Journal of Information Technology Management
ISSN-OnLine: 2980-7972
University of Tehran

Visual System for Configuring Machine Learning Models to Support IT Management and Decision-Making

Vitalina Babenko*

*Corresponding author, Prof., Kharkiv National University of Radio Electronics, Kharkiv 61166, Ukraine; Kharkiv National Automobile and Highway University, Kharkiv, 61002, Ukraine. E-mail: vita.babenko@gmail.com

Andrii Brazhnykov

8440, Creekside green dr, apt 5302, Spring, TX, 77389, USA. E-mail: Brazhnicov@gmail.com

Nataliia Gavkalova

Professor, Institute of Production Systems Organization, Warsaw University of Technology, Warsaw, 02-524, Poland. E-mail: nataliia.gavkalova@pw.edu.pl

Serhii Rudenko

Associate Professor, State Biotechnological University, Kharkiv, 61002, Ukraine; Kharkiv National University of Internal Affairs, Kharkiv, 61080, Ukraine. E-mail: sr7000388@gmail.com

Marianna Oliskevych

Professor, Ivan Franko National University of Lviv, Lviv, 79000, Ukraine. E-mail: olisk@ukr.net

Olena Fridman

Associate Professor, V. N. Karazin Kharkiv National University, Kharkiv, 61002, Ukraine. E-mail: o.fridman@karazin.ua

Dariia Babenko

Senior Lecturer, O. M. Beketov National University of Urban Economy in Kharkiv, Kharkiv, 61002, Ukraine. E-mail: dary.babenko@gmail.com

Journal of Information Technology Management, 2025, Vol. 17, Issue 4, pp. 117-135
Published by the University of Tehran, College of Management
doi: https://doi.org/10.22059/jitm.2025.105486
Article Type: Research Paper

Received: June 14, 2025
Received in revised form: August 26, 2025
Accepted: September 14, 2025
Published online: November 01, 2025

Abstract

Deep learning models have become indispensable across scientific and business domains, offering new approaches to problem-solving but requiring substantial technical expertise for their implementation. This article presents StudySupport, an open-source visual system for configuring and training machine learning models via a graphical interface rather than traditional coding. The system enables users to manage the entire pipeline - from data preprocessing and model construction to optimization and performance evaluation - while maintaining flexibility for advanced customization. By lowering the technical entry barrier, the StudySupport system facilitates the adoption of machine learning in IT management and organizational decision-making. The proposed framework supports faster integration of data-driven methods into enterprise information systems, reduces implementation costs, and empowers managers, analysts, and educators to leverage artificial intelligence in digital transformation processes. The study contributes to the field of information technology management by bridging the gap between advanced machine learning techniques and their practical application in business, education, and decision-support systems.

Keywords: Visual System, StudySupport system, Machine Learning, Decision-support Model, IT Management, Decision-Making

Introduction

Over the past few years, artificial intelligence and neural networks have permeated almost every aspect of life. And although theoretical calculations in this field have been conducted for a long time, the opportunity to develop practical programs emerged relatively recently, when, in the early 2000s, sufficient data accumulated and computing power and its cost reached a certain level.

Neural networks are used very widely, you directly interact with them when you search for something in a search engine, find out the route to the point you need (taking into account traffic jams, of course), check the weather forecast for the coming days, enter a parking lot with license plate recognition, and if you are lucky, you can even talk to a neural network. These are the simplest examples that we encounter every day. Neural networks are no less widely used in professional fields: sociology, journalism, linguistics, and many other areas (Amershi et al., 2019; Babenko, 2013, 2020; Gontareva et al., 2020; Kyrylieva et al., 2023).

At the same time, almost all modern web services, computer programs, and mobile applications are based on a graphical interface. The next most popular interface is the command line, which is used only in specific areas, most often related to programming (Andrienko et al., 2022). This is quite logical, given that working in the command line itself is, in many ways, programming, but not everyone has this skill, because acquiring it requires several months, if not years, of painstaking study of the subject.

Despite the high demand for neural networks, the only way to build them remains programming. In this paper, we explore the possibilities of integrating an alternative way of working with neural networks - a graphical interface- and also try to understand why such programs did not appear earlier.

Literature Review

Modern research in machine learning indicates a growing need for tools that simplify the process of building and tuning models for users without deep programming training. A significant part of the work (Huang et al., 2022) focuses on developing graphical user interfaces (GUIs) that enable visual construction of neural networks. Such solutions make the process of tuning architectures more intuitive and accessible, especially for educational and research purposes.

A significant contribution to the development of such systems has been made within the framework of the Barista project (Klemm et al., 2018), which provides an environment for designing and training deep neural networks using flowcharts. Similar approaches are also actively used in the educational environment - for example, a tool for teaching the basics of convolutional neural networks in medicine (O'Shea & Nash, 2015).

Some research focuses on creating explainable systems that not only simplify model tuning but also aid in interpreting their results (Shorikov et al., 2014; Baldi & Sadowski, 2013). This is especially relevant for critical industries where transparency of the decision-making process is required (Babenko et al., 2018; 2020).

However, most existing tools have limited functionality: they are either focused on a narrow range of tasks or do not allow the integration of new architectural elements without code. This lays the groundwork for further research toward universal systems (Kashchena et al., 2024) that combine the flexibility of traditional programming with the ease of use of graphical interfaces.

Working with data. There are many ways to store data. Each of them requires its own Dataloader that accounts for their specific storage (Hinton et al., 2012). However, in practice, in most tasks, only a few different types are encountered (Corbató & Vyssotsky, 1965**)**. Here are some of them:

- Both input data and markup are stored in the same text file,
- Data and markup are stored in different files,
- The data is a set of files (e.g., images or documents), and the markup is stored in a separate text file. and

- Data is a set of files whose location determines the markup (for example, images of different classes are in different folders).

The data type also needs to be considered, as different types may require different processing (He et al., 2016). For example, if an image needs to be reduced to a specific size before being processed by the model, then for text documents, such a transformation is undefined. However, the storage methods may coincide (Engel, 1988). The most common types can also be briefly listed:

- Numerical characteristics (actual, integer), strings with a small number of values (classes), as well as their lists:
- Image
- Video
- Text documents
- Audio

The Dataloader device is also affected by the task type, which determines the markup type (Michael & Maynard, 2003). There are not so many of them anymore, although new ones appear regularly.

Methodology

The global goal of this study is to create open-source programs that allow creating and configuring any deep learning models with a teacher from scratch, without requiring anything beyond a dataset. However, profound teaching has been rapidly developing for more than a dozen years now. During this time, many tricks and tools have been invented, and implementing them all requires a lot of time, since all the code is written by the author alone. Therefore, the task at hand is to create a prototype of a universal application with a graphical interface, demonstrating key opportunities, adding most of the tools, and identifying areas that need additional processing.

Solving each machine learning problem can be divided into three semantic parts that are solved practically separately: creating a dataloader class that provides data in the desired format, writing neural network models that accept input data from the dataloader, and selecting metrics and analyzing their graphs and results. Consider each subtask in more detail.

Also, because predicting all possible ways to store data and task types won't work (at least because new task types appear), in this work, let's add the possibility for the user to load their own dataloader into the program using a built-in text editor.

Results

Creating a model

A deep learning model is a complex, nonlinear function composed of simple, basic functions - layers. Among them, nonlinearities and regularization are usually treated separately (Corbato & Vyssotsky, 1965), but in this study, we treat them on a par with ordinary layers, such as linear, convolutional, etc.

Neural networks can be represented as a directed graph, with some "special" vertices, incident edges entering "from nowhere" or "to nowhere"; this is how neural networks are usually visualized in scientific works (Savytska et al., 2023, 2024). Edges “from nowhere” are the very first layers that receive input data from the dataloader, and edges “to nowhere” feed their output into the loss function.

This visualization is really very convenient and makes it easy to understand how the model is arranged, if you know how each layer is arranged separately (Hrabovskyi et al., 2020). This is what the model editor in our program looks like: On the canvas, which can be moved, rectangles-layers are placed, and by selecting them, you can adjust the parameters, look at the dimensions of the input and output, or study the device in more detail by reading the description of the layer.

By analogy with the previous section, in order for the application to be as flexible as traditional model programming, it is possible to add the function of creating your own layers, loss functions, and optimization algorithms (Pilavakis, 1989). To do this, you need to replicate the dataloader functionality, adjusting it to the specific details. However, in this work, this function will not be added: the principal possibility of adding it has been demonstrated, and implementing the function would have to be at the expense of other capabilities of the prototype.

Interpretation of metrics

Training a neural network is the task of optimizing a function, namely the loss function, chosen based on the task. Most often, the same loss functions are used, for example, for classification or regression tasks; the squared mean or cross-entropy is often used (Wagner, 2022). Therefore, it is necessary to add as many "popular" loss functions as possible to the program library so that they can be used without programming.

After choosing the loss function, the neural network training begins. Most often, if mistakes are made somewhere and the model is trained incorrectly, this becomes clear from the behavior of the loss function or other metrics (Shtal et al., 2023). Therefore, during

training, it is almost always prescribed to build graphs for various metrics (Donnat & Holmes, 2018). Our program also automatically constructs and updates graphs during training.

Justification

A typical implementation of a learning algorithm in PyTorch looks like this: (Algorithm 1).

Algorithm 1 Schematic of the neural network training algorithm on PyTorch

```
dataloader =
dataloader _ init ( )
model = model_init
() loss = loss_init ()
optimal = optimal_init (
model.params ) for data
in dataloader out =
model(data) loss_value =
loss(out)
loss_value.backward ()
optim.step ()
update_plots (
loss_value ) end for
```

Logistic system congestion is a significant problem in many cities worldwide. It poses a name. Note that the initializations of the objects here do not depend on each other in any way (except for the optimizer object, the only limitation of which is the need to be initialized after the initialization of the neural network model), which allows you to configure them independently of each other. It is precisely this property that gives the graphical interface great potential: each algorithmic detail can be configured separately in its own window, tailored to its specific information.

In addition, the graphical interface has many advantages over the traditional approach. Here are some of them: challenges and negative consequences for individuals, communities, and economies. Here is a deeper look at the problem of traffic congestion:

Friendly error messages

The graphical interface also allows you to interact with the model before the model is ready for use (Kuznetsov et al., 2019; Nesterenko et al., 2024). This approach allows the output of information about the model to be generated directly during construction (for example, the total number of model parameters, the size of the tensor produced by each layer, etc.) (Pietrołaj & Blok, 2024). This information also helps identify search errors, allowing them to be corrected before launching neural networks.

Except for error prompts, the program may give tips on use. For example, in which cases worth using this or that other layer of the neural network, nonlinearity, or regularization, to provide references to theoretical justification, as well as describe hyperparameters, what they affect, and how to set them correctly to configure.

Small entry threshold

When done correctly, the advantages described above allow a much lower entry threshold. The absence-necessity program already does this, and, if available in the program, references and explanations reduce the amount of necessary technical knowledge, because some essential information can be found in the app itself.

Absolutely, an unprepared user is unlikely to cope with writing non-trivial neural networks, but understanding the device will be much easier if you don't have to get distracted by programming, and all the necessary theory is in one place.

About the program

The StudySupport system provides practical value for IT management by lowering the barrier to adopting AI methods in decision-support environments. Organizations can integrate machine learning models into enterprise systems such as ERP, CRM, or data analytics platforms without requiring specialized programming expertise (O'Shea & Nash**,** 2015). This facilitates more rapid prototyping, reduces implementation costs, and enhances the role of managers and analysts in driving digital transformation. In educational contexts, the system can also be used to train future IT professionals, bridging the gap between theoretical AI concepts and applied management practices. Window creation and discovery projects

At launch, programs open a start window that allows users to create new projects or open existing ones. The project is represented in specific directories by a folder containing the project configuration file, project config.json, which includes information about the project. Upon creation, a new project generates a corresponding file, leaving the main configuration file, main_config.json. For works with file configurations, a special Config class was created. It is arranged so that any parameter changes are automatically saved to the corresponding file.

Project window

The project window is divided into two parts: the main and the container menu. More of the screen is occupied by the main, which is divided into five tabs: Settings, Layout Data, Data, Model, and Training. One tab is active at any given time. On the right side of the screen, the menu container contains the latest menu object the user interacted with.

For convenience, two classes were implemented: MenuWidget and WidgetWithMenu. For each widget that inherited from WidgetWithMenu, you can realize your class that

inherited from MenuWidget and pass it as a parameter when calling the super functions, along with an instance of the class container. Then, when calling the activate_menu method in the corresponding menu container, the menu of this widget will be displayed. Yes, each tab has its own menu and when switching tabs, the menu of the one that was open.

"Settings" tab

Allows changing some values in the project configuration file.

The "Markup" tab data »

In this tab, you can mark up an image for detection objects.

"Data" tab

Has two divisions: Loader Data and Reprocessing

"Loader" tab data »

Loader data is built into a Python syntax-highlighted text editor. Traditionally, to create a data loader, a programmer creates a special class that inherits from the Dataset class in torch. Utils.data module. This class has to determine such methods:

- __ init __ - the method that is responsible for initialization data
- __ getitem __ is a method that returns an element dataset by index
- __len__ is a method that returns the size of a dataset

"Recycling" tab

For augmentation, there is a Preprocessing tab. In the left pane of the main screen, a raw image is displayed, which can be selected by clicking the corresponding button. On the right, the posted image after processing. Between them is a space where applicable transformations are located. Transformation can be added or removed using the corresponding buttons in the menu.

Currently, such a transformation:

- InvertImg - Applies a "negative" to an image, subtracting the value of each pixel out of 255
- Equalize - Approaching distribution values pixels to a uniform size, thus increasing contrast
- CLAHE - Applies local adaptive histogram
- ToSepia - Applies filter sepia
- GaussianBlur - Applies a Gaussian filter

- HueSaturationValue - Randomly changes the hue, saturation, and brightness of each pixel.
- RandomContrast - Randomly changes the image contrast
- Resize - Changes the size of the image

When choosing a widget for each conversion in the menu on the right, the relevant settings parameters appear. Clicking the "Refresh" button will run the incoming image through the provided transforms and update the transformed image.

To save the current dataloader and the applied layers, reprocessing is necessary. Click on the corresponding button at the very bottom of the screen

Model Tab

Contains two subtabs: Designer and Options learning. The first is intended to create neural networks, and in the second, we can choose function errors and optimization algorithms.

"Designer" tab

On the main screen, the canvas dimensions prevent it from being fully displayed, so at any given time, only a portion is shown (Jin et al., 2022). This part can be moved using the mouse or sliders located at the right and bottom of the canvas. The tab menu contains buttons that allow creating layers of neural networks (Fig. 1). By default, on the canvas, incoming and outgoing layers are located, the first one passes the neural network data, and the second receives the result of the work as input for the neural networks; unlike other layers, it's impossible to delete them.

The layer appears as a rectangle on the canvas with a fixed size. In the middle, the text is located in the type < layer type> _ <id>. In the upper and lower parts of the layer are a certain number of buttons, depending on the layer type. The top buttons represent incoming layer data, and the bottom represents the weekend. To create communication between the two layers, first press one of the output buttons in the first layer, then the input button in the second layer. After this, draw a line between the corresponding buttons to denote the connection between the layers. Each input button can accept at most one connection; thus, between the connective lines and the input buttons, a single communication is enough to select the appropriate button. Each layer stores information about which layers it is connected to, which is why the model is built in linear time.

Each layer can be moved relative to the canvas by clicking on it and holding down the left mouse button while moving to the desired location. When selecting a layer in the menu container located on the right, it opens the settings menu parameters for this layer. At the moment, the program is built with the following types of layers :

1. Layers – functions activations (Non-linear activations) (Zhang et al., 2016; Zhang & Zhao, 2024).

Such layers receive on Exit tensor $X = (x_{and})$, *and* ∈ N^{k} and return tensor ($f (x_i)$), *and* ∈ $N^{k,}$ where *f* - function activation, also called nonlinearity

- ReLU - $f (x) = \max (x, 0)$
- Sigmoid - $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

2. Convolutional layers (Wang, 2025). Apply to the input tensor *an n-dimensional* convolution. The input tensor can have dimensions *n* + 1 or *n* + 2. Passes through a window of size ((1), $C_{in,} k_{1,} k_{2,}.., k_n$), also called the kernel over the entire input *tensor* along with offsets of size $_(0)$, 0, *p1*, $p_{2,}.., p_n$) with a step ((1),,,,,, applying a linear transformation of the window value. Gaps depending on the axis ((0), can be added between the kernel elements. 0, $d_{1,} d_{2,}.., d_n$) If the input is a tensor of size ((*N*), $C_{in,}$ *D1, D2,.., Dn*), the output tensor will have $((N), C_{out}, D'_1, D'_2, \ldots, D'_n)$, где D'_i:

$$D'_i = \left\lfloor \frac{D_i + 2 \cdot p_i - d_i \cdot (k_i - 1) - 1}{s_i} + 1 \right\rfloor \quad (1)$$

Parameters :

– in _ channels (C_{in}) - Number of channels in the input tensor

– out _ channels (C_{out}) — quantity channels output tensor

– kernel_size * ($k_{1,}.., k_n$) - size kernels

– stride * ($s_{1,}.., s_n$) - kernel step for each axis

– padding * ($p_{1,}.., p_n$) - Dimensions indents for each axis

– padding _ mode - method filling indentations. Maybe take four values :

∗ *zeros* - Fills indents zeros :

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

∗ *reflect* - Reflects value indents of border tensor

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 2 & 3 & 2 & 1 \\ 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

∗ *replicate* - each value indentation equal to the nearest element from the main tensor

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 3 & 3 \\ 1 & 1 & 1 & 2 & 3 & 3 & 3 \\ 4 & 4 & 4 & 5 & 6 & 6 & 6 \\ 4 & 4 & 4 & 5 & 6 & 6 & 6 \end{pmatrix}$$

∗ *circular* - Fills indents on circle along each axes

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 5 & 6 & 4 & 5 & 6 & 4 & 5 \\ 2 & 3 & 1 & 2 & 3 & 1 & 2 \\ 5 & 6 & 4 & 5 & 6 & 4 & 5 \\ 2 & 3 & 1 & 2 & 3 & 1 & 2 \end{pmatrix}$$

– dilation * ($d_{1,}.., d_n$) - Size gaps between elements kernels

– groups - number groups, on which are breaking down incoming and weekend channels

– bias - if *True,* adds to the exit displacement being studied.

* — parameter can admit one number a, What will be interpreted as a vector of size n : (a,..., a)

- Conv1d - One-dimensional roll
- Conv2d - Two-dimensional roll

3. Linear - Fully connected linear layer: applies linear transformation to incoming data, $y = xA^T + b$. If on Exit is served, the tensor dimensions

($d_{1,}.., d_{n,} H_{in}$), On exit will come out tensor dimensions ($d_{1,}.., d_{n,} H_{out}$), in accordance with the matrix A having size $H_{in} \times H_{out.}$ Parameters :

– in_features (H_{in}) - size incoming data

– out_features (H_{out}) - size weekend data

– bias — if *True,* adds to the exit displacement that is being studied.

4. Unifying Pooling layers are used to form an incoming tensor n-measurable association by a specific rule. The incoming tensor may have dimensions n + 1 or n + 2. Passing window size ((1), C,$k_{1,} k_{2,}.., k_n$), also invited core on to everything incoming *tensor* along with indents size ((0), 0, $p_{1,} p_{2,}.., p_n$) with a step ((1),,,,,, applying transformation to values windows, in case transformation — operation taking maximum by meaning windows (max pooling). Between kernel elements can be added intervals that depend on axes ((0), 0, $d_{1,} d_{2,}.., d_n$)

If at the entrance given a tensor of size ((N), C, D *1*, D *2*,.., D_n), the output tensor will have size $((N), C, D'_1, D'_2, \ldots, D'_n)$, where D'_i is calculated by the formula:

$$D'_i = \left\lfloor \frac{D_i + 2 \cdot p_i - d_i \cdot (k_i - 1) - 1}{s_i} + 1 \right\rfloor \quad (2)$$

Parameters:

– kernel_size * ($k_{1,..}, k_n$) - size kernels

– stride * ($s_{1,..}, s_n$) - kernel step for each axis

– padding * ($p_{1,..}, p_n$) - Dimensions indents for each axis

– dilation * ($d_{1,..}, d_n$) - Size gaps between core elements

– ceil_mode - if *True,* for calculation size, the output tensor will be rounded up

* - parameter can admit one chilo (a), which will be interpreted as a vector of size n : (a,...,a)

• MaxPool2d - Two-dimensional maximum pooling

5. • Dropout (Srivastava, et al., 2014) - Special layer that, under time teaching, resets random elements in the incoming tensor with probability *p*.

6. Constants - Layers where nothing is accepted at the entrance, that return a fixed value.

7. • ToType - A layer that casts a tensor to a given data type.

8. Arithmetic operations - such layers receive two layers as input, apply an arithmetic operation to them. Support broadcasting (Link to documentation). PyTorch, e.g., if possible, leads tensors up to a standard dimension if they differ.

9. Change tensor shapes • Flatten - Smoothes the adjacent tensor axes by one. In other words, if the input is a tensor of size ($a_{1,..}, a_{n,} b_{1,..}, b_{k,} c_{1,..}, c_m$), at the output we get a tensor of size ($a_{1,..}, a_{n,} b_1 \times b_2 \times \cdots \times b_{k,} c_{1,..}$).

Each layer corresponds to a class in your Torch model. nn module, let's call it the working module. All necessary functions and properties layers are implemented in the base class Layer, so when creating a new layer, you need only to register widgets that accept parameters, and if the working module is not implemented in the module n.n. Libraries like torch require creating a class that inherits from nn—module in which the forward method will be implemented.

Also, when creating a layer, you can adjust the number of input and output buttons by passing a list of their names, initialization methods, and the imitate method (Layer). During

training, each layer takes as input and returns dictionaries, with keys corresponding to the response button names. Since the majority of workers' modules take as input and return one tensor at a time, by default, each layer accepts a dictionary of the form { ′ *in* ′ : *input* }, transmits the value *input* into the working module, obtaining the tensor at the output, and then returns a dictionary { ′ *out* ′ : *output* }.

"Settings" tab training

In this tab, you can select and configure function losses, as well as the optimization algorithm.

The basic screen consists of two lines: the first corresponds to the function losses, and the second to the optimization algorithm. In the future, additional lines can be added that correspond, for example, to the planner speed teaching (learning rate scheduler).

There are two buttons in each row; the first one displays the name of the current function, loss, or optimizer. If you click on it, on the right, the options menu will open. Clicking the second button labeled Change opens a selection menu for functions, losses, or the optimizer, respectively.

Function losses are function that accepts entry foresight input models, data, together with the target values, and return a number or vector that characterizes quality predictions. Currently added features lose work on the following principle:

They have exactly one parameter – reduction - which may take the values none, mean, or sum.

Let x = (x 1,..., x N) - prediction models, y = (y 1,..., y N), where N - size batcha. Then, if value reduction - none :

$$l(x, y) = L = \{ l_1, \ldots, l_N \}^{\mathrm{T}}, \text{ where } l_n = f(x_n, yes_n) \quad (3)$$

Here *f* – function losses for scalar values.

If the reduction has a value, then

$$l(x, y) = \frac{\sum_{l_i \in L} l_i}{N} \quad (4)$$

And if the value is equal to the *sum*, then

$$l(x, y) = {}^{\mathrm{X}} l_{and}$$

$$l_{and} \in L$$

Currently added such functions losses :

- MSELoss - $f(x, y) = (x - y)^2$
- BCELoss - $f(x, y) = y \cdot \log x + (1 - y) \cdot \log(1 - x)$.

"Learning" tab

After that, once the dataloader was initialized, selected transformations for reworking, created, compiled, and tested the model, and chose the function, losses, and the optimization algorithm, you can launch the training. To do this, in the "Training" tab, press the Start button. After this, training for Algorithm 1 will begin.

At the bottom of the tab is an indicator of progress that shows the current era and the percentage processed data on it (Fig. 2). In the middle, a schedule shows values, functions, errors in training and validation samples, and above a menu that allows interaction with the graph: change the scale, change the colors of the lines, and more.

Discussion

Even though users can already create and configure neural networks with the help of Learn 2 Learn, the program is still in its early stages of development: many functions regularly used by machine learning engineers were not added, which makes the StudySupport system challenging to use in practice. Among the tasks that will be added in the future, you can list:

- Possibility of device settings on which the model is trained, and in the future, the possibility of connecting to remote servers and their computational capacities
- Adding opportunities creation layers and functions, losses by analogy with the current method of creating a data loader
- Possibility creation layers and functions, loss graphical method - combining already existing layers into blocks and saving them on the device
- Adding libraries built-in standard datasets such as MNIST, CIFAR, Coco, etc., and dataloaders built for them
- Possibility of adding additional metrics on graphs during training
- Parsing existing models created in the traditional way
- Adding libraries' previous models, such as ResNet, DenseNet, etc.
- Improvement design.

Conclusion

This study introduced the StudySupport, a visual system for configuring deep learning models that reduces reliance on programming and enables broader adoption of AI technologies. The system supports IT management processes by streamlining data handling, model configuration, and performance evaluation. Future improvements will focus on expanding integration with enterprise decision-support platforms, adding libraries of pre-trained models and datasets, and enhancing interoperability with cloud-based resources. These developments will strengthen the system's role as a bridge between advanced AI research and its application in IT management, business decision-making, and digital transformation initiatives.

Thus, a GUI application was created that allows you to create and configure arbitrary deep learning models from scratch with virtually no code. The only element configured traditionally was designed to demonstrate different approaches to visual configuration and the possibility of switching to a GUI without losing the flexibility of the command-line interface.

The developed program allows the user to control the entire learning process: data loading, neural network construction, selection of optimization methods, and quality analysis. The system is built so it is easy to supplement and improve. The main achievement of the work is the development of a base layer class, which allows adding an arbitrary data transformation layer to the program with ease.

Although the graphical interface's capabilities have not been fully implemented, the system already automates a large part of the user's work by default, displaying additional information about objects on the screen that would otherwise require extra effort to obtain.

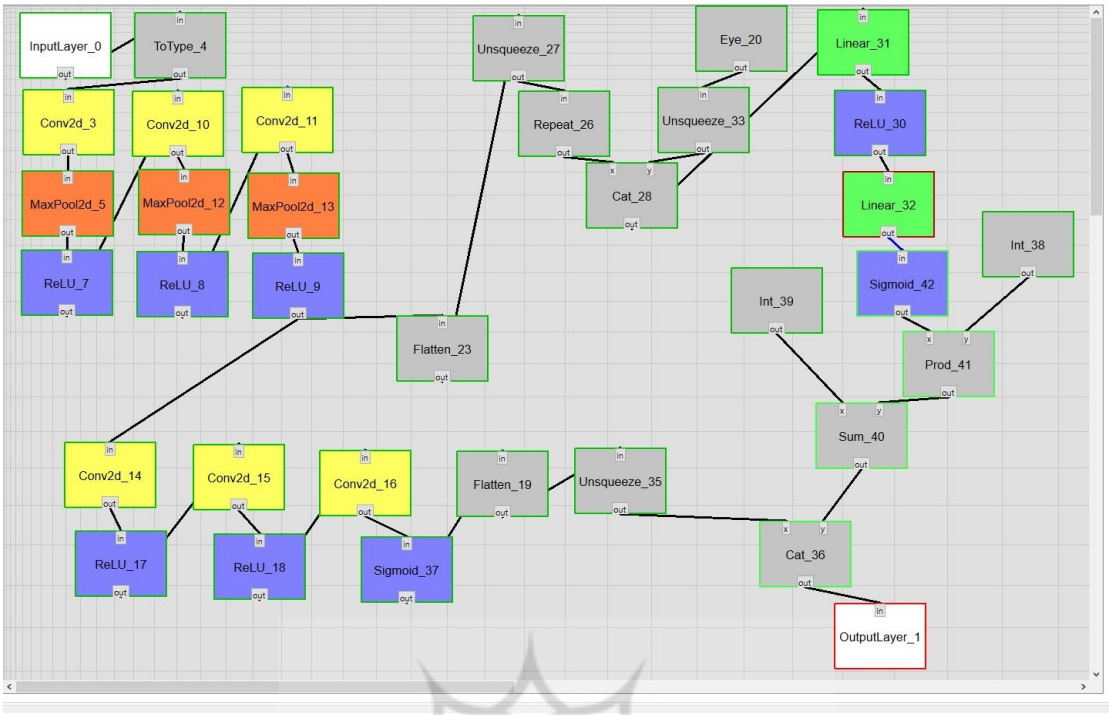

Figure 1. Designer neural networks

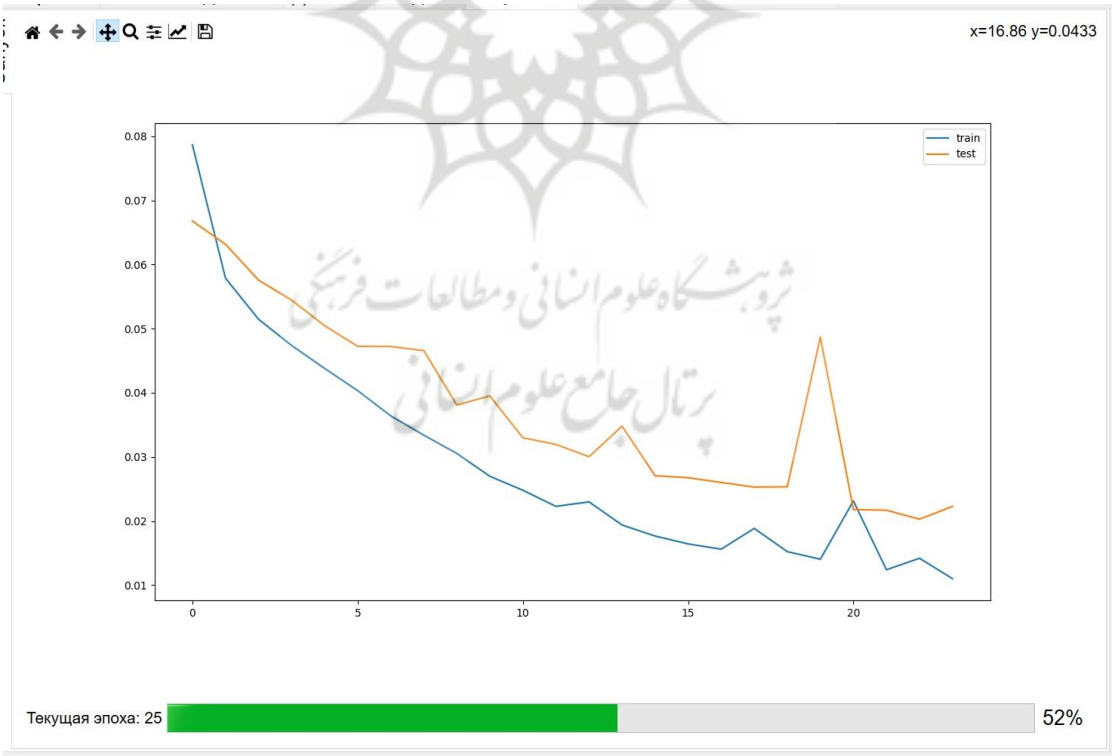

Figure 2. Progress training and schedule functions mistakes

Conflict of interest

The authors declare that there is no conflict of interest regarding the publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

References

Amershi, S., Weld, D. S., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S. T., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). *Guidelines for human–AI interaction*. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3290605.3300233

Andrienko, N., Andrienko, G., Adilova, L., & Wrobel, S. (2022). *Visual analytics for human-centered machine learning*. *IEEE Computer Graphics and Applications, 42*, 123–133. https://doi.org/10.1109/MCG.2021.3130314

Babenko, V., Chebanova, N., Ryzhikova, N., Rudenko, S., Birchenko, N. (2018). Research into the process of multi-level management of enterprise production activities with taking risks into consideration. *Eastern-European Journal of Enterprise Technologies,* 1, 3 (91), 4-12. http://dx.doi.org/10.15587/1729-4061.2018.123461

Babenko, V. (2020). Enterprise Innovation Management in Industry 4.0: Modeling Aspects. Emerging Extended Reality Technologies for Industry 4.0: Early Experiences with Conception, Design, Implementation, Evaluation and Deployment, pp. 141–163. https://doi.org/10.1002/9781119654674.ch9

Babenko V. A. (2013). Formation of economic-mathematical model for process dynamics of innovative technologies management at agroindustrial enterprises. *Actual Problems of Economics.* 139 (1), 182-186. Retrieved from: https://www.scopus.com/record/display.uri?eid=2-s2.0-84929991982&origin=inward&txGid=c69f0746cede0da5f287471cd688 08af

Babenko, V., Pravotorova, O., Yefremova, N., Popova, S., Kazanchuk, I., Honcharenko, V. (2020). The Innovation Development in China in the Context of Globalization. WSEAS Transactions on Business and Economics, Vol. 17, 2020, Art. #25, pp. 523-531. https://doi.org/10.37394/23207.2020.17.51

Baldi, P., & Sadowski, P. J. (2013). *Understanding dropout*. In *Advances in Neural Information Processing Systems* (Vol. 26). Retrieved from: https://papers.nips.cc/paper_files/paper/2013/hash/14acdf64020072a74d1cf17d7e6f5576-Abstract.html

Corbató, F. J., & Vyssotsky, V. A. (1965). Introduction and overview of the Multics system. In *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I* (pp. 185–196). Association for Computing Machinery. https://doi.org/10.1145/1463891.1463912

Donnat, C., & Holmes, S. (2018). *Tracking network dynamics: A survey using graph distances*. *The Annals of Applied Statistics, 12*(2), 971–1012. https://doi.org/10.1214/18-AOAS1176

Engel, F. R. K. (1988). *Magnetic tape - from the early days to the present*. *Journal of the Audio Engineering Society, 36*(7/8), 606–616. Retrieved from: https://www.aes.org/journal/

Gontareva, N., Babenko, V., Shmatko, N., Litvinov, O., Obruch, H. (2020). The Model of Network Consulting Communication at the Early Stages of Entrepreneurship. *WSEAS Transactions on Environment and Development,* 16(39), 390-396. https://doi.org/10.37394/232015.2020.16.39

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv:1207.0580. https://arxiv.org/abs/1207.0580

Hrabovskyi, Y., Babenko, V., Al'boschiy, O., Gerasimenko, V. (2020). Development of a Technology for Automation of Work with Sources of Information on the Internet. WSEAS Transactions on Business and Economics, 17 (25), 231-240. https://doi.org/10.37394/23207.2020.17.25

Huang, Y. L., et al. (2022). *ExBrainable: An open-source GUI for CNN-based EEG decoding and model interpretation*. arXiv:2201.04065. https://arxiv.org/abs/2201.04065

Jin, H., Wagner, M. W., Ertl-Wagner, B., & Khalvati, F. (2022). An educational graphical user interface to construct convolutional neural networks for teaching artificial intelligence in radiology. *Canadian Association of Radiologists Journal*. Advance online publication. https://doi.org/10.1177/08465371221144264

Kashchena N., Chmil H., Nesterenko I., Lutsenko O., Kovalevska N. (2024). Diagnostics as a Tool for Managing Behavior and Economic Activity of Retailers in the Conditions of Digital Business Transformation. *Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies.* 194, 149–173. Springer, Cham. https://doi.org/10.1007/978-3-031-53984-8_7

Klemm, S., et al. (2018). *Barista: A graphical tool for designing and training deep neural networks*. arXiv:1802.04626. https://arxiv.org/abs/1802.04626

Kyrylieva L., Polyvana L., Kashchena N., Naumova T., Akimova N. (2023). Organizational aspects of forming an information and analytical service for the management of trade enterprises in the period of digitalization. Financial and Credit Activity Problems of Theory and Practice. Vol. 3 No. 50, 127–138. [in Ukrainian] https://doi.org/10.55643/fcaptp.3.50.2023.3996

Kuznetsov, A., Kavun, S., Smirnov, O., Babenko, V., Nakisko, O., Kuznetsova, K. (2019). Malware Correlation Monitoring in Computer Networks of Promising Smart Grids. 2019 IEEE 6th International Conference on Energy Smart Systems, ESS 2019 - Proceedings, 8764228, 347-352. https://doi.org/10.1109/ESS.2019.8764228

Maynard, M. M. (2003). Univac I. In *Encyclopedia of Computer Science* (pp. 1813–1814). John Wiley & Sons.

Nesterenko I., Kashchena N., Chmil H., Chumak O., Shtyk Yu., Nesterenko O., Kovalevska N. (2024). Devising a methodological approach to identifying the economic potential of production costs for eco-innovative products. *Eastern-European Journal of Enterprise Technologies,* 3, 13 (129), 6–15. https://doi.org/10.15587/1729-4061.2024.304805

O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks*. arXiv:1511.08458. https://arxiv.org/abs/1511.08458

Pietrołaj, M. & Blok, M. (2024). *Resource constrained neural network training*. *Scientific Reports, 14*, Article 2421. https://doi.org/10.1038/s41598-024-52356-1

Savytska, N., Babenko, V., Chmil, H., Priadko, O., & Bubenets, I. (2023). Digitalization of Business Development Marketing Tools in the B2C Market. *Journal of Information Technology Management,* 15(1), 124-134. https://doi.org/ 10.22059/jitm.2023.90740

Savytska, N.; Zhehus O.; Polevych K.; Prydko O. & Bubenets I. (2024). Enterprise Resilience Behavioral Management in a Decision Support System. *Journal of Information Technology Management,* 16 (4), 100-121. https://doi.org/10.22059/jitm.2024.99053

Shorikov, A.F., Babenko, V.A. (2014). Optimization of assured result in dynamical model of management of innovation process in the enterprise of agricultural production complex. Economy of Region, Issue 1, pp. 196-202. http://dx.doi.org/10.17059/2014-1-18

Shtal, T., Proskurnina, N., Savytska, N., Mykhailova, M., Bubenets, I. (2023). Analysis of the Vectors of Digital Transformation of Retail Trade in Ukraine: Determination Methodology and Trends. *Economic Affairs,* 68(Special Issue), 939-945. https://doi.org/10.46852/0424-2513.2s.2023.42

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A simple way to prevent neural networks from overfitting*. Journal of Machine Learning Research, 15, 1929–1958. https://doi.org/10.5555/2627435.2670313

Zhang, S., Lu, J., & Zhao, H. (2024). *Deep network approximation: Beyond ReLU to diverse activation functions*. *Journal of Machine Learning Research, 25*(1), 1–39. Retrieved from: https://jmlr.org/papers/volume25/23-0912/23-0912.pdf

Wang, A. (2025). *A Review of Convolutional Neural Networks: Evolution, Applications, and Future Directions*. *Applied and Computational Engineering, 166*. https://doi.org/10.17605/OSF.IO/XXXXXX

luster-based web servers. *Indones. J. Electr. Eng. Comput. Sci*, *19*(1), 510-517.

Bibliographic information of this paper for citing:

Babenko, Vitalina; Brazhnykov, Andrii; Gavkalova, Nataliia; Rudenko, Serhii; Oliskevych, Marianna; Fridman, Olena; Babenko, Dariia (2025). Visual System for Configuring Machine Learning Models to Support IT Management and Decision-Making. *Journal of Information Technology Management*, 17 (4), 117-135. https://doi.org/10.22059/jitm.2025.105486