CEUR-WS.org/Vol-3171/paper94.pdf

Specifics of Designing and Construction of the System for Deep Neural Networks Generation

Oleksandr Mediakov and Taras Basyuk

Lviv Polytechnic National University, Bandera str.12, Lviv, 79013, Ukraine

Abstract
In the article we propose designing and implementing an information system for code-free generating of Deep Neural Networks (DNN) based on TensorFlow and Flutter. Because of the mathematical properties and production success of DNN, it`s clear that many medium and small companies or individual users must be able to use the benefits of machine learning models. In order to simplify and speed up the process of generating one – we have created the object-oriented design of the respective informational system using UML. The analysis of the basic requirements to the system with the description of the text scenario according to the RUP methodology was carried out, which allowed forming the leading use case of our system. For the object-oriented designing of our system, we have used the IBM Rational Rose CASE tool, which let the decomposition of complex objects and contributed to writing the software solution. Our software was created using Flutter as development technology and TensorFlow as DNN making and training backend technology. It can be seen as proof of design relevance, its validation, and hence the possibility of its practical application in code-free generating DNN. Further research will design related software packages to expand the functionality and test their work.

Keywords [1]
Deep Neural Networks (DNN), object-oriented design, TensorFlow, models, machine learning.

1. Introduction

Machine learning models have been used successfully in research, business, corporate and government management, device controlling, smart homes, and more. One of the most popular algorithms is Deep Neural Networks. The use of DNNs allows one to solve a robust set of problems: regression and classification [1], clusterization and anomalies detection [2], dimensionality reduction [3], generation [4, 5], etc.

The creation, training, and deployment of a neural network require a particular intellectual resource, i.e., an analyst who has specific knowledge of programming, mathematics, and other related sciences. The need to have most of these skills significantly delays the use of DNN in small businesses or by individuals.

However, in the context of globalization, Industry 4.0, and other factors, the use of DNN should become more accessible to average users and small companies. The growing need for such access is evidenced by the development of appropriate information and intelligent systems that will create and implement models of DNNs. But, such systems have not gained popularity yet. Nevertheless, researchers increasingly prove the high efficiency and superiority over other algorithms and machine learning methods [6, 7].

That is why there is a need to study the necessary functionality for a system that will create neural networks for the user, create a project that developers can use to develop software, and hence the relevance of design and development of DNN generation system.

COLINS-2022: 6th International Conference on Computational Linguistics and Intelligent Systems, May 12–13, 2022, Gliwice, Poland
EMAIL: Oleksandr.Mediakov.sa.2019@lpnu.ua (O. Mediakov); Taras.M.Basyuk@lpnu.ua (T. Basyuk)
ORCID: 0000-0002-2580-3155 (O. Mediakov); 0000-0003-0813-0785 (T. Basyuk)
© 2022 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

2. Analysis of the subject area

2.1. Definitions and concepts for the DNN generation

2.1.1. Main definitions and components of DNN

Creating the Deep Neural Network (DNN) is an intellectual and informational task that involves a clear understanding of terminology, study of the subject area, and the correct use of existing mechanisms and technologies. Therefore, the paper's primary task is to form a unified set of definitions that will determine the rules and particular constraints in generating the neural network.

The abstract highest concept is the Artificial Neural Network (ANN). ANN is a brain simulation system for the perception and analysis of information. In the article, an Artificial feed-forward Neural Network called a system which takes an input vector X and builds a nonlinear function f(X) to predict the response Y [8]. The dimensionality of X and Y, and DNNs shapes and sizes accordingly, depend on the application [9]. As we mentioned earlier, the classic issues of ANN: regression and classification, clusterization and anomalies detection, dimensionality reduction, generation, etc.

This study focuses on creating networks that can solve the first two problems. Its known that DNN consists of at least three fully connected layers: input, hidden, and output. A sequence of layers and their properties determine network architecture. The more hidden layers there are, the 'deeper' the network is. Classic DNN uses dense-connected layers also called Dense. The neural network topology is a graphical representation of DNN architecture in the language of graph theory. Using the introduced concept of topology, it is possible to create a definition of DNN, which we will use throughout the paper. Deep Neural Network is a multilayer Artificial feed-forward Neural Network in which the topology of two sequential layers forms a complete bipartite graph.

Those networks use backpropagation as a training method [1, 10]. The defined classes of tasks, definitions and learning algorithm bound the domains for the main parameters required to create a DNN and constrain some architectural solutions. For example, AutoEncoders (AE) can meet DNN criteria. But the main task of AE is dimensionality reduction [3], which is not included in the tasks list. It is related to the necessity of combining two separate models (encoder and decoder), and this possibility is not yet provided by the design of the developed system. Typically, each network has three related processes: learning or fitting, testing or validation, and inference or prediction. Each of the processes requires input from the subject area. Such input data must have a unified form or a pipeline of obtaining it (cleaning, normalization, feature engineering, etc.) [11]. However, the first two processes require some other related values that are part of the standard parameters for DNN.

The standard parameters of the DNN are the number of layers (or its 'depth'), number of units of each layer, activation functions of each layer, loss function, stochastic optimization algorithm, number of repetitions of the learning process, the batch size for input data splitting. Optional features are a set of DNN performance metrics, callbacks of the learning process (functions that are called after one step of the train process), the initialization kernel (method of random initialization of weights and biases) [12], etc. Incidentally, the number of layers has a limit of three positions, and the upper limit does not exist. But the study of an infinite number of layers and continuous models is another domain. The set of valid layers' activation and loss functions is determined by the tasks that DNN solves.

In order to build fairly complex nonlinear relationships in the regression process it will be sufficient to use several types of the most common functions: *sigmoid* (transform data to the interval [0,1]), *hyperbolic tangent* (transform data to the interval [-1,1]), *relu*, and their variations: *leaky relu* or *elu*, for some regression tasks *swish* function is the best choice [13, 14], and last but not least - the usual *linear* function. The loss function can be a classic *mean square or absolute error*.

The classification problem requires more complex combinations of functions, particularly the widespread use of *softmax* (or *numerically stable softmax* also called the *normalized exponential* [1]) as an activation function for the output layer, combined with *cross-entropy* as a loss function [1]. When considering the classification task, it is necessary to build an agreement with the input data and network parameters, which minimizes the cost of building and debugging the model. When considering the classification task, it is necessary to agree the input data (esp. 'ground truth output') with network parameters, which minimizes the cost of building and debugging the model, and for example, representing a target class. Suppose the data is not decoded and saved as the original labels or class numbers. In that case, any of the mentioned loss functions will not make sense because the cross-entropy works with One-Hot encoded data [11]. Therefore, it is necessary to use sparse functions, such as *sparse cross-entropy*, in the absence of encoding.

The optimization algorithm is a stochastic method that defines the logic of applying gradients to the weights and biases of the DNN [15]. The most common and popular general-purpose method is Adam [10], that we are going to use in the default settings. But users could use many other algorithms, and an information system will allow them to change the method.

The last parameter to consider is initialization kernel, the rule by which the initial random values of weights and biases are selected.

2.1.2. Preventing overfitting and stabilizing of the training process

There are several problems and difficulties when dealing with neural networks at every implementation stage. The common downside is the requirement of a massive amount of prepared data, which occurs at the beginning of the process. Two central problems of the learning process are overfitting and underfitting. And one of the main drawbacks of the ready-to-use DNN is a lack of meaningful interpretation of its learned parameters.

But, when building a system that can generate DNN, we need to consider those problems that appear at this step: over- and underfitting. That is why IS should offer users some prevention mechanisms.

An advanced, effective way to stabilize learning is the intermediate normalization of data batches. When using Keras API, this mechanism of normalizing is implemented using the BatchNormalization layer [16]. The main goal of normalization is to transform data into a normal distribution $N(0, 1)$ [12,16]. During the training process, normalization is performed with the standard deviation and mean of batches, and during the inference, the layer uses the rolling average of the mean and standard deviation of the batches.

The problem of overfitting also has classic approaches of handling it. For instance, one could make learning a little bit harder or with some difficulties, which will help to avoid forming erroneous patterns in the network. Dropout is one the most common technique of preventing overfitting with mentioned approach [1]. TensorFlow Dropout layer randomly drops out some fraction of a layer's input units every training step, making the fitting process a little more challenging. As a network layer, it affects the next dense layer and is only called during the training. The layer's behavior is determined by a single parameter - the percentage of dropped connections in one step (aka the rate).

Often, a combination of three layers: Dropout, BatchNormalization, Dense – a classic "composite" layer, is used as the hidden and output layers of the network, see Figure 1. Normalization can be used as the first layer of the DNN if input data is not normalized.

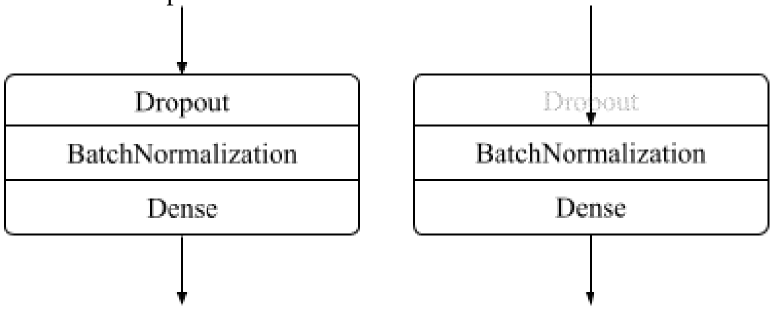


Figure 1: "Composite" hidden layer of DNN (for training and inference respectively)

Such layers correspond to the concept of the system and the definition of DNN described earlier. Of course, building such layers is a good solution for large datasets or complex tasks. Still, it may not be appropriate to use more complex structures with minor challenging tasks, such as multivariate regression, so the system should have the opportunity to remove these additional layers.

For example, the DNN architecture for the classification task, typically provided by the system, is represented in Figure 2.

One possible way to improve a template is to use intelligent automated neural architecture search methods, which can be used to generate the most proper architecture for the given dataset [17].

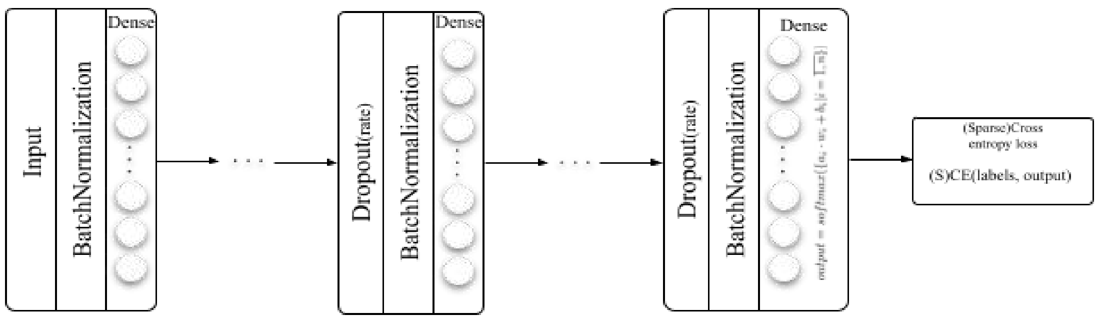


Figure 2: Complete DNN template

2.2. Analysis of available solutions

As shown in [18], DNN entail an inherent advantage over traditional machine learning. They can handle data in its raw form without the need for manual feature engineering. That is why they are constantly used in various business models, which gives rise to their popularity. The popularity and power of ANN have turned into a large number of information technologies of different levels of complexity that allow one to create, teach and deploy DNN for production.

We allocate three types of systems to generate DNN:

- Frameworks, libraries, and extensions of programming languages for machine learning
- Analytical platforms with support for the creation of multilayer perceptron's or other types of DNN
- Specialized systems for DNN creating

The first type requires the developer to directly write program code, quality mathematical training, and programming skills. Examples of such systems are well-known libraries such as TensorFlow (Google), PyTorch (Meta, Nvidia, etc.) [19], Theano [20], OpenNN, Caffe, and many others. Those systems run on several programming languages, like Python, R, C ++, Java, Julia, etc. Systems of this type are a powerful tool for knowledgeable users, and these systems have the highest flexibility and support from communities and big IT companies, can run on GPU [7]. However, for users with limited programming knowledge, expertise of a specific language or framework will not allow you to create the necessary artificial network quickly, respectively, a possible loss of time or money from customers.

The second and third types are usually code-free systems. The second class includes systems such as NeuroDesigner or Neuroph, which allow you to use the graphical interface to go through the necessary steps to create a network. They use their add-ons and libraries to get an advantage in the learning speed of large DNNs. However, this approach imposes limitations on the rate of updating and efficiency of systems and directly machine learning libraries.

Our system is designed and developed as the third type. This means that it must define for the user the minimum required set of steps and input to achieve the desired result, i.e., the creation of DNN. Due to the possibility of quick updates, additions, and simple integration, it was decided to delegate networks' direct building and training processes to the TensorFlow library with direct using of Keras API [21, 22].

2.3. System specifications and design

2.3.1. System conception and main use case

Conceptually, the designed system generates correct code scripts for DNN creating using TensorFlow, see Figure 3. Two of the main processes of the system are delegated to other information systems. The code execution process is delegated to programming language interpreters, while learning and saving processes are charged to the TensorFlow library. With this approach, we can increase the number of data formats, that out system uses and produces, as TensorFlow works with many programming languages. Therefore, the generated code can be saved as files of these languages or integrated into projects with one of that languages supports.

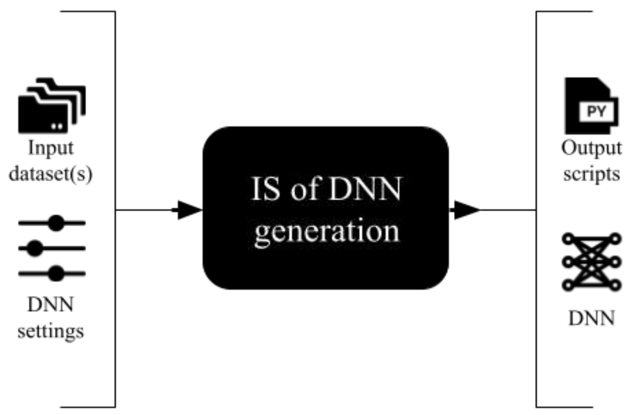


Figure 3: Black box interpretation of IS conception

Since creating a system is a complex job, we have subjected it to a particular logical decomposition. In the aftermath, we can cluster systems functions within decomposed subtasks. The main subtasks of the designed system are:

- Configuration of main DNN parameters
- Downloading of input files
- Generation of code and its execution configuration
- Saving and uploading of projects (locally and from database)
- Visualization of processes and their results
- Implement login and system context supporting

We propose a text description of the system scenario according to the RUP methodology to set out system functionality.

1. Stakeholders and their requirements:
- User – requires simplicity of the system
- TensorFlow – requires correctly generated program code

2. The user of the software system:
- User – a person who needs using the system

3. Prerequisites for use case:
- User with Google account (or email associated with Google account)
- For local code execution: installed programming language (e.g., R, Python) and TensorFlow library
- Stable Internet connection
- Correct, ready-to-use dataset file

4. Successful scenario:
- The user logs in (or registers) with Google account
- The user creates a new project, adds a name and a subscription (if needed)
- The user uploads the dataset file into the system
- The user chooses the target field optionally edits newly configured input and output layers
- The user changes train/test split percentage
- The user adds and modifies any number of hidden layers, modifies activation functions, units count, removes non-trainable layers if needed
- The user chooses the Loss function
- The user modifies metrics
- The user sets the number of epochs
- The user initiates the learning and testing process for a current DNN model
- TensorFlow executes scripts, while IS streams stdout results for the user
- The user checks model performance and metrics results
- The user starts the process of project saving into the database, choosing a saving configuration
- The database actor saves files and document and returns confirmation of successful results
- The user logs out

5. Scenario Extensions or Alternative threads:
- Invalid input shape:
    - TensorFlow returns an error
    - The user analyses given error
    - The user changes input shape for the model
    - The user initiates the learning process (extension's return point)
- Model performance unsatisfied the user
    - The user changes the layers count if needed
    - The user modifies activation functions
    - The user chooses a different optimization method
    - The user initiates the learning process (extension's return point)
- Alternative thread
    - The user checks model performance and metrics results (thread's start point)
    - The user modifies model parameters
    - The user initiates the learning and testing process for a current DNN model
    - TensorFlow restarts processes and saves the current model
    - TensorFlow returns results via streamed stdout
    - The user checks model performance and metrics results (thread's return point)
- Alternative thread
    - The user checks model performance and metrics results (thread's start point)
    - The user starts the process of project saving project on the local device with all scripts
    - The user logs out (thread's return point)
- Alternative thread
    - The user initiates the learning and testing process for a current DNN model
    - The IS sends scripts to the server
    - Server's TensorFlow executes scripts, while IS streams results for the user
    - The user checks model performance and metrics results (thread's return point)

6. Post-conditions:
- Clean up temporary files of the model and close threads (processes).
- Saved files and projects
- Generated DNN and scripts

7. Special system conditions:
- Ensuring the correctness of the generated code files
- Nice UI/UX design of the system
- Support of many popular file extensions for datasets or model
- For local execution: programming language, TensorFlow, minimum required architectural requirements for the device needed for the language or library

It is important to point out that current trends in software development require cross-platform support of the system on various devices, including desktop, mobile, and web. Therefore, code execution will be performed using cloud services to maintain multi-platforming, scalability, and minimally dependence on user's devices. The ability to run program code on user's devices can only be done for advanced users, for instance, with devices based on TPU or a specific language version preference [7]. These aspects affect the choice of system development tools.

2.3.2. Object-oriented design

Object-oriented design with UML is the study's next step. It's based on the previously given specifications. Implementing this step is necessary for the object-oriented decomposition of complex objects and then writing an effective program using the benefits of object-oriented design and programming paradigm. First thing first, we designed a UseCase diagram, which is displayed (Figure 4). The development of diagram allows us to describe the functional purpose of the system, i.e., to show the initial conceptual model of the system in order to further detail it in the form of logical and physical models.

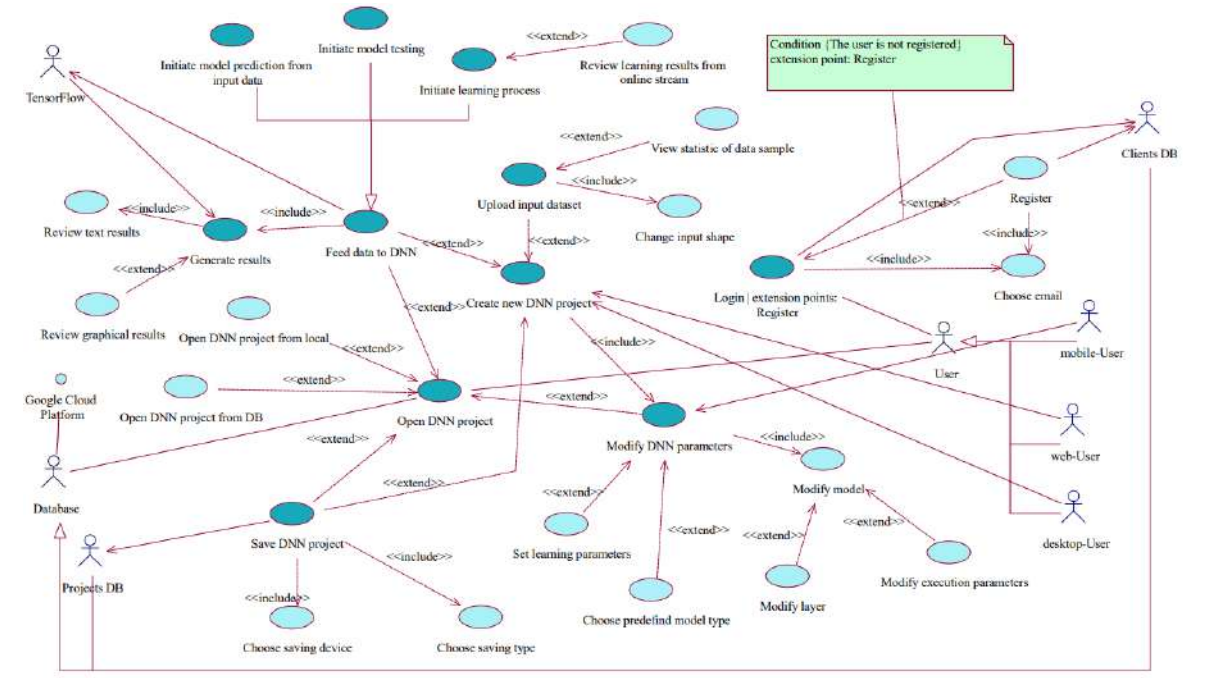


Figure 4: UseCase diagram

Determining the system's basic functionality and interactivity with external actors allows choosing the minimum set of objects whose interaction and cooperation can fully perform use cases or use cases. The description of such objects is made on the class diagram [22] – Figure 5.

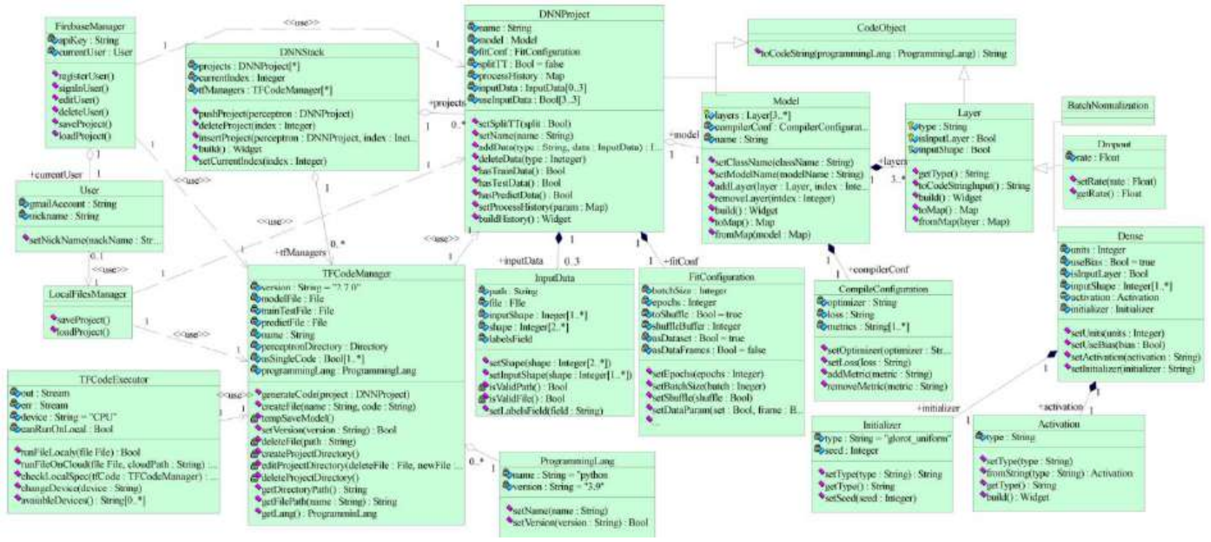


Figure 5: Class diagram of information system

Methods and attributes have minimal binding to the specific programming language. Table 1 shows some sequences of class methods that allow one to implement specific use cases of the system.

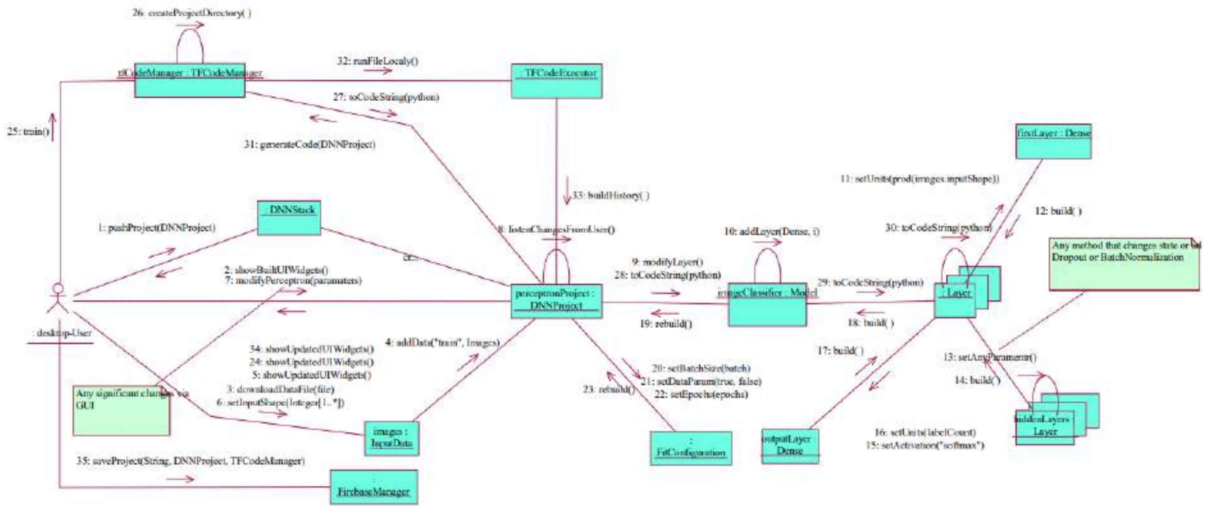
The achievement of the main goal of the information system - the generation of DNN, can be seen as a logical sequence that represents the cooperation of class instances.

A cooperation diagram was constructed to demonstrate that sequence as a specific action of the instances (Figure 6). We specify a brief description of the classical scenario of the use case for building a DNN project from a cooperation diagram.

Table 1
Correspondence of use cases and sequence of classes' methods

UseCase	Classes' methods
Log in/Sign in	FirebaseManager().signIn()
Create new DNN project	DNNProject().setName(), DNNStack().push()
Upload input dataset	DNNProject().addTrainData() OR/AND DNNProject().addTestData() OR/AND DNNProject().addPredictData()
Initiate learning process	TFCodeManager().createProjectDirectory(), TFCodeManager.createFile(train_test), TFCodeManager.generateCode(), DNNProject.toCodeString(), TFCodeExecutor().run()
Save DNN project	FirebaseManager().saveProject(), DNNProject().toMap(), TFCodeManager().getDirectoryPath()

The successful DNN generating scenario assumes that the user has all the necessary programming language modules (for local execution), an Internet connection, installed information system. To generate the network, the user uploads data, sets the model's parameters, optimization method, and configuration of the learning process. After starting the training, the system creates files, initializes execution, streams the results to the user, and after saves the project in the database.


26: createProjectDirectory( )
tfCodeManager : TFCodeManager
32: runFileLocaly()
: TFCodeExecutor
27: toCodeString(python)
25: train()
31: generateCode(DNNProject)
33: buildHistory( )
: DNNStack
1: pushProject(DNNProject)
8: listenChangesFromUser()
2: showBuiltUIWidgets()
7: modifyPerceptron(paramaters)
perceptronProject : DNNProject
: desktop-User
34: showUpdatedUIWidgets()
24: showUpdatedUIWidgets()
5: showUpdatedUIWidgets()
3: downloadDataFile(file)
6: setInputShape(Integer[1..*])
4: addData("train", Images)
Any significant changes via GUI
images : InputData
35: saveProject(String, DNNProject, TFCodeManager)
: FirebaseManager
10: addLayer(Dense, i)
9: modifyLayer()
28: toCodeString(python)
imageClassifier : Model
19: rebuild()
29: toCodeString(python)
18: build( )
: Layer
firstLayer : Dense
11: setUnits(prod(images.inputShape))
12: build( )
30: toCodeString(python)
Any method that changes state or Dropout or BatchNormalization
17: build( )
13: setAnyParametr()
14: build()
20: setBatchSize(batch)
21: setDataParam(true, false)
22: setEpochs(epochs)
23: rebuild()
: FitConfiguration
outputLayer : Dense
16: setUnits(labelCount)
15: setActivation("softmax")
hiddenLayers : Layer
Figure 6: Cooperation diagram

In order to construct a specific physical system, it is necessary to somehow implement all the elements of the logical representation (Figures 4-6) in one particular material entity. Another aspect of the model representation is intended to describe such an entity, namely the physical manifestation of the model. In the UML language, for the physical representation of the system model, so-called implementation diagrams are used, one of which is the component diagram [23,24], shown in Figure 7. The application of this diagram allowed to carry out visualization of the general structure of the software system's source code, showing the specifications of the implemented version of the software system and ensuring the modularity of its construction [24].

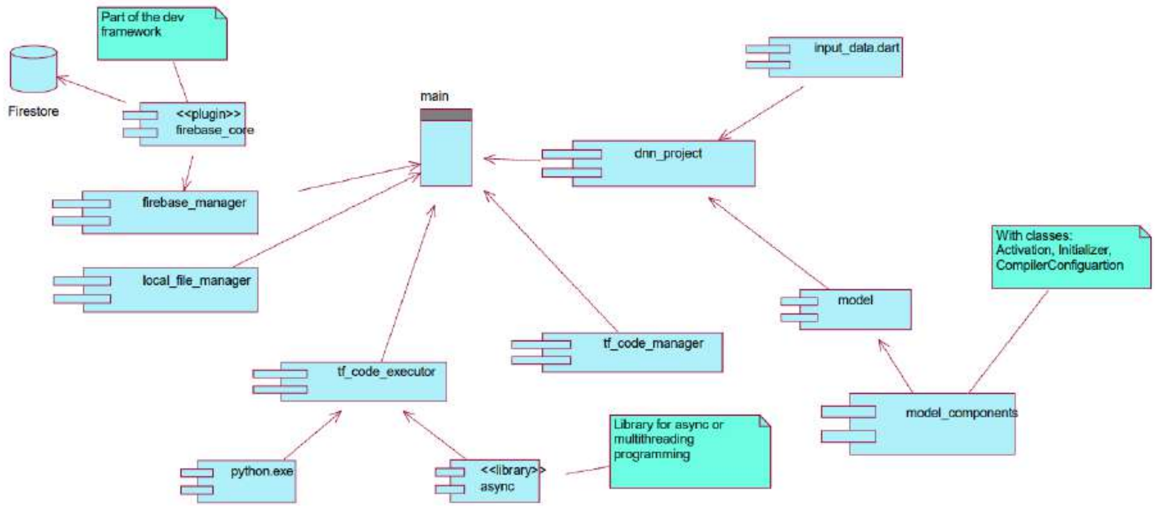


Figure 7: Component diagram

3. Means of solving the problem and practical implementation

3.1. Choice of development tools

The next stage of the study was the choice of developing tools that allow high-quality and rapid development of system software. The structural logic of the system's primary task can be described in two parts: data collection from the user and generation with the execution of code scripts. Executing scripts that build, teach and save DNN is delegated to the TensorFlow library. Many programming languages support the library, and the system design provides the option to choose one, but the default and recommended choice is Python.

Creating an ANN in TensorFlow is possible in different ways, but we have preferred the Functional API within developed software [20].

For the high-quality implementation of the designed system in the form of software with a graphical interface, the development technology must meet specific parameters.

- Built on imperative programming language, with support for the paradigm of object-oriented programming
- Speed of code execution and efficiency of its writing and debugging
- Cross-platform - creating a unified code base for the main platforms: mobile (Android, iOS), desktop (devices based on Windows, Linux, or macOS) and web applications

We are talking about a particular stack of technologies, or frameworks that can easily interact, avoiding the need to use many technologies at once. Many tools meet the described requirements, among which are the following.

- Python. The most important advantage of using Python is the nativeness of TensorFlow (within the system context), which will increase the speed of tasks and ease of integration. The main drawback is using Python for developing a mobile application. There are many high-quality libraries and frameworks for desktop and web (Tkinter, Django, etc.), while for Android and iOS, there is a need to use native development tools - Kotlin and Swift.
- C/C++. C ++ is a standard among object-oriented programming languages. With the help of assistive technologies, it is possible to implement the system on different platforms, and it is possible to use frameworks: CMake, Android NDK, C ++ iOS development tool. However, creating web applications and applications for the macOS operating system is problematic when using C ++, especially for visual components. Therefore, there is a need for further use of Swift for macOS and HTML / CSS / JavaScript - for web development.
- JavaScript with React Native. With the help of this programming language, you can describe the logic of the interaction of system objects to provide the necessary functionality. Also, there is TensorFlow support for JS [25]. With the React Native framework, it is possible to use a single code

base for all platforms, which significantly speeds up developing a multiplatform system with a quality UI.

● Java. The use of Java will have some advantages in terms of security and data integrity. However, there are some problems while using Java in implementing the GUI on the iOS platform, integration with TensorFlow, etc.

Based on the analysis, we have chosen the Flutter framework of the Dart programming language [26] as the software solution [27]. The main advantages of Flutter over the analyzed options are the use of a single source code for all platforms, speed of execution, creation, and launch of applications, its own kernel for rendering [28, 29].

The graphical component of the software is a set of widgets. Flutter contains ready-made graphic elements with a specific shape and design [27]. These widgets come in two different designs - Material and Cupertino [29]. We used self-designed and Material components while creating the software. Concerning database, in terms of ease of implementation and integration with the selected development technology, it is recommended to use a system such as BaaS – Firebase [30]. Firebase is a service from Google that includes a set of subsystems, including Firestore and Realtime Database. These two NoSQL databases allow you to store system information, user data and required project files [30].

3.2. Practical implementation

With provided analysis of the development tools, we have construct the NeuroPIS information system. The features of NeuroPIS are presented in the form of the classical use case of the system, namely: creating a project, uploading the input dataset file, configuring the model and learning parameters, initiating code execution, saving the model. We have chosen the flatten version of the classic DNN training dataset - MNIST, to set a use case [31]. Each data row contains a display of numbers from 0 to 9 ("label") and 784 values (as flattened by row 28 by 28 images) from 0 to 255, corresponding to the pixel's grayscale.

Figure 8: Part of the visualized MNIST dataset

To show the simplicity of creating DNN within an information system, we will use the default architecture for the use case example. The start screen of the system is shown in the Figure 9.

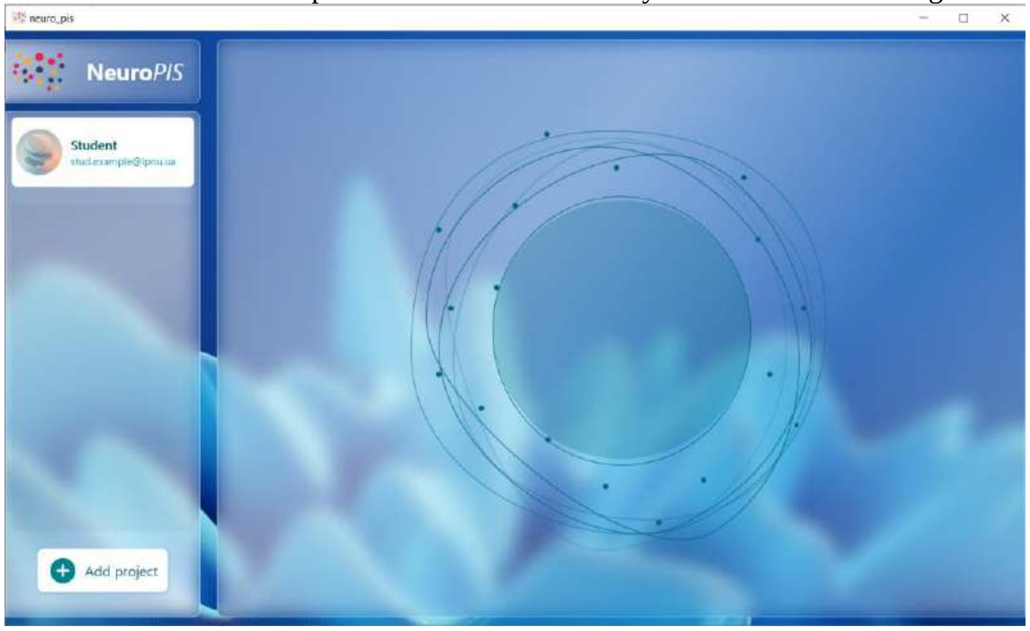
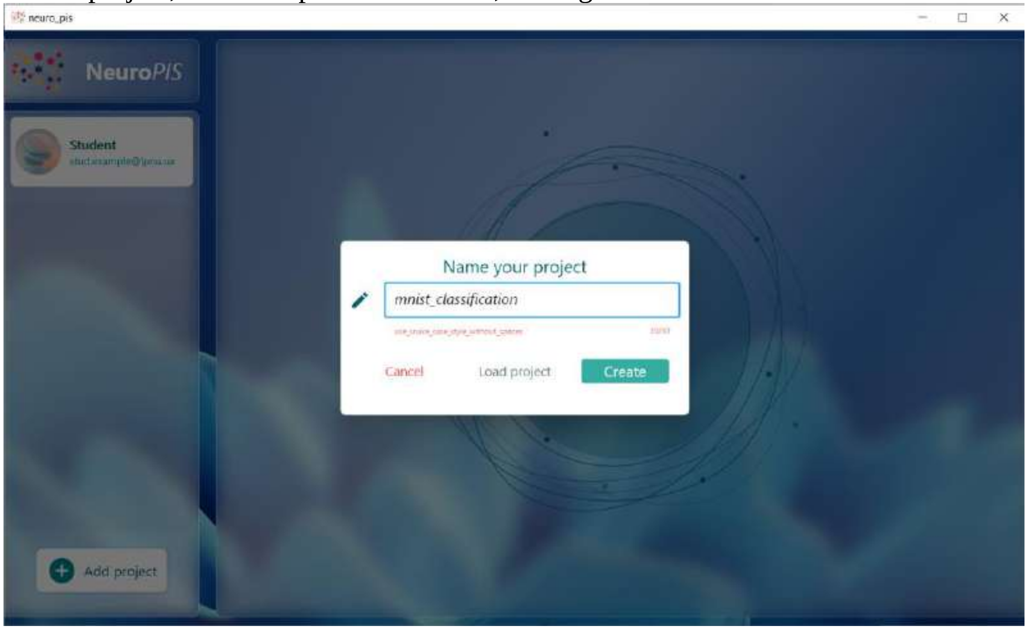


Figure 9: The main window of the application without any projects

To create a project, one must provide its name, see Figure 10.

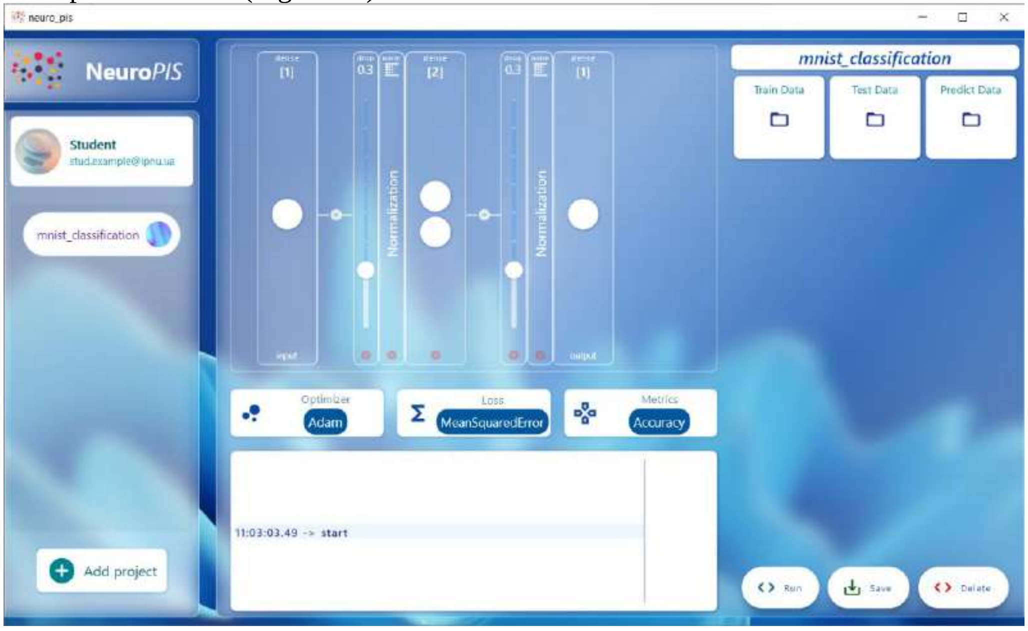


Name your project

mnist_classification

Figure 10: Adding of the new DNN project

After creating a new project, the system automatically fills in most parameters. Usually, the minimum or most probable values are used. As for the model, there are initially three layers. The last two are "composite" blocks (Figure 11).

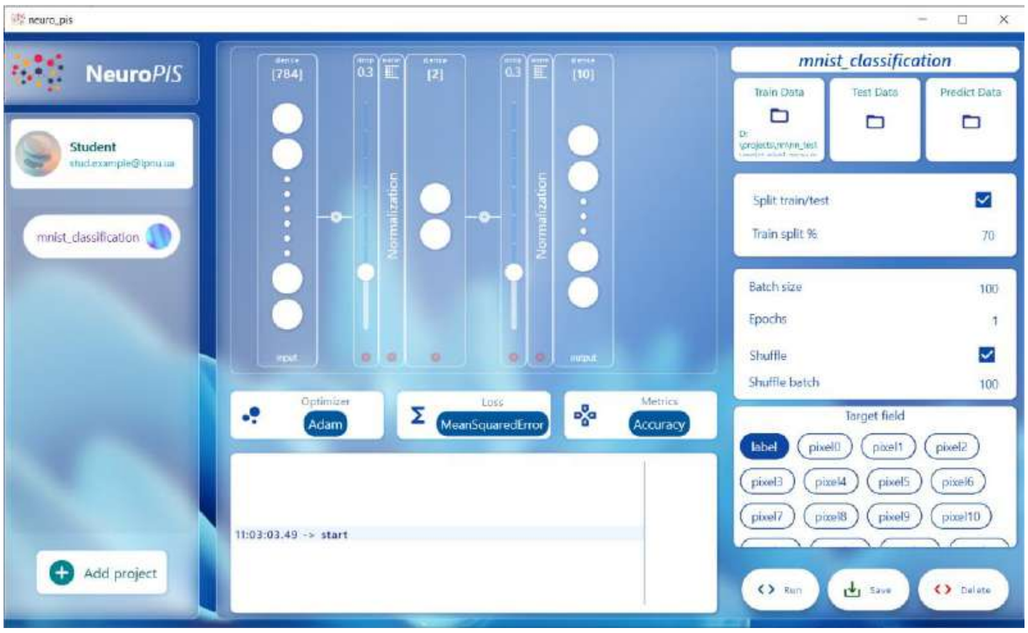


Optimizer: Adam
Loss: MeanSquaredError
Metrics: Accuracy

```
11:03:03.49 -> start
```

Figure 11: Project's start windows

After uploading the training and test dataset into the system, first and last layers will be automatically corrected. For the output Dense layer number of units is equal to the length of the unique values in the target field of the dataset. In the example - Figure 12, the target field is the first one – "label".



mnist_classification

Split train/test ✓
Train split % 70

Batch size 100
Epochs 1
Shuffle ✓
Shuffle batch 100

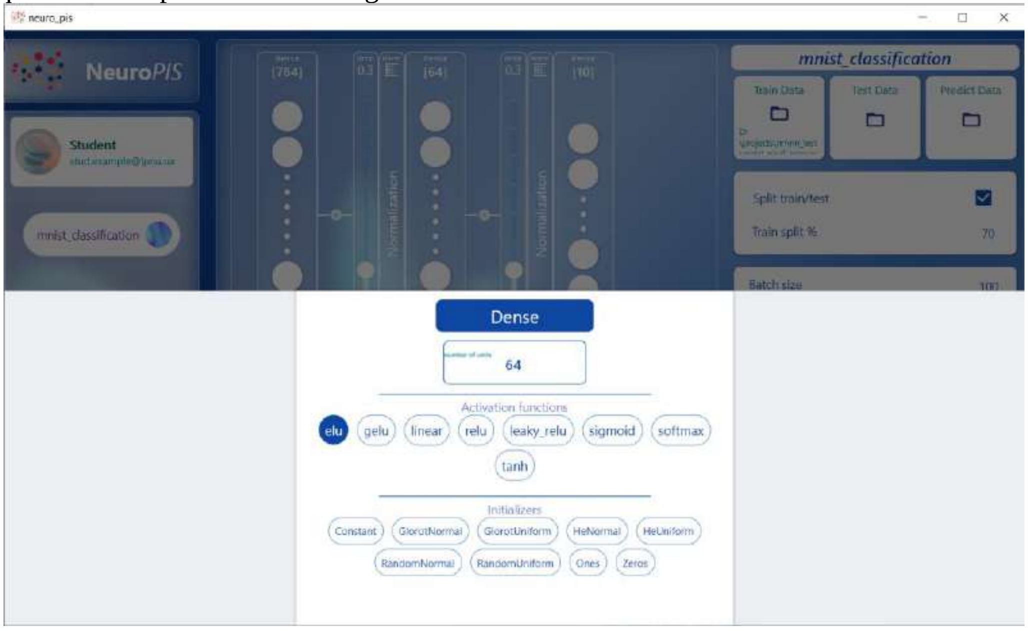
Optimizer: Adam
Loss: MeanSquaredError
Metrics: Accuracy

Target field: [✓ label] pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 pixel10

```
11:03:03.49 -> start
```

Figure 12: Updated state of the model

The next stage is the settings of the model and training parameters. The hidden Dense layer configuration is shown in Figure 13 with 64 units and elu activation. Dropout layers will be used with a rate equal to 0.1 to prevent overfitting.



Dense

number of units: 64

Activation functions: [✓ elu] gelu linear relu leaky_relu sigmoid softmax tanh

Initializers: Constant GlorotNormal GlorotUniform HeNormal HeUniform RandomNormal RandomUniform Ones Zeros

Figure 13: Hidden Dense layer configuration

Because projects' task is classification – the last layer has softmax activation. Loss function and metric are set as CategoricalCrossetropy (Figure 14). The last change is the number of epochs, and for the example, it is set to 3.

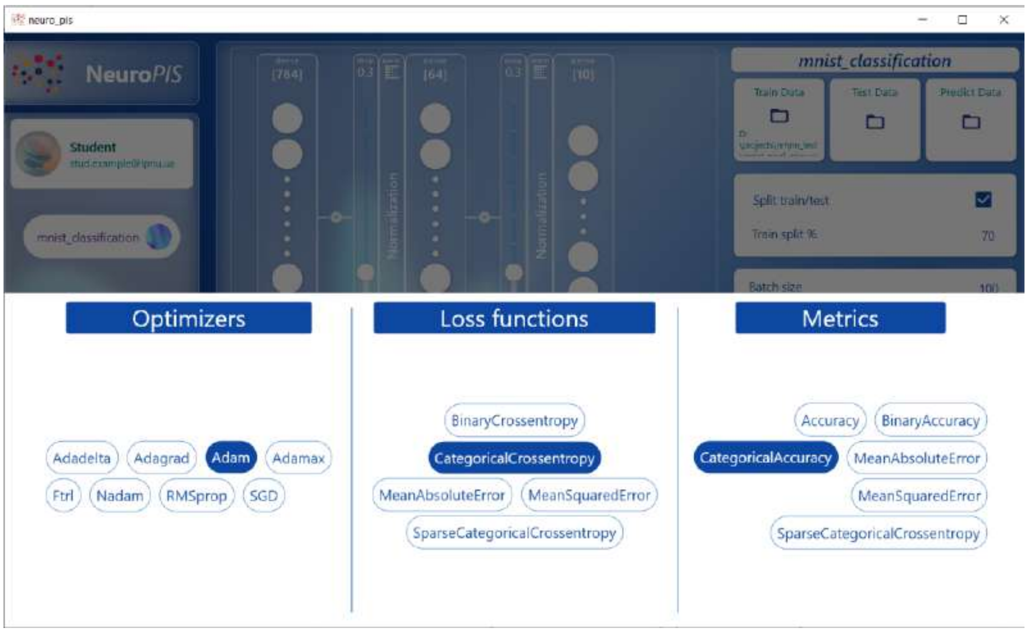
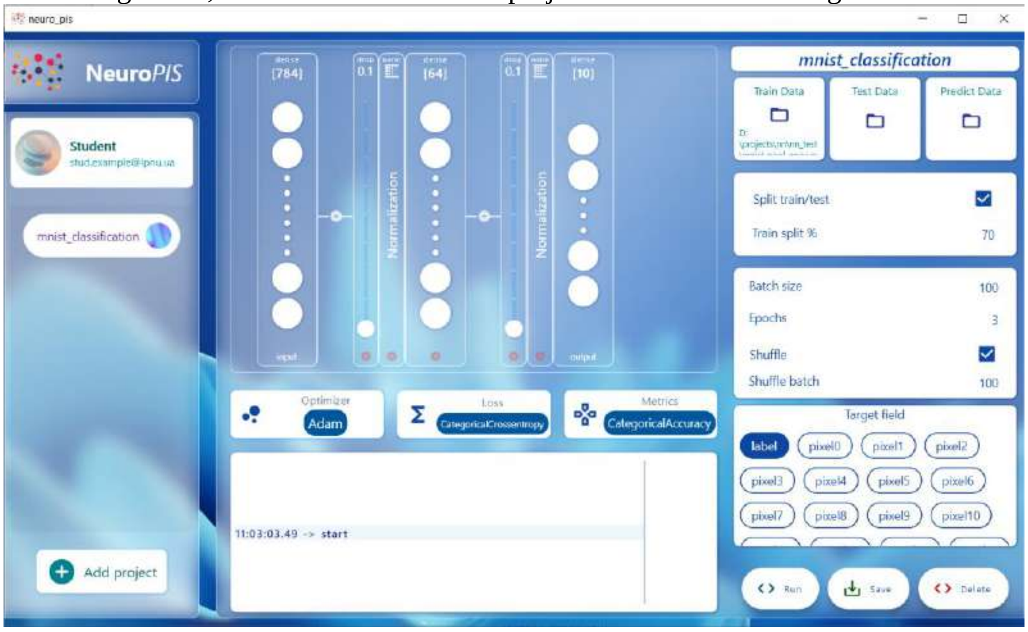


Figure 14: Training parameters settings

With all settings done, the main window of the projects will look like in Figure 15.

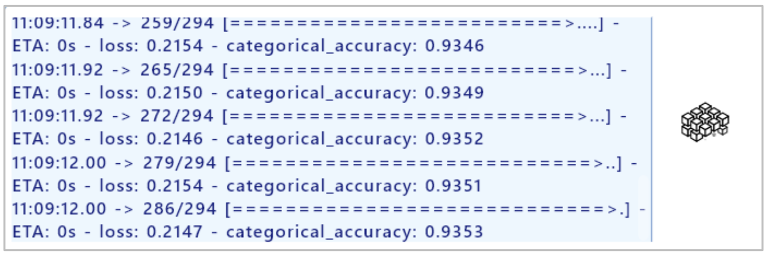


Optimizer: Adam
Loss: CategoricalCrossentropy
Metrics: CategoricalAccuracy
Split train/test ✓
Train split % 70
Batch size 100
Epochs 3
Shuffle ✓
Shuffle batch 100

```
11:03:03.49 -> start
```

Figure 15: Training parameters settings

After initiating the running process, the system will include all messages in the log widget like in Figure 16. While the file execution particular progress indicator is active, and the user can read current training parameters.



```
11:09:11.84 -> 259/294 [========================>....] -
ETA: 0s - loss: 0.2154 - categorical_accuracy: 0.9346
11:09:11.92 -> 265/294 [=========================>...] -
ETA: 0s - loss: 0.2150 - categorical_accuracy: 0.9349
11:09:11.92 -> 272/294 [=========================>...] -
ETA: 0s - loss: 0.2146 - categorical_accuracy: 0.9352
11:09:12.00 -> 279/294 [==========================>..] -
ETA: 0s - loss: 0.2154 - categorical_accuracy: 0.9351
11:09:12.00 -> 286/294 [===========================>.] -
ETA: 0s - loss: 0.2147 - categorical_accuracy: 0.9353
```

Figure 16: Logs of the training process

The end of the process notifies the user using the message in the log widget. The last numbers given in the list are the tested model performance parameters. For instance, after three epochs, our model gets categorical accuracy of 94%. The last step of the classical use case is saving the DNN. The user should choose a directory where the scripts and TensorFlow model will be saved. After the saving process, the user receives the project directory. Users can reuse projects like any python script. For example, the DNN model definition generated by the NUEROPIS looks like that:

```
import tensorflow as tf
def get_mnist_classification_model():
 inputs = tf.keras.layers.Input(shape = (784))
 x = tf.keras.layers.Dropout(0.1)(inputs)
 x = tf.keras.layers.BatchNormalization()(x)
 x = tf.keras.layers.Dense(units = 64, activation = tf.keras.activations.elu)(x)
 x = tf.keras.layers.Dropout(0.1)(x)
 x = tf.keras.layers.BatchNormalization()(x)
 outputs = tf.keras.layers.Dense(units = 10, activation = tf.keras.activations.softmax)(x)
 model = tf.keras.Model(inputs, outputs)
 model.compile(
 optimizer = tf.keras.optimizers.Adam(),
 loss = tf.keras.losses.CategoricalCrossentropy(),
 metrics = [tf.keras.metrics.CategoricalAccuracy()],
 )
 return model
```

Given the success of setting up and saving the DNN project, the classical use case example showed the correctness of the design decisions. The generated result proves the possibility of practical use of the developed information system.

4. Conclusion

The implementation of this work allowed us to analyze the availability of models of Deep Neural Networks for users in the form of information systems. The problem analysis highlights the main limitations, shortcomings, and typical algorithms in modern software solutions. Based on the study, we have: created a conceptual model of the system, identified the necessary functionality, prepared information content for the system project.

The logical continuation of the work was creating an object-oriented design of the information system in the form of multiple diagrams of the UML language. Created design yields descriptions of the logical and physical levels of functioning within the formed concept. Among the possible combinations of UML diagrams, we have described those that allowed detailing the system's use cases, specify a set of classes that can fully implement them, and instances' interaction and cooperation. A possible set of system design components is formed as a respective diagram. Created theoretical and graphic materials became the basis for developing our system software. The validation of the system confirmed the correctness of the results and thus the possibility of its practical application in creating Deep Neural Networks.

Further research will focus on implementing automating model construction and designing related software modules to expand the functionality and test their work.

5. References

[1] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 2019.

[2] D. Sarkar, R. Bali, T. Sharma, Practical Machine Learning with Python. Springer Science & Business Media, 2018.

[3] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks. Science, 313 (5786):504–507, 2006

[4] S. Cheong, Hands-On Image Generation with TensorFlow: A practical guide to generating images and videos using deep learning, Packt Publishing, 2020.

[5] A. Lamb, A Brief Introduction to Generative Models. arXiv preprint arXiv:2103.00265.
[6] G. Grolemund, H. Wickham. R for Data Science. O'Reilly Media, 2016.
[7] R. Karim, TensorFlow: Powerful Predictive Analytics with TensorFlow: Predict valuable insights of your data with TensorFlow, Packt Publishing, 2018.
[8] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning with Applications in R. Second Edition. Springer Science & Business Media, 2021.
[9] V. Sze,Y-H. Chen, T-J. Yang, J. Emer, Efficient Processing of Deep Neural Networks, Morgan & Claypool Publishers, 2020.
[10] D. Kingma, J. Adam, A Method For Stochastic Optimization. Conference paper at ICLR, 2015.
[11] A. Zheng, A. Casari, Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists. O'Reilly Media, 2018.
[12] J. Dawani, Hands-On Mathematics for Deep Learning: Build a solid mathematical foundation for training efficient deep neural networks, Packt Publishing, 2020.
[13] A. Vasyliuk, T. Basyuk, Construction Features of the Industrial Environment Control System, Proceedings of the 5rd International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021). Volume I: Main Conference, Kharkiv, Ukraine, April 22-23, 2021, Vol-2870: pp.1011-1025.
[14] E. Alcaide, E-swish: Adjusting Activations to Different Network Depths. arXiv preprint arXiv:1801.07145.
[15] V. Lakshmanan, S. Robinson, M. Munn, Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps, O'Reilly Media, 2020.
[16] M. Ekman, Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow, Addison-Wesley Professional, 2021.
[17] E. Thomas, J. Hendrik Metzen, F. Hutter, Neural architecture search: A survey, The Journal of Machine Learning Research 20.1, 2019, pp. 1-21.
[18] Deep learning in business analytics and operations research: Models, applications and managerial implications. arXiv preprint arXiv:1806.10897
[19] L. Pietro, G. Antiga, E. Stevens, T. Viehmann, Deep Learning with PyTorch: Build, train, and tune neural networks using Python tools, Manning, 2020.
[20] C. Bourez, Deep Learning with Theano: Perform large-scale numerical and scientific computations efficiently, Packt Publishing, 2017.
[21] L. Long, X. Zeng, Beginning Deep Learning with TensorFlow: Work with Keras, MNIST Data Sets, and Advanced Neural Networks, Apress, 2021.
[22] A. Vasyliuk, T. Basyuk, V. Lytvyn, Specialized interactive methods forusing data on radar application models, Proceedings of the 2nd International workshop on modern machinelearning technologies and data science (MoMLeT+DS 2020). Volume 1: Mainconference, Lviv-Shatsk, Ukraine, June 2-3, 2020, Vol. 2631, pp. 1–11.
[23] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Professional, 2003.
[24] R.Miles, K.Hamilton, Learning UML 2.0: A Pragmatic Introduction to UML, O'Reilly Media, 2006.
[25] S. Cai, S. Bileschi, E. Nielsen, Deep Learning with JavaScript: Neural networks in TensorFlow.js, Manning, 2020.
[26] Dart Programming Language Specification 6th edition draft. version 2.15-dev. URL: https://spec.dart.dev/DartLangSpecDraft.pdf
[27] P. Tyagi, Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web & Desktop, CRC Press, 2021.
[28] C. Zaccagnino, Programming Flutter: Native, Cross-Platform Apps the Easy Way (The Pragmatic Programmers), Pragmatic Bookshelf, 2020.
[29] M. Clow, Learn Google Flutter Fast: 65 Example Apps, Independently published platform, 2019.
[30] L. Moroney, The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, Apress; 1st ed. Edition, 2017.
[31] C. Aggarwal, Neural Networks and Deep Learning: A Textbook, Springer, 2018.