functions, the segment of valid argument values is divided into a fixed number of parts (the accuracy is controlled by parameters). For each part, the value of cosine and sine is calculated and stored in a special table. The acceleration of calculations is achieved in the many times repeated use of tabular approximate values instead of the full calculation of the function value.

To train all non-recurrent networks with gradient methods, the back-propagation algorithm is used to calculate the gradient. For restricted Boltzmann machines, the gradient calculation uses the Constructive Divergence (CD-1) algorithm. Then, the user-selected gradient modifications are applied to the result.

The user interface contains a fairly wide range of functionality, covering all the procedures necessary for convenient framework use. This is loading (saving) a neural network from a hard disk, constructing neural networks based on the architecture base, forming a training dataset based on selected objects classes and data. Neural networks training process implementation, using various types of algorithms. Solving applied problems, color image compression and object recognition. The neural networks training process can be described as follows. First, the user loads the training data from the database and calls the building training dataset methods (if needed). It then declares the neural network skeleton on which the neural network will be built. To do this, the framework implements a special fakeDeepNN data type. Using the prebuildDeepNN method, the user tells the designed neural network the images resolution, the images type (black and white, grayscale or color), indicates whether the network will be convolutional and assigns (if necessary) the future network a serial number.

Then, a neural network is constructed to the created skeletone (by sequentially adding different types of layers). The need for layer decomposition is indicated when adding the first layer of the network. In other layers, the decomposition size is calculated automatically based on the layer sizes entered by the user.

At the end of the neural network design process, the user declares an object of deepNN type and calls the buildDeepNN method. Passes the previously created network skeleton as a parameter. This function, in addition to connecting individual fragments into a single network, checks the correctness of the constructed network. For example, the incompatibility network layers' sizes, the incompatibility of layers' types (for example, one layer generates continuous data, and the next layer receives discrete data, or vice versa). If an error is found, the program generates the appropriate error and exits.

In the next step, the user sets the neural network training settings. The framework is highly flexible and allows for many different training modes. To do this, it declares an object of a special type TrainingSettings and, by calling the object's method for adding settings for a separate layer, sequentially sets the training settings for all layers of the network. In the training settings of a separate layer, the following options are set: optimization algorithm, training style, layer training time, number of objects in the training and validation sets.

Setting the training style allows you not to train individual neural networks' layers, but to load network's trained fragments from the hard disk. The framework also supports the partial training option. To do this, the training style indicates: simple training or initial training is carried out — after which additional training of the layer is allowed, or continuation of training — the layer continues training. This option is useful when training large neural networks or when using low-power computing devices.

In the case of initial training, in addition to the trained network, the training state of a separate layer is stored. The description of the learning state depends on the optimizer chosen. In the case of continuing training, in addition to the initial training, the saved learning state of the neural network layer and the partially trained layer are loaded.

The training time is set separately for each layer. Inside all optimizers, there is a built-in implementation of the timer function that controls the time spent on training the layer. As soon as the time allocated for training has expired, the training process for this layer is completed. The sizes of the training and validation sets allow us to control the amount of data required for layer-by-layer neural network training.

After creating a neural network and setting up the train- ing of its individual layers, the user calls the constructed network training function using the trainDeepNN function. With this function, the user informs about the network being trained, training settings, some information about the input data and the need to use external devices for training.

The trainDeepNN function is key to the framework. At the beginning of the execution, if necessary, it scans the connected computing devices and initializes the executable modules on them. The function then checks that the the network layers training settings are correct. For example, it checks for the trained layers presence, that do not require training or will be retrained, etc. This function loads all settings of the entire algorithms library and optimizer selected by the user. Inside this function, a complex interaction with the input data for their decomposition (if necessary), linking the layers of the trained network, creating and deleting service buffers for the optimizer calling the training of the corresponding layer type by the corresponding optimizer, and other technical details are implemented. This method, at the end of training, saves the trained network to the hard disk.

The user can also call the compressImages and predict functions. Both functions take a trained neural network

and input data. The first of them performs data compression and returns their compressed image, the second returns a set of labels. The number of labels corresponds to the number of sended images. The label specifies the object class that the neural network has detected in the image.

IV. Architectures library

The architectures library fully fits into the ostis system described in [2] [3]. The software package supports the following types of neural networks: restricted Boltzmann machine of Gauss-Bernoulli and Bernoulli-Bernoulli types, autoencoders, decomposition of network layers from the above types, multilayer perceptrons, convolution layers, pooling layers.

A small number of frameworks support restricted Boltzmann machines, although they are used for key frames detecting in video sequences [4] filtering data [5], encoding key phrases in information retrieval [6] [informational retrieval]. Based on supported autoencoders and restricted Boltzmann machines, the framework allows us to design deep belief networks for data compression and preprocessing for subsequent data classification.

Support for sampling and convolution layers, multilayer perceptrons allows us to design deep convolutional neural networks to build neural network classifiers that are used for a wide class of applied problems.

Thanks to the support for the decomposition of individual layers, the framework allows you to significantly reduce the number of tunable network parameters, reduce the required amount of data for training, and increase the efficiency of parallelizing the neural network layers training.

The supported wide library of architectures allows us to design neural networks of almost any architecture, which makes the framework a universal tool for neural network data processing.

V. Training algorithm based on the annealing method

To solve the problem of efficiency of neural networks training algorithms, there are algorithms based on random search. The annealing method is the most promising random search algorithm for training neural networks. The paper proposes the following training algorithm based on the annealing method.

Let an objective function F be defined on a finite set of admissible solutions Ω and for each element x $\subset \Omega$ of which the set of neighboring elements is N(x) $\subset \Omega$ given. The conditional optimization problem in this case can be specified as a triple (Ω, F, N). Let us consider the possibilities of its solution using the annealing method. The algorithm includes the following main steps.

Preliminary stage. Initialization of the neural network initial state $Net_0 = \mathrm{Net}(x_1 0, x_2 0, ..., x_{m0})$ and temperature sequences $T_0, T_1, ..., T_k$, related by the ration:

$$T_k = \frac{T0}{ln(k+2)}, k > 0,$$

where T_0 - present value.

General k-th iteration.

Step 1. Random value generation. Generated m uniformly distributed on the segment from zero to the number of parameters in the set of discrete random variables $a_1, a_2, ..., a_m$. Generated m random permutations of length equal to the number in the set of parameters. The first $a_1, a_2, ..., a_m$ elements of each permutation define the indexes of the parameters to be changed in each set of parameters, respectively.

Step 2. New solution generation. For each changing parameter, a uniformly distributed on the segment [-l/2;l/2] random value b is generated. Value l depends on what set is changing parameter belongs to and equal to $l_1, l_2, ..., l_m$ respectively. Values l for each set are given as algorithm parameters.

Let x_i is changing parameter and $x_i^{'}$ is its new value, then:

$$x^{'i} = x_i + b$$

Step 3. Transition principle. Let x be a current solution and y was generated on step 2 as new solution. Then solution $x^{'}$ on the next iteration is determined in a following way:

$$P(x' = y \mid x) = \min\left\{1, \exp\left(\frac{F(x) - F(y)}{T_k}\right)\right\}$$

Step 4. Stop criteria. If the time for training the neural network has expired, then the algorithm ends. Otherwise, the transition to the next iteration is performed.

Previously, it was proved that the proposed algorithm converges in probability to the optimal solution, and from any initial solution [7].

VI. Genetic algorithm for neural networks training

Additionally, a special genetic algorithm modification for neural networks training was developed for the framework.

Preliminary stage. Generation of several random solutions. Each individual solution is a complete neural network, the architecture of which is set before training algorithm running and doesn't change. The number of solutions N is a parameter of the algorithm. For each solution, the value of the quality functional to be optimized is calculated.

General k-th iteration.

Step 1. The worst one is chosen from the current set of solutions.

Step 2. For the chosen solution, a "mutation" is performed – generation of a new solution from the current

one according to the same scheme as for the developed annealing algorithm. The only difference is that the values of the annealing parameters may differ from the genetic algorithm.

Step 3. For the obtained solution, the quality functional value is calculated.

Step 4. If the value of the obtained functional for the new solution is better than for the current one, then the current solution is replaced with a new one, otherwise the new solution is discarded.

Step 5. The solution b is randomly selected from the set of current solutions. The best solution a is also selected from the set.

Step 6. "Crossbreeding". For all values of the solution parameters a, b, the following calculations are made.

$$\begin{cases} d_i = b_i - a_i, \\ C_i = a_i + d_i * a, \\ \alpha \in [0; \phi], 0 < \phi \leq 1. \end{cases}$$

where α is uniformly distributed random variable on the segment, ϕ is an algorithm parameter.

Step 7. For the obtained solution, the quality functional value is calculated.

Step 8. In the set of solutions, the worst one is chosen. If it is worse than the new solution, then it is replaced by a new solution, otherwise the new solution is discarded.

Step 9. Stop criteria check. If the time for training the neural network has expired, then the algorithm ends. Otherwise, the transition to the next iteration is performed.

VII. Software package implementation features

The base of algorithms is developed taking into account the cross-platform framework property. When developing optimizers, the computing devices architectural features were taken into account. For example, video cards are characterized by a large number of low-power computing cores, which requires good scalability from the parallelization algorithm. In addition, the performance of the video card mainly depends on the efficiency of working with video memory, since it is very slow. This imposes requirements on the locality of the data used for the most efficient video card's cache use. It was also necessary to implement the interaction of the video card with the processor for the results collection and other data transfer.

The above requirements have resulted in each optimizer being implemented twice using different algorithms. One set of algorithms is used to train neural networks on a processor, the other set is used on video cards.

The most time-consuming part of training a neural network is moving data through the training layer. The input data is presented in matrix form, as are the parameters of the neural network. In a simplified form, moving data through a network layer is a matrix multiplication of data by network parameters. Thus, to ensure efficient training, it is necessary to implement efficient matrix multiplication.

To ensure the best data caching, a block data multiplication algorithm was used. It is worth noting that improving caching increases the training efficiency both on the video card and on the processor. The block size for the matrices was selected empirically. For the processor, the optimal block size was 25, for the video card — 16. It should be noted that the data size, the amount of data, and the network size are often not a multiple of these values, so incomplete blocks were filled with zeros at the end of the real data. This approach makes it possible to increase the training efficiency by more than 70processor and more than twice on the video card. This fact makes it possible to compensate for the significantly complicated implementation of optimizers and the entire complex as a whole due to the formation and disbanding of data blocks and work inside data matrix blocks.

It is known that random search methods have slow convergence and require a large amount of computation at a single iteration. Therefore, in order to increase the efficiency of annealing training, a calculation hiding approach (when using connected devices for training) was used.

Each iteration of the annealing method, as mentioned above, consists of 4 stages. All stages, except for calculating the functional value for a new solution, are performed by the processor. The most time-consuming stage is the calculation of the functional value and, with a small network architecture, the new solution generation. The computational power of a video card is on average 20 times higher than that of a processor. This leads to the fact that when training small neural networks, a significant part of the time is spent on generating new solutions. All stages in a separate iteration are performed strictly sequentially.

The generation of a new solution consists of two stages: determination of the number and selection of changeable network parameters; changing the values of the selected network parameters.

To reduce the execution time of a single iteration, you can use the fact that the definition and choice of network parameters to be changed don't depend on the stage of making a new decision. This can be explained by the fact that the generated increment to the values of the changing parameters will not change in this case. Changing the solution changes only the initial values of the parameters being changed. In this case, you can use the well-known calculations hiding trick.

To achieve the most efficient use of computing devices, 3 special procedures were designed.

Procedure A1. You can determine the number and select the parameters to be changed and their increments in parallel on the processor, when the video card calculates the value of the quality functional. Under such conditions,