knowing the coefficients, A and B, we can easily find the values of the parameters c and α included in equation (4). Our calculations show that $c \approx 68.29$ and $\alpha \approx 1.55$. In this case, the desired function given in equation (1), taking into account formula (2), takes the following form:

$$z(x,y) = 68.29 \cdot y \cdot x^{1.55}. \quad (9)$$

The correctness of function (9), which makes it possible to predict the required number of LUTs when implementing an SNN on an FPGA, can be checked using the data from the same Table 1. This time, we will plot the dependence of the number of required LUTs on the number of neurons for fixed values of the number of layers. Fig. 5 shows the graphics data.

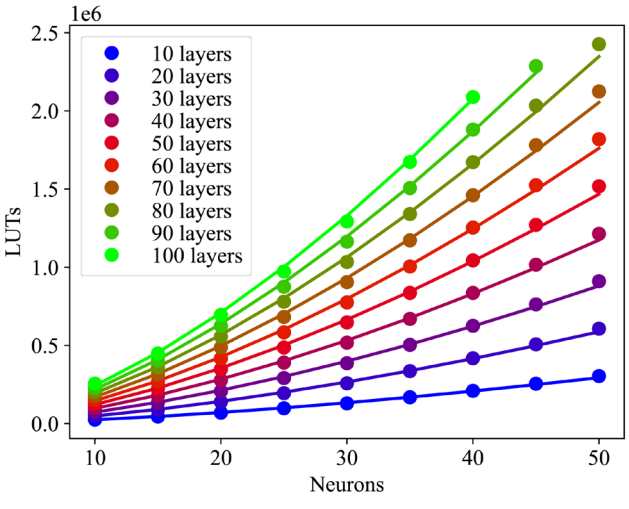

Fig. 5. Dependence of the required number of look-up tables on the number of neurons in the neural network for various fixed values of the number of layers

In Fig. 5, the dots indicate the data taken from Table 1, and the lines are the graphs of the approximating functions. These graphs show that the found empirical dependence of the required number of LUTs on the number of layers and neurons for SNN is quite accurately described by the function presented in (9). Thus, knowing this dependence, it is possible to estimate how many LUT resources are required to place an SNN of a certain structure on an FPGA. An example of using this rating is shown in Fig. 6, where this rating is applied to the FPGA resource capabilities of the HAPS-80 S52 system. In Fig. 6, the assessment is carried out as follows:

– set the number of layers and neurons SNN;

– the required number of LUTs is calculated by the formula (9);

– the calculated value of the required number of LUTs is compared with the maximum number of LUTs available in the considered FPGA;

– if the calculated value of the required number of LUTs is less than the maximum number of LUTs available in this FPGA model, then we consider that this SNN structure can be placed on this FPGA, otherwise – no.

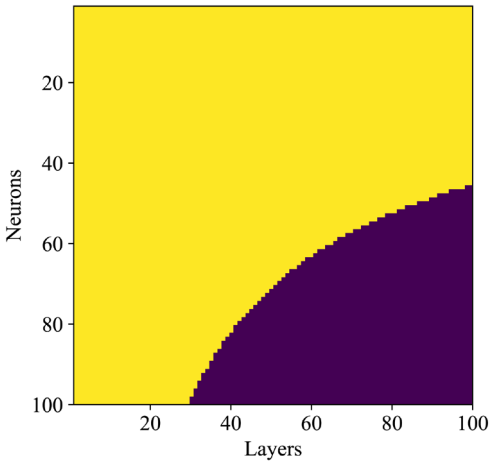

Fig. 6. Distribution of the number of look-up tables on the plane of the number of neurons and the number of layers for the field-programmable gate array of the HAPS-80 S52 system when placing a spiking neural network on it

The light area of the two-dimensional plane in Fig. 6 corresponds to the SNN structures that can be placed on the FPGA of the HAPS-80 S52 system, and the dark area corresponds to the inaccessible SNN structure for this type of FPGA.

5. 1. 2. Experiments for Multilayer Perceptron

During the experimental measurements for a multilayer fully connected perceptron (MLP), the ReLU activation function was chosen. As you know, the structure of this network is determined by the number of layers and the number of neurons on the layers. In this regard, experimental measurements of the required number of LUTs were carried out by the following rules:

– each layer has the same number of neurons;

– the number of neurons varies from 3 to 24 with a step of 3;

– the number of neurons is fixed, and the number of layers varies in the range from 5 to 50 with a step of 5.

Table 2 shows the data obtained from the results of compiling different MLP structures.

Table 2

MLP compilation results with different numbers of layers and neurons

layers / neurons	5 layers	10 layers	15 layers	20 layers	25 layers	30 layers	35 layers	40 layers	45 layers	50 layers
3 neurons	6545	12963	19854	27457	34335	40707	46605	54619	59833	67967
6 neurons	24252	48045	74257	95344	121064	143614	170100	192827	219079	243834
9 neurons	51586	107084	159014	213663	268356	321457	378265	431294	481581	537312
12 neurons	91207	183578	275396	368140	459211	553262	642267	735192	831347	921994
15 neurons	141069	285657	426817	570035	715282	861666	1002101	1147448	1291019	1437629
18 neurons	201395	403522	605566	813908	1013488	1219525	1423056	1626500	1821259	2029374
21 neurons	272300	550049	823817	1095390	1381356	1655055	1926840	2203047	2491536	–
24 neurons	347767	703136	1063951	1413983	1768049	2131086	2487761	–	–	–

Obviously, the dependence of the required number of LUTs on the MLP parameters, as in the case of SNN, is a function of two arguments. Fig. 7 shows the dependence of the required number of LUTs for MLP on the number of layers for a fixed value of neurons on the layers. Fig. 7 shows that, just as in the case of SNN, the dependence of the required number of LUTs on the number of layers is a linear function. Calculations show that in this case, the coefficient k also depends non-linearly on the number of neurons (Fig. 8).

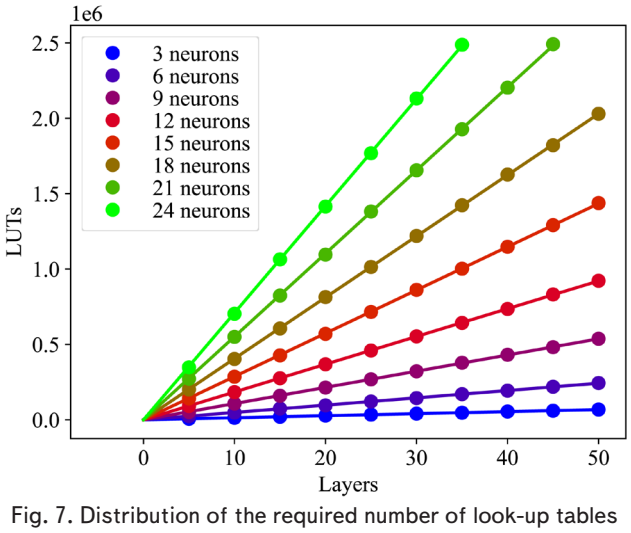

Fig. 7. Distribution of the required number of look-up tables on the number of layers for multilayer perceptron at various fixed values of the number of neurons

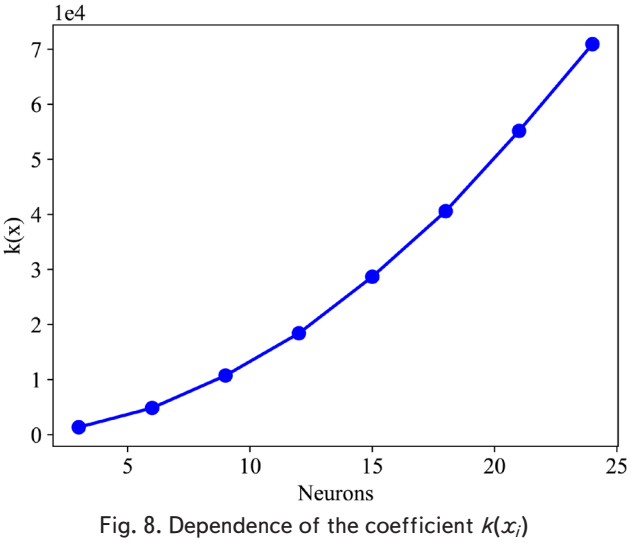

Fig. 8. Dependence of the coefficient $k(x_i)$ on the number of neurons

We represent the dependence in Fig. 8 in the form (4). Further, Fig. 9 shows a graph of the dependence of the coefficient k on the number of neurons on a logarithmic scale.

As we can see from Fig. 9, the dependence of the coefficient $k(x_i)$ on the number of neurons on a logarithmic scale for MLP is indeed linear. The final calculations lead us to the fact that the dependence of the required number of LUTs on the number of layers and neurons for MLP obeys the following pattern:

$$z(x,y)=161.67\cdot y\cdot x^{1.91}. \quad (10)$$

Fig. 10 shows graphs of the number of required LUTs versus the number of neurons for fixed values of the number of layers.

An example of using the empirical dependence (10) to assess the resource capability of the FPGA of the HAPS-80 S52 system is shown in Fig. 11.

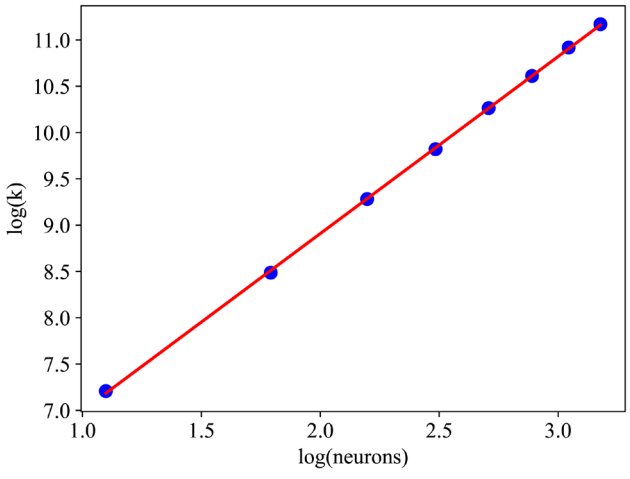

Fig. 9. Dependence of the coefficient $k(x_i)$ on the number of neurons on a logarithmic scale for multilayer perceptron

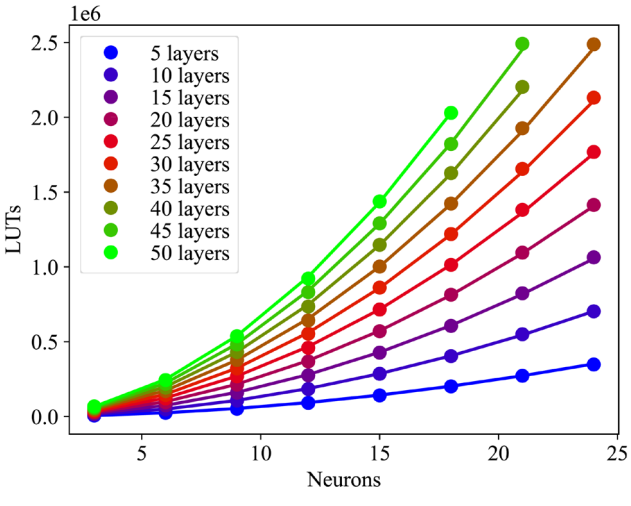

Fig. 10. Dependence of the required number of look-up tables on the number of neurons for multilayer perceptron at various fixed values of the number of layers

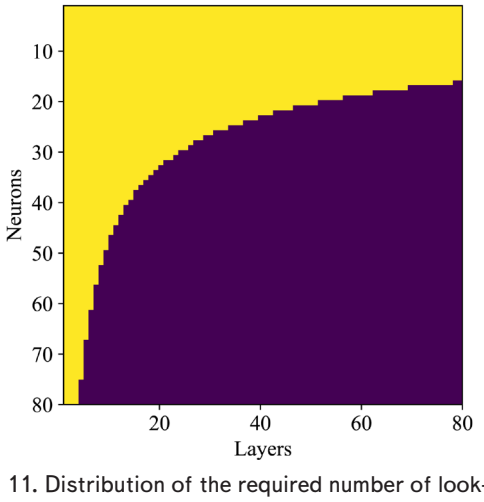

Fig. 11. Distribution of the required number of look-up tables on the plane of the number of neurons and the number of layers for the FPGA of the HAPS-80 S52 system when placing a multilayer perceptron on it

Just as in the case of SNNs, Fig. 11 shows the areas of acceptable and inaccessible values for the parameters of an MLP-type network. For example, on the FPGA model HAPS-80 S52, it is impossible to place an MLP network that has 40 layers and 50 neurons on each layer. But there an MLP-type neural network can be placed if it has 20 layers and 30 neurons on each layer.

5. 1. 3. Experiments for Long Short-Term Memory networks

During experimental measurements in networks with Long Short-Term Memory (LSTM), we considered the case when the number of inputs, the number of cells in the layer, and the number of outputs of the network are equal. The sequence input layer injects sequence or time series data into a neural network that can learn long-term relationships between the time steps of the sequence data.

In our LSTM cells, we used four non-linear activation functions, of which, three sigmoid functions to activate the node output and one hyperbolic tangent (tanh) function to activate the cell state.

When carrying out experimental measurements of the required number of LUTs for LSTM networks, the number of layers changed from 2 to 14 with a step of 2, and the number of cells changed from 2 to 10. The results of the calculated numbers of LUTs depending on the number of layers and cells of the LSTM network are shown in Table 3.

The structure of the LSTM, as in the cases of SNN and MLP, is determined by two parameters, respectively, the function of predicting the required number of LUTs is determined in a similar way. Fig. 12 shows a graph of the number of LUTs versus the number of layers of the LSTM network.

From Fig. 12, we see that in the case of an LSTM network, the dependence of the required number of LUTs on the number of layers is also quite well described by a linear function of the form (2). And Fig. 13, 14 show graphs of the dependence of the coefficient k on the number of cells in normal and logarithmic scales.

Fig. 15 below shows the graphs of the required number of LUTs versus the number of cells for LSTM networks.

In Fig. 15, the points are the experimental data, and the solid curves are the graphs of the corresponding approximation functions. This figure shows that for small values of the number of layers, there is a noticeable discrepancy between the experimental data and the approximation function. To predict the required number of LUTs, the most important are the limit values for the number of layers, so we believe that these discrepancies will not greatly affect the final function of the form (1) for LSTM networks.

The empirical form of the dependence of the required number of LUTs on the number of cells and layers for LSTM networks takes the following form:

$$z(x,y)=8213.65\cdot y\cdot x^{1.38}. \quad (11)$$

As in the previous cases, Fig. 16 shows an example of using the prediction function (11) for LSTM networks to the FPGA of the HAPS-80 S52 system.

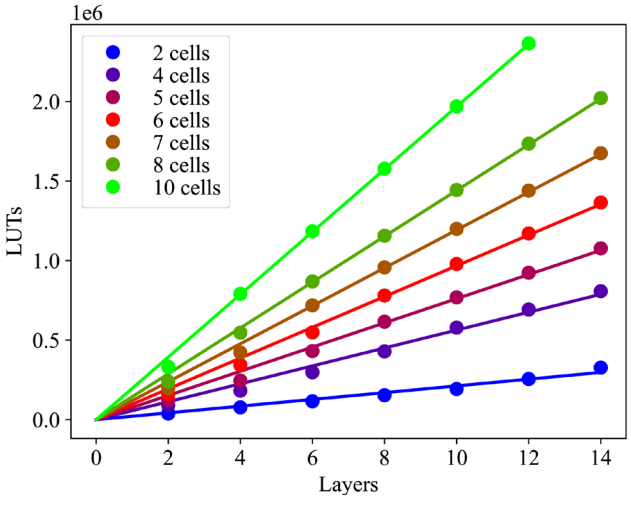

Fig. 12. Dependence of the required number of look-up tables on the number of layers for long short-term memory networks at various fixed values of the number of cells

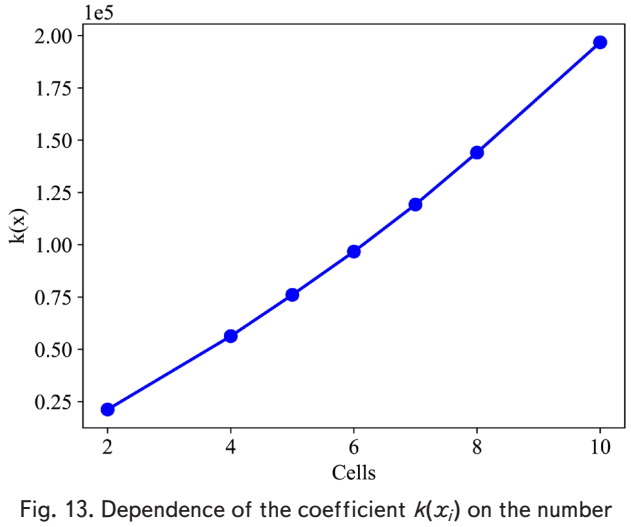

Fig. 13. Dependence of the coefficient $k(x_i)$ on the number of cells for long short-term memory networks

Table 3

Results of compilation of LSTM networks with different numbers of layers and cells

cells \ layers	2 layers	4 layers	6 layers	8 layers	10 layers	12 layers	14 layers
2 cells	38485	77316	115325	153555	192596	255622	326956
4 cells	91148	182306	297802	428826	578482	692407	807442
5 cells	123444	245416	430949	615704	768686	923438	1076829
6 cells	159652	342968	548528	780248	978601	1170438	1365135
7 cells	196452	421872	718463	957348	1199257	1439995	1674785
8 cells	238936	546626	868567	1155944	1443844	1735005	2021585
10 cells	331606	790870	1184062	1577875	1969138	2365579	–

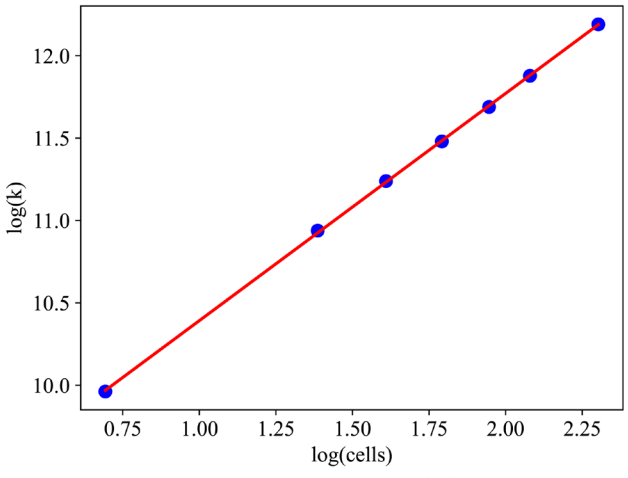

Fig. 14. Dependence of the coefficient $k(x_i)$ on the number of cells on a logarithmic scale for long short-term memory networks

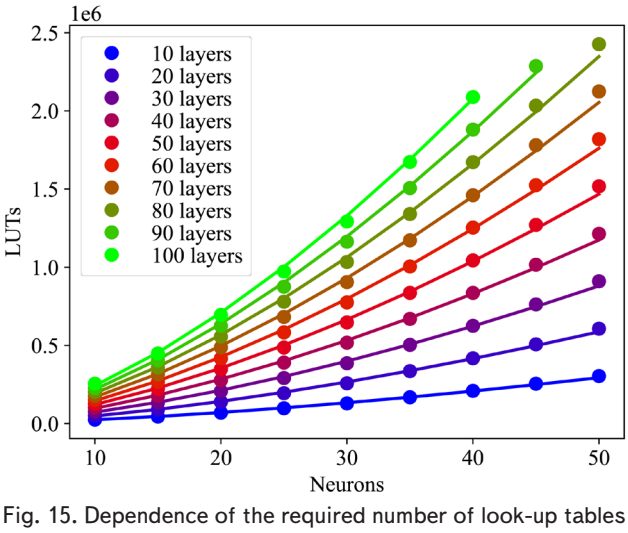

Fig. 15. Dependence of the required number of look-up tables on the number of cells for long short-term memory networks at various fixed values of the number of layers

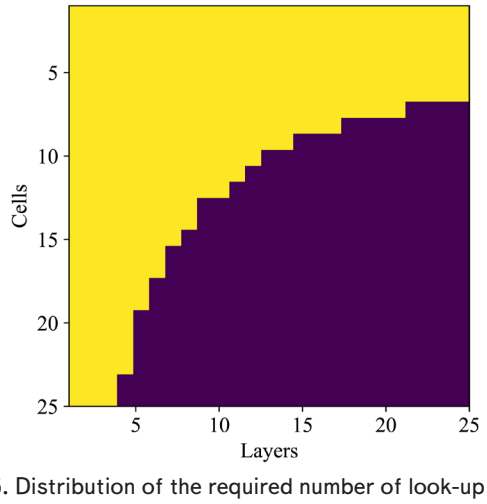

Fig. 16. Distribution of the required number of look-up tables on the plane of the number of neurons and the number of layers for the FPGA of the HAPS-80 S52 system when a long short-term memory networks is placed on it

Fig. 16 shows a graph showing the areas of acceptable and inaccessible values for the parameters of the LSTM-type neural network when placed on the FPGA model HAPS-80 S52.

5. 1. 4. Experiments for Convolutional Neural Network

During the experimental measurements on CNN, it was decided to use the padding=same parameter so that the amount of output data is equal to the amount of input data. Accordingly, at each stage of the experiment, the number of a set of layers consisting of conv1d, relu and maxpooling1d increases. A rectifier linear unit (ReLU) is used as the main activation function. Thus, the number of required LUTs for a CNN depends on only one parameter – the number of network layers. Table 4 shows the data obtained from the results of compiling different CNN structures.

Table 4

CNN compilation results with different numbers of layers

10 layers	20 layers	30 layers	40 layers	50 layers	60 layers	70 layers
39641	72910	104676	145457	177870	211614	239500
80 layers	90 layers	100 layers	200 layers	300 layers	400 layers	500 layers
288151	310735	333975	679556	1027971	1382391	1685905

Analysis of the data in Table 4 shows that the dependence of the required number of LUTs on the number of CNN layers is linear and can be described by the function:

$$z = k \cdot x, \tag{12}$$

where z is the number of required LUTs, x is the number of CNN layers.

The empirical coefficient k is calculated in the same way as by formula (3). The value of the coefficient turned out to be 3411.92, while the relative error of its determination is 0.67 %. Fig. 17 shows a graph of the approximating function (12) and experimental data.

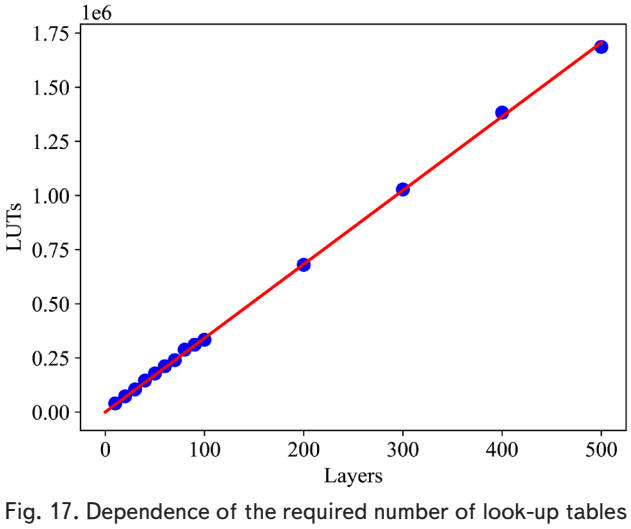

Fig. 17. Dependence of the required number of look-up tables on the number of layers for convolutional neural network

Thus, the empirical form of the dependence of the required number of LUTs on the number of layers for CNN takes the following form:

$$z = 3411.92 \cdot x. \tag{13}$$

According to the results of experimental measurements, it was found that CNN consumes the least amount of

LUT resources. Calculations using formula (13) show that a CNN with a number of layers of about 750 can be placed on the FPGA of the HAPS-80 S52 system.

5. 1. 5. Experiments for Generative Adversarial Network

It is known that GAN is a framework for learning a DL (deep learning) model to collect a distribution of training data so that we can generate new data from the same distribution. They consist of two different models: generator and discriminator. The generator's job is to create «fake» images that look like training images. The job of the discriminator is to look at the image and infer whether it is a real training image or a fake image from the generator.

There are several different architectures for building GAN networks:

– fully connected GAN (Fully connected GAN);
– convolutional GAN (Convolutional GAN);
– conditional GAN (Conditional GAN);
– GAN with inference models;
– adversarial autoencoders.

Early GAN architectures used fully connected neural networks for both generator and discriminator. Moving from fully connected to convolutional neural networks (CNN) is a natural progression given that CNN is very well suited to image data [12].

If we take the architecture of Fully connected GAN, then the results of network scalability will be identical to the results of MLP. Therefore, in this work, the architecture was used – Convolutional GAN. Since the discriminator model is similar to the CNN model, it will be enough for us to implement only the generator model.

The implementation of the GAN generator model corresponded to the model built using the PyTorch library. The network consisted of a ConvTranspose2d layer. ReLU was used as an activation function. The parameters of the ConvTranspose2d layer are as follows: in_channels in the input layer 5, in the rest 2, out_channels in the output layer 1, in the rest 2, kernel_size=(3,1), stride=1, padding=0, output_padding=0, groups=1, bias=False, dilation=1, padding_mode='zeros', device=None, dtype=None.

The scale of a GAN network is determined primarily by the number of layers. In this regard, in the experiment to create GAN networks of different structures, the number of layers varied in the range from 5 to 40 with a step of 5.

Table 5 shows the data obtained from the results of compiling different GAN structures.

Table 5

Results of compiling GAN networks with different numbers of layers

Layers	5	10	15	20	25	30	35	40
LUTs	35079	149041	328311	591727	923218	1310442	1819853	2387473

The required number of LUTs for GAN is a function of one variable (the number of layers) and can be represented as follows:

$$z = f(x), \tag{14}$$

where x is the number of layers, z is the number of LUTs required to place the neural network on the FPGA.

Fig. 18 shows graphs of the required number of LUTs versus the number of layers for GAN networks.

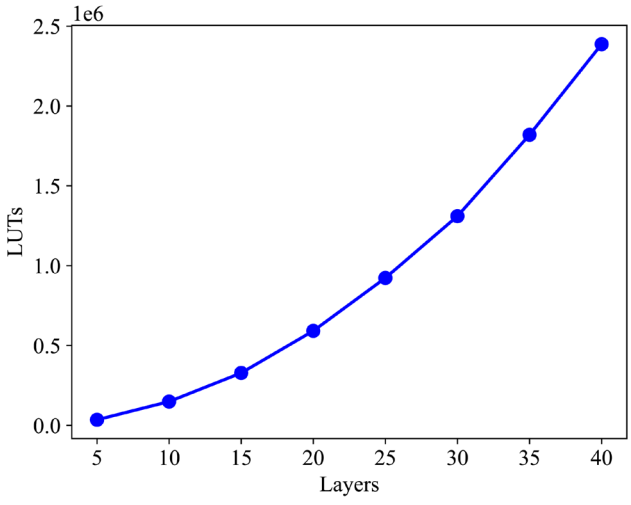

Fig. 18. Dependence of the required number of look-up tables on the number of layers for generative adversarial network networks

Fig. 18 shows that the dependence of the required number of LUTs on the number of layers of the GAN network is non-linear.

This dependence can be approximated by a power function of the following form:

$$k(x) = c \cdot x^{\alpha}, \tag{15}$$

where c and α are some positive real numbers.

To find the parameters c and α, we can again use the least squares method. For this, equation (15) can be translated into the logarithmic form:

$$\ln(z) = \ln(c) + \alpha \cdot \ln(x). \tag{16}$$

If we take $Y=\ln(k)$, $A=\ln(c)$, $B=\alpha$ and $X=\ln(x)$, then equation (16) can be rewritten as follows:

$$Y = A + B \cdot X. \tag{17}$$

The coefficients A and B are calculated by the least squares method.

Fig. 19 shows a graph of the required number of LUTs versus the number of GAN layers on a logarithmic scale.

In Fig. 19, the points correspond to the experimental data, and the red line corresponds to the graph of the approximating function according to formula (17).

As can be seen from Fig. 19, the dependence of the required number of LUTs on the number of layers on a logarithmic scale is indeed linear. Knowing the coefficients, A and B, we can easily find the values of the parameters c and α in (15). Our calculations show that $c \approx 1383.33$ and $\alpha \approx 2.02$. In this case, the required number of LUTs for GAN networks can be calculated using the formula:

$$z = 1383.33 \cdot x^{2.02}. \tag{18}$$

Calculations using formula (18) show that GAN networks with a maximum number of layers equal to 42 can be placed on the HAPS s-52 FPGA.

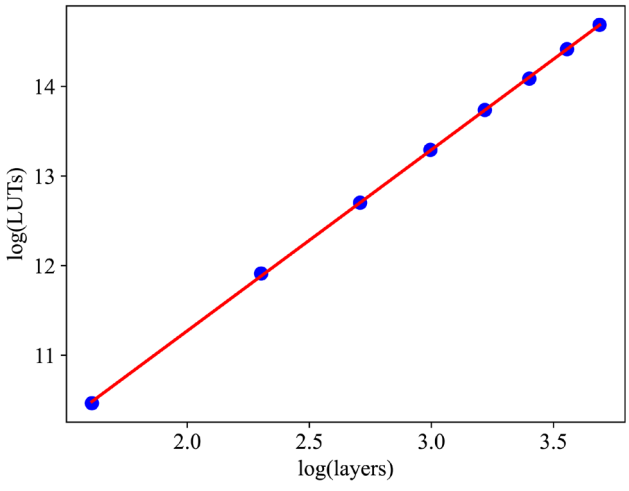

Fig. 19. Dependence of the required number of look-up tables on the number of layers on a logarithmic scale for generative adversarial networks

5. 2. Conducting a comparative analysis of the computational resource intensity of different types of artificial neural networks

As our calculations show, LSTM networks turned out to be the most demanding for FPGA computing resources. Fig. 20 shows graphs showing the boundaries of the permissible parameters of SNN, MLP, and LSTM networks when they are placed on the FPGA of the HAPS-80 S52 system. In Fig. 20, the area above the curves corresponds to the values of the admissible parameters of the neural network. From this figure, we see that the range of valid parameters for the LSTM network has the narrowest width.

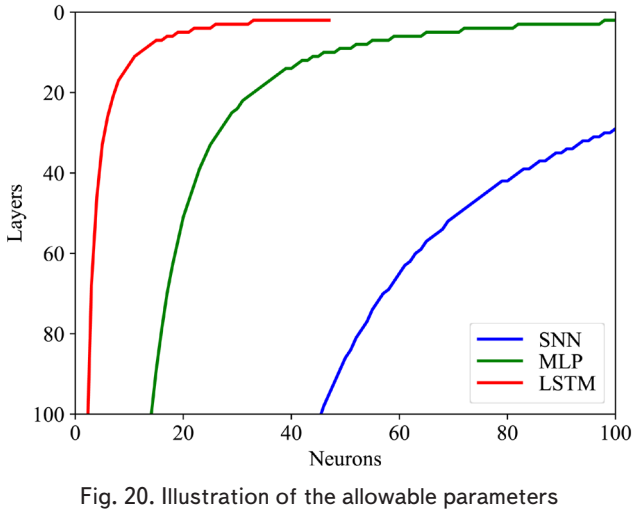

Fig. 20. Illustration of the allowable parameters of spiking neural network, multilayer perceptron and long short-term memory networks when they are placed on the field-programmable gate array of the HAPS-80 S52 system

From Fig. 20 we also see that the SNN turned out to be the least demanding on the computing resources of the FPGA. As stated above, in the case of GAN and CNN networks, the required number of LUTs depends on only one parameter – the number of network layers. And if we compare these networks with each other, then CNN requires fewer FPGA resources.

6. Discussion of the results of processing experimental data

As a result of the research, we obtained an empirical dependence of the required number of LUTs on the parameters of neural networks. The parameters of neural networks in this case are the number of layers and the number of neurons in each layer. The form of the dependencies obtained turned out to be different, for example, for networks of the CNN type, this dependence is a linear function, the argument of which is only one parameter – the number of layers. And for other types of MLP, LSTM, SNN, and GAN networks, this dependence is described by a non-linear function, the argument of which are two parameters – the number of layers and the number of neurons in the layer. And at the same time, the function that describes the changes in the GAN network has only one parameter – the number of layers. All found nonlinear functions have the same form – a power function.

Compared with the results of other works [1–10], this work allows us to determine the number of LUTs from the empirical dependence obtained as a result of an experiment on the implementation of neural networks such as MLP, CNN, LSTM, SNN, and GAN. Other works are intended to implement a neural network model for solving a specific problem, and not to implement the entire neural network on an FPGA.

The development of this study lies in the fact that the use of FPGAs for the implementation of ANNs can increase performance and reduce energy consumption due to their parallelism for computing and their structural features compared to other digital devices. The study may encounter experimental difficulties, since large-scale calculations of LUT resources will require more time to synthesize the recording circuit in the FPGA.

When choosing an FPGA, the main issue is to determine the sufficiency of its resources to perform a certain number of calculations in parallel mode. We believe that the empirical formulas obtained will help answer this question since they provide an opportunity to make an approximate estimate of the required number of LUTs for implementing a certain neural network on an FPGA.

We should draw your attention to the fact that when carrying out experimental calculations, we adopted some restrictions, for example, the data has a size of 24 bits, and only LUT resources are used to store data and perform operations. It is known that in addition to the LUT, other FPGA resources can be used for data processing, for example, DSP, but the amount of these resources compared to the LUT resource is usually very small. Accordingly, when choosing a different data size and deciding to additionally use other FPGA resources, the end result in the form of an estimate of the required number of LUTs can be quite different. Nevertheless, we believe that the empirical formulas (9)–(11), (13), (18) we propose can be used to estimate, as a first approximation, the required number of LUTs for implementing a certain neural network on an FPGA.

The disadvantage of this study may be the static value of the data size. Changing them can affect the amount of LUT resources. The choice of data bit depth depends on the specific model, that is, on the weight coefficients of the network. However, by choosing this size, you can achieve excellent fixed-point data accuracy for implementing a neural network on an FPGA.

It should be noted that these patterns are not universal. For example, they naturally depend on the bit depth of the representation of numbers in the FPGA and on the algorithm for implementing neural calculations. Obviously, in the

case of an increase in the number of operations performed sequentially, the requirements for the number of LUTs can be reduced. But in this case, there will definitely be a loss in computing performance. In our work, we tried to achieve maximum computing performance on the FPGA by using parallel execution of operations as much as possible. Accordingly, the established empirical patterns to some extent reflect the limiting case of the resource use of FPGAs for the implementation of neural networks on them.

7. Conclusions

1. Based on the results of experimental measurements and processing of experimental data, empirical patterns were found that can be used to approximately estimate the number of required LUTs needed to place ANNs (SNN, MLP, LSTM, CNN, GAN) of different structures on the FPGA. These patterns, presented in the form of specific analytical records of functions, can be used to select an FPGA and plan the structure of neural networks in such a way that the computing resources of the FPGA correspond to the computational requirements of neural networks.

2. A comparative analysis of the computational resource intensity of different types of ANNs showed that it is most efficient to place neural networks of the SNN and CNN type on the FPGA. The GAN neural network, whose function depends on one parameter, uses almost 100 % of the FPGA LUT with a size of 41 layers. And with a similar number of layers, the CNN network uses only 5.52 % of the LUT. And when comparing neural networks, the function of which has two arguments, it was revealed that the LSTM network with 11 layers and 11 neurons in each layer uses almost 100 % of the FPGA LUT. Compared to LSTM, SNN, and MLP networks use only 1.22 % and 6.85 % of LUTs in a similar situation, respectively. Neural networks like GAN and LSTM turned out to be very demanding on FPGA computing resources.

Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

The work was carried out within the framework of the project AP19677321 «Development of digital experimental setups for studying physics phenomena in laboratory conditions of educational institutions using modern computer technologies».

Data availability

The manuscript has no associated data.

References

1. Ádám, N., Baláž, A., Pietriková, E., Chovancová, E., Feciľak, P. (2018). The Impact of Data Representationson Hardware Based MLP Network Implementation. Acta Polytechnica Hungarica, 15 (2). doi: https://doi.org/10.12700/aph.15.1.2018.2.4
2. Gaikwad, N. B., Tiwari, V., Keskar, A., Shivaprakash, N. C. (2019). Efficient FPGA Implementation of Multilayer Perceptron for Real-Time Human Activity Classification. IEEE Access, 7, 26696–26706. doi: https://doi.org/10.1109/access.2019.2900084
3. Westby, I., Yang, X., Liu, T., Xu, H. (2021). FPGA acceleration on a multi-layer perceptron neural network for digit recognition. The Journal of Supercomputing, 77 (12), 14356–14373. doi: https://doi.org/10.1007/s11227-021-03849-7
4. Bai, L., Zhao, Y., Huang, X. (2018). A CNN Accelerator on FPGA Using Depthwise Separable Convolution. IEEE Transactions on Circuits and Systems II: Express Briefs, 65 (10), 1415–1419. doi: https://doi.org/10.1109/tcsii.2018.2865896
5. Zhang, N., Wei, X., Chen, H., Liu, W. (2021). FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection. Electronics, 10 (3), 282. doi: https://doi.org/10.3390/electronics10030282
6. He, D., He, J., Liu, J., Yang, J., Yan, Q., Yang, Y. (2021). An FPGA-Based LSTM Acceleration Engine for Deep Learning Frameworks. Electronics, 10 (6), 681. doi: https://doi.org/10.3390/electronics10060681
7. Shrivastava, N., Hanif, M. A., Mittal, S., Sarangi, S. R., Shafique, M. (2021). A survey of hardware architectures for generative adversarial networks. Journal of Systems Architecture, 118, 102227. doi: https://doi.org/10.1016/j.sysarc.2021.102227
8. Wang, D., Shen, J., Wen, M., Zhang, C. (2019). Efficient Implementation of 2D and 3D Sparse Deconvolutional Neural Networks with a Uniform Architecture on FPGAs. Electronics, 8 (7), 803. doi: https://doi.org/10.3390/electronics8070803
9. Han, J., Li, Z., Zheng, W., Zhang, Y. (2020). Hardware implementation of spiking neural networks on FPGA. Tsinghua Science and Technology, 25 (4), 479–486. doi: https://doi.org/10.26599/tst.2019.9010019
10. Ju, X., Fang, B., Yan, R., Xu, X., Tang, H. (2020). An FPGA Implementation of Deep Spiking Neural Networks for Low-Power and Fast Classification. Neural Computation, 32 (1), 182–204. doi: https://doi.org/10.1162/neco_a_01245
11. Medetov, B., Serikov, T., Tolegenova, A., Dauren, Z. (2022). Comparative analysis of the performance of generating cryptographic ciphers on the CPU and FPGA. Journal of Theoretical and Applied Information Technology, 100 (15), 4813–4824. Available at: http://www.jatit.org/volumes/Vol100No15/24Vol100No15.pdf
12. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., Bharath, A. A. (2018). Generative Adversarial Networks: An Overview. IEEE Signal Processing Magazine, 35 (1), 53–65. doi: https://doi.org/10.1109/msp.2017.2765202