Education and Information Technologies
https://doi.org/10.1007/s10639-023-12349-5

# Investigating the usability of a new framework for creating, working and teaching artificial neural networks using augmented reality (AR) and virtual reality (VR) tools

**Roland Kiraly**[1] · **Sandor Kiraly**[1] · **Martin Palotai**[1]

Received: 18 April 2023 / Accepted: 7 November 2023

**Abstract**
Deep learning is a very popular topic in computer sciences courses despite the fact that it is often challenging for beginners to take their first step due to the complexity of understanding and applying Artificial Neural Networks (ANN). Thus, the need to both understand and use neural networks is appearing at an ever-increasing rate across all computer science courses. Our objectives in this project were to create a framework for creating and training neural networks for solving different problems real-life problems and for research and education, as well as to investigate the usability of our framework. To provide an easy to use framework, this research recruited five instructors who have taught ANNs at two universities. We asked thirty-one students who have previously studied neural networks to fill out an online survey about what were "the major difficulties in learning NNs" and the "key requirements in a Visual Learning Tool including the most desired features of a visualization tool for explaining NNs" they would have used during the course. We also conducted an observational study to investigate how our students would use this system to learn about ANNs. The visual presentation of ANNs created in our framework can be represented in an Augmented Reality (AR) and Virtual Reality (VR) environment thus allowing us to use a virtual space to display and manage networks. An evaluation of the effect of the AR/VR experience through a formative test and survey showed that the majority of students had a positive response to the engaging and interactive features of our framework (RKNet).

**Keywords** Artificial Neural Network · Augmented Reality · Virtual Reality · Education · Visualization

Extended author information available on the last page of the article

## 1 Introduction

Recently, Machine Learning and one of its subfields, Neural Networks (NN), have become increasingly popular due to their widespread use and success in various applications. NNs can assist humans in improving many industrial and professional processes, as well as enhance daily life (Sarker, 2021; Xiang et al., 2018; Tulbure et al., 2022; Kumar et al., 2022; Xin et al., 2021). Thus, in scientific research, including medical and biology-based use (Sarvamangala & Kulkarni, 2022; Tang et al., 2019) food science (Ma et al., 2022; Gorbachev et al., 2022), and research related to the spread of viruses (Wieczorek et al., 2020; Gao & Caines, 2022), the need to use neural networks is appearing at an ever-increasing rate. This success of NN applications has generated a huge amount of interest from practitioners and students, inspiring many to learn about this technology.

At first sight, constructing an NN for solving a problem is easy: we need to specify a structure and a loss function to optimize, before further optimizing it using gradient descent (Legaard et al., 2021; Behler, 2015). This means that the network feeds forward with just matrix multiplication and pointwise activations, before the network backpropagates using the multivariate chain rule when the update of the weights accordingly must be performed. When applying the theory to difficult problems, it turns out that understanding how deep learning works is very different from actually applying it successfully. There are plenty of pitfalls to watch out for before getting a neural network that does exactly what we want. For example, it is essential to optimize its structure to prevent over or underfitting, to get the network to converge (to a high-quality local minima) to make sure we have the right loss function, to do data augmentation correctly, etc. Moreover, in a complex NN, there are several network layers each with a different structure and underlying mathematical operations (López et al., 2022; Taye, 2023). Thus, students need to develop a mental model of not only how each layer operates, but also how to choose different layers that can work together to transform data. Therefore, learning about neural networks presents a significant challenge due to the complex relationship between basic mathematical operations and their overall integration within the network. In addition, the majority of neural networks run in virtual space, i.e. they are not visible to the human eye. Although for successful implementation it would often be necessary to make the "part of the virtual world where the neural networks run" available to the students in real-time and space, our tool provides such a user interface, thus making them literally "touchable" for their use.

The need to visualize artificial neural networks for computer scientists who need to construct neural networks to solve difficult problems and to control the working of their constructed NNs is evident (Shahroudnejad, 2021; Li et al., 2020). Thus, there is nowadays a strong interest in integrating VR in everyday working life (Babić & Meštrović, 2019; Bock & Schreiber, 2019).

### 1.1 Objectives

One of our objectives in this project was to create a framework for creating and training neural networks for solving different problems of real life and for research and education. In this paper, we focus on the investigation of the usability of our framework in education.

We sought answers to the following questions: (a) what are the major difficulties in learning NNs? (b) What are the key requirements in a Visual Learning Tool including the most desired features of a visualization tool for explaining NNs? (c) How usable is the created system?

Our system can work as a distributed system since it can use the available computers as resources. This is made possible by the distributed operation of the Erlang programming language[1], which allows us to run on several machines – even on several threads on one machine or process – thus enabling us to create nodes that communicate without using shared memory. (Naturally, here the elements of the neural network communication should be considered as network communication, and the node is the one running the Erlang process).

The framework also gives the opportunity for the individual nodes we start to perform a task specified with the help of a lambda function and to share their results with other members of the network. The neurons are capable of placing data to or from a distributed database reading out data.

Accordingly, we have developed a model and, based on the model, a prototype device that is suitable for the above tasks, i.e. it can show the structure of neural networks and make them controllable even for users who do not have sufficient knowledge of computer science and the disciplines that use it.

### 1.2 Research contribution

In this work, we contribute:

- Our system (RKNet) that contains an interactive visualization tool designed for both experts and non-experts to learn and analyse NN's structure, and monitor the working of NNs.
- Novel NN generator implemented in VR and AR environment.
- RKNet that provides the possibility of high error tolerance for the development of distributed systems, for messaging without shared memory between the nodes of a distributed system and provides many options that make the creation of neural networks simple.

Although framework was not created for educational purposes it is a behaviour-based system where we can practically create empty networks and then add functionality to them (with Lambda expressions that can be implemented as a higher-order, lazy evaluation function).

Although the creation of the network is a programmer's task, its use, visual control and monitoring require only user skills.

[1] https://www.erlang.org/

## 2 Method

### 2.1 Literature review

An increasing amount of research utilizes interactive visualizations to elucidate the operational mechanisms of neural networks. Harley's visualisation tool focuses on demonstrating the high-level model structure and connections between layers of Convolutional Neural Networks (CNN), Karpathy's extended ConvNetJS tool allows us to formulate and solve Neural Networks in Javascript (Karpathy 2016). TensorFlow Playground (Smilkov et al., 2017) and GAN Lab (Kahng et al., 2019) can help students to learn about dense neural networks and generative adversarial networks (GANs) respectively. Wang et al. (2021) presented the CNN Explainer, an interactive visualization tool designed for non-experts to learn and examine convolutional neural networks (CNNs), a foundational deep learning model architecture. Apart from the aforementioned tools, there are several others designed for non-experts (Olah 2014; Smilkov et al., 2017; Norton & Qi 2017) but most of them are developed to help NN experts analyse their models and predictions (Bilal et al. 2018; Garcia et al. 2018; Harley, 2015; Hohman et al., 2019; Kahng et al., 2018; Liu et al., 2017a, b). Mohamed et al. (2022) presented a review of visualisation-as-explanation techniques for convolutional neural networks and their evaluation. Zhang et al. (2021) evaluated the visualization performance of CNN models using driver model. 3D visualization of deep learning algorithms were developed for both experts and non-experts with an interactive user interface that allows interactive exploration on different levels of detail (Bock & Schreiber, 2019; Jin et al., 2020; Schreiber & Bock, 2019).

Meissler et al. (2019) examined how CNNs can be visualized in Virtual Reality and developed a software prototype based on the Unity platform and STEAMVR. Their deep learning networks are defined using Keras (Chollet et al., 2015) which are used as a high-level layer on top of Tensorflow (Abadi et al., 2015).

Queck et al. (2022) designed and implemented an ANN visualization in virtual reality (VR) specifically targeted at machine learning users. Their approach was especially well-suited for teaching and for individuals who are new to machine learning or lack expertise but wish to gain an understanding of the overall workings of neural networks. The common feature of the aforementioned tools is that they are visual systems, created for educational purposes and do not have their own framework behind them.

Although different frameworks have been developed to enhance experts' work constructing NNs to solve difficult problems in real life, such as TensorFlow, Keras, PyTorch, Theano, Deeplearning4j or Microsoft CNTK they do not contain visual tools that can help users in analysing the constructed NNs by checking the state of a neuron or a layer. Although none of them uses virtual reality (VR) and augmented reality (AR) AR does permit the superimposition of digital content on top of the real-world environment through the use of a smartphone, tablet or virtual reality (VR) headset (Carmigniani et al. 2011). AR presents an opportunity to deliver a 'virtual' object-based learning activity. The dominant feature of VR is the ability to promote a higher level of immersion than other media. Immersive VR puts the user in an environment that takes after the real world and feels to some extent like it, with the person having a sense of self localization (Psotka 1995).

VR with head-mounted displays has proven itself as a learning medium in the engineering field (Abulrub et al. 2011). Indeed, there is evidence that a virtual learning environment can achieve better learning outcomes than traditional teaching (Alhalabi, 2016).

Our framework can be widely used to construct NNs to solve problems in real life. It can also be parameterized easily and the created NNs can be visually presented in an AR and VR environment. Moreover, the interaction can also take place in this environment. Using AR and VR tools, the created NNs can be shown in a virtual environment and the control, programming and management of the networks become available here. By using the framework for educational purposes, students without fundamental or relatively low-level programming knowledge can construct different NNs to solve problems and can also examine the working of NNs.

### 2.2 The structure of the system

Since one of the goals of creating our framework was the process of creating, programming and testing NNs, as well as the process of creating NNs, to simplify their programming and testing, as well as to create NNs that are suitable for solving real-life problems easily, we first created the subsystem in the Erlang language. In this subsystem, the configuration settings can be used to define the number of neurons in the NN and their connections, before starting them using a command issued on a graphical interface. The neurons in the network can be parameterized with Lambda expressions. (*Neural Network Framework*.)

Figure 1 shows the structure of our framework. Currently, it shows a very simple NN (*Neural Network 1*). The central item of a NN, created by our framework, is the node, called SNeuron. (S1, S2,…, S5 in Fig. 1). One SNeuron contains two neurons (neuron pairs). One of them is responsible for communication and the other for performing the currently imposed task. SNeurons can perform tasks specified by Lambda functions, and share their results with other neuron pairs in the network. A single SNeuron can communicate with other SNeurons if they share the same password stored in a common cookie file on the host computer whilst being connected to a network. This mode of operation is described in the Erlang language and each SNeuron is an Erlang process, allowing our system to function as a distributed system.

SNeurons are able to cooperate and directly exchange data with other SNeurons defined in their configuration. Furthermore, they are also able to place data in a distributed database or read data from it. Based on a pre-set frequency or in an impromptu manner, the SNeurons in the network can learn the data collected by the other SNeurons by reading them from the database. That is, they use each other's data and learn based on the experiences of the others.

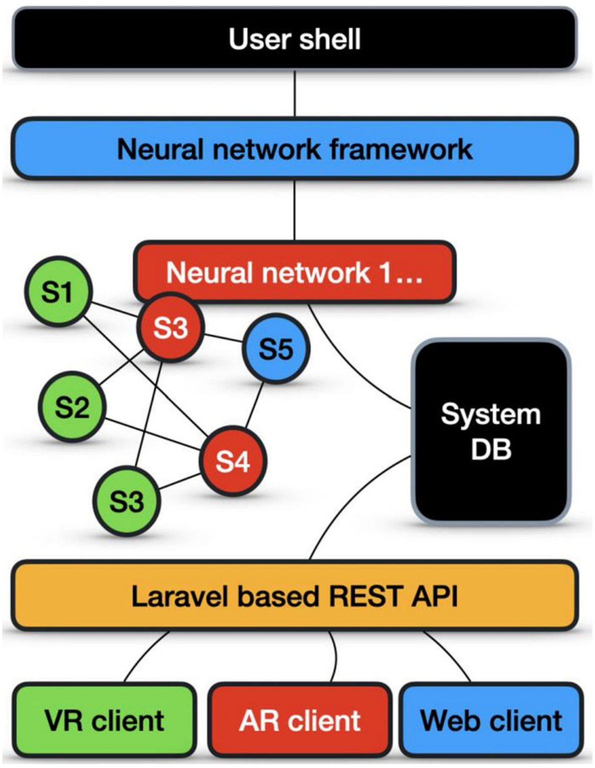


**Fig. 1** Block diagram of the frame work. Source: created by the authors

The system monitors and presents graphically the neural network as an edge-tagged directed graph through database snapshots in pseudo real time. The graphic display module and its extensions make our framework suitable for using the virtual environment. The part of the module responsible for communication was implemented on a web server using a Laravel-based REST (*Representational State Transfer*) API (*Application Program Interface*).

This SQL-based (*Structural Query Language*) database makes it possible to visualize the neural network in real time for the virtual environment. The functions of the SNeuron are:

- It can communicate with the rest of the network – with other neurons;
- It can read data from the distributed database and can write data there;
- It can register itself in the database of the system in order to provide data to the connected interfaces.

The database thus provides an accurate and up-to-date status of the network, which can return the connections and the current tasks of the neurons in real time. Every time there is a change, the database is updated, which triggers an event in the viewer, making the network quasi-animated on the visual interfaces.

In this way, we can actually query a snapshot of the network, which is suitable for its visual display and control

The neural network provides a significant part of the system, but not its full functionality. A very important element of the entire framework is the UI (*User Interface*) that provides the control, as well as the API that implements the communication among the subsystems.

### 2.3 Rendering

When we create a network with its neurons (*SNodes*) in the system, we enter the number of neurons in the neural network. They are initially equal and have no task. We need to define which neurons can communicate with each other. As an extra function, we can group neurons by way of specifying their group ID, which can be used to refer to them later.

The VR system runs under the Unity Engine, which can display neural networks in the virtual environment. This subsystem also provides the opportunity to generate learning networks, which are only useful for educational purposes and cannot solve complex problems. It automatically colours, and also provides the possibility to introduce individual templates. For the efficient use of the framework in multiple contexts and externally, we have implemented a template system, which enables a relative positioning system to place the displayed networks in the visible middle thanks to the possible dimensions of the data received from the system. In virtual reality, individual nodes can be grasped, touched, rotated and their functions managed. We can examine the functions of running neurons by holding them in our hands. In VR, there are two ways of interacting with the environment, one is our hands, and the other is a laser pointer device that allows us to simply grab distant nodes and then pull them closer to us. This helps to manage and display larger networks to the user.

To illustrate the model, we created a mobile and Unity-based interface that can be used through VR glasses for managing and illustrating the neural network. Figure 2 shows a randomly generated NN without functionality to represent how we can choose a node and check its state.

### 2.4 Timing and visualization

The working of the neural network in the system is very fast, especially if we use Lambda functions defined with simple functions. Although in the case of complex calculations, it is very useful, for visual display it is not feasible, since it cannot be followed with the eyes and in the VR or AR GUI only the final result is visible. To solve this problem, we have enabled the neural network to slow down its operation during the creation of the network and during its working as well. It is possible to specify the speed of network creation, as well as the time elapsing between sending messages for each network element group type. In addition to specifying the speed, it is also possible to select certain processes in the network that should not be executed automatically, but upon external intervention, as for example, in virtual

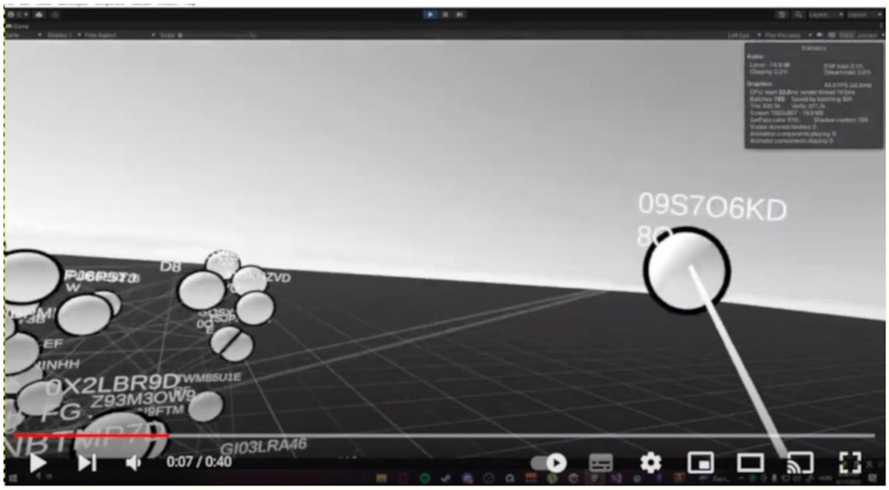


**Fig. 2** Neural Network Visualization VR in RKNet. Now, neurons have functionality. Source: screenshot from the video. https://www.youtube.com/watch?v=zoD0oWan7u8

reality with a click or a VR movement. The VR software of the system sends a message to the REST API such as grabbing one of the neurons or touching it through a corresponding endpoint if a command is issued in VR. The REST API then calls a function in the Erlang module through a purpose-written function, instructing the neural network to proceed or to send a message. This function also enables an interactive illustration for the user.

## 2.5 Defining NNs for the visualization interface of RKNet

When we create a NN with its neurons in the system, we specify the neurons in the neural network and the communication types they must use. We also specify with whom they will communicate in the NN. This can be one or more neurons or the console from which we start the NN. It may also be another node or the rest of the system. We also have to define the neighbours of each node, i.e. neuron, to whom a message must be sent in connection with an event taking place in a network. We can also specify, for example, what starting value an input neuron should start with, i.e. what it should send to the neurons of the hidden layer. In the example below, we defined a neural network representing a perceptron.

In the example, see Fig. 10 in the Appendix, the neural network is given in a list. Each list element is defined in an ordered *n-tuple*, where the elements of the tuple describe the properties of the neuron.

$$\{\text{iln1},\ \text{input},\ [\{\text{hln1},\ \{0.1,\ 0.4\}\}]\}$$

*iln1* is the name of the neuron, i.e. the internal identifier of the system. The input atom tells us that this element is an input neuron. The third element in the tuple is also a list that specifies the neighbour numbers of the neuron. It will send a message to them, if we instruct it to do so, in the Lambda function describing its function. This function can be defined by using the console of the framework. In the list describing the neighbourhood, since this is an input neuron, we can specify the ID of the neighbours, as well as the initial values that must be sent to each neighbour at the start and the value of the weight for each. An initial input value does not need to be specified for the neurons of the hidden layer, as they send a message based on what is described in their Lambda function. (It is also possible to specify a weight here, but in this example, we defined the weights assigned to the edges for the input.) The situation is the same in the case of the output layer, but here we can specify that the result is written to the console for verification and testing purposes. This can happen if, for example, we do not want to calculate its error with a cost function, but want to check the result produced by the output neuron at the output, as in the case of the above output neuron:

$$out1, output, [console].$$

Figure 3 shows a simple perceptron model including the input values of the NN, Fig. 4 represents its visualization with the output value.

Without defining the lambda functions, that describe the functioning of the neurons the NN does nothing. Therefore, our Erlang list (see Fig. 10) requires adding the Lambda functions. Figure 11 in the Appendix shows the completed Erlang list with the name of the Lambda functions applied in the definition of the NN.

Knowing the implementations of the Lambda functions or watching the video, it can be seen how the NN works. The three input neurons send (*0.4, 0.5, 0.2*) values to the hidden neuron, which multiplies them with the weights (*0.4, 0.2. 0.6*), the sum of the resulting three values and send to the output neuron. This neuron displays the true value to the output if the given value is more the 0.5, otherwise it is false. The lambda functions can be defined both in the Erlang list or outside the list. Figure 12 in the Appendix shows the Lambda function that defines the behaviour of the output neuron.

In this case, the function of the output neuron waits for a message. If the content of the message is the *input, Value* pair, the function adds its current value (*State*) to the one received in the message. If this value is greater than 0.5, the result will be *true*, otherwise *false*. Then, at each step, it saves its own state for the system database (from which, for example, the VR interface can query it), and then calls itself with the new state. If it receives a stop message, it stops, that is, it does not call itself again.

We can specify the neurons of the hidden and input layers in the same way as its Lambda functions and we can assign them to the appropriate group in the definition of the NN. Functions are defined at the start of the neural network and must be given to all neurons of that type by specifying the name label, as we can see it in Fig. 11.

At the start of the network, the system will start the *input_neuron* function in the case of the input neurons in the input layer, and naturally, the *hidden_neuron*

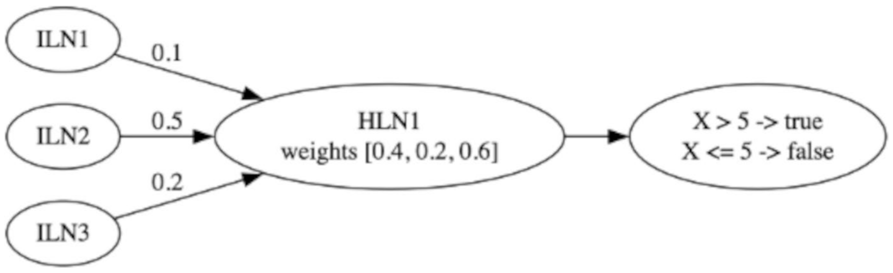


**Fig. 3** A simple perceptron model. Source: made by the authors

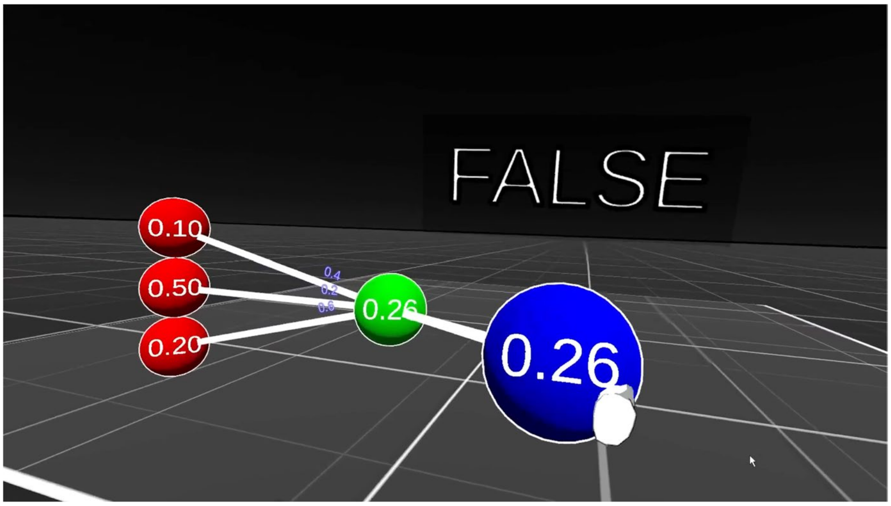


**Fig. 4** The visualization in AR of the previous perceptron model. Screenshot from the video. Source: https://youtu.be/ghI-OezDAFE

function in the case of the hidden layer and the *output_neuron* function in the case of the output. After that, we just have to start the network either from the console or from the REST API endpoints and its operation can be monitored on one of the visual interfaces.

By defining more complex and complicated Lambda functions, neurons can overcome more difficult tasks, thus the created NNs in the system can work as a complex, highly error-tolerant distributed system, whose functions are defined in the Lambda functions of each individual neuron.

In the next example, we define a simple NN to study its working including backpropagation. Figure 13 in the Appendix shows the Erlang list with the name of the Lambda functions.

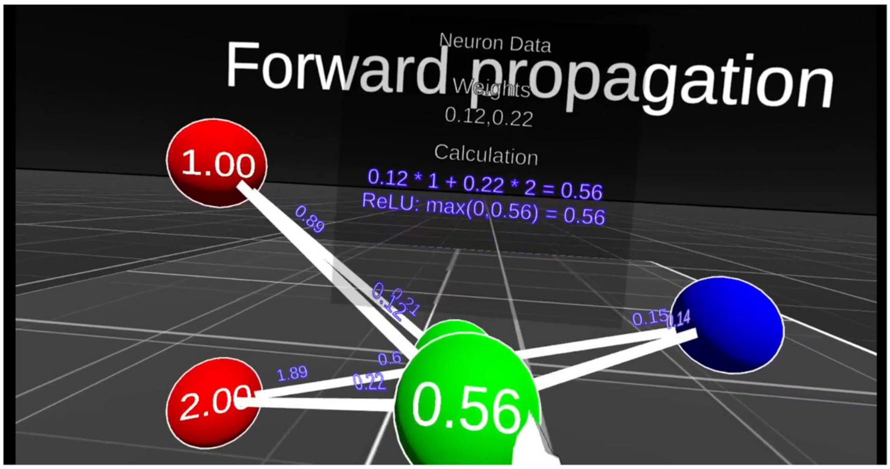


**Fig. 5** Investigation of the operational of a Neural Network in. (In the calculation of the MSE value, we used 0.5 instead of 1 to ease the calculation of the derivative.) Screenshot of the video. Source: https://youtu.be/Ak-j9_RD-Rk

After the implementation of the functions of the layers and starting the NN, we can study the working of the NN through the VR output screen. Figure 5 represents a screenshot of the video.

## 3 Formative research to identify the learning challenges faced by the students

Before designing the educational part of our visualization tool, we recruited 5 instructors (5 female) who have taught NNs at two universities. We interviewed them one-on-one in a conference room (2/5) and via video-conferencing software (3/5); each interview lasted about 25 minutes. Through these interviews, we learned that instructors currently use simple illustrations with simple examples to explain NN concepts and give URLs of demonstration videos about the working of NNs.

We asked our BSc and MSc students who have previously studied neural networks to fill out an online survey. We received 31 responses (4 female, 27 male). Among them, 26 were BSc. students and 5 were MSc. students. We asked participants what were "the major difficulties in learning NNs" and the "key requirements in a Visual Learning Tool including the most desired features of a visualization tool for explaining NNs" they would have used during the course.

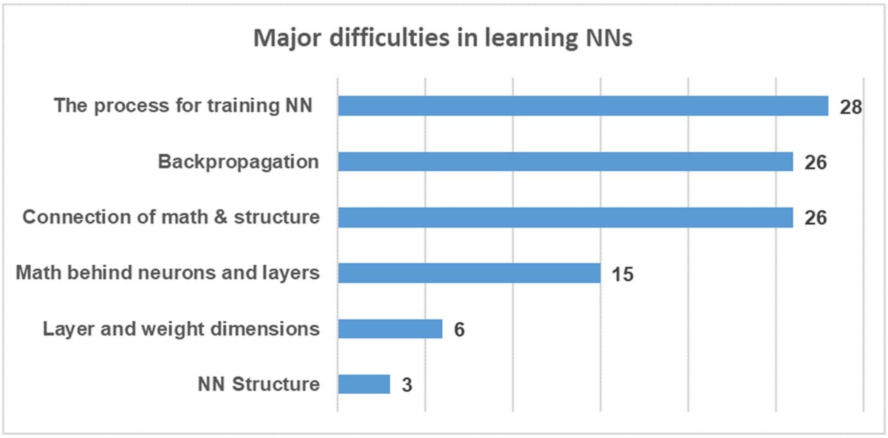


**Fig. 6** Survey results from 31 students who have already learned about NNs. Major difficulties encountered during learning

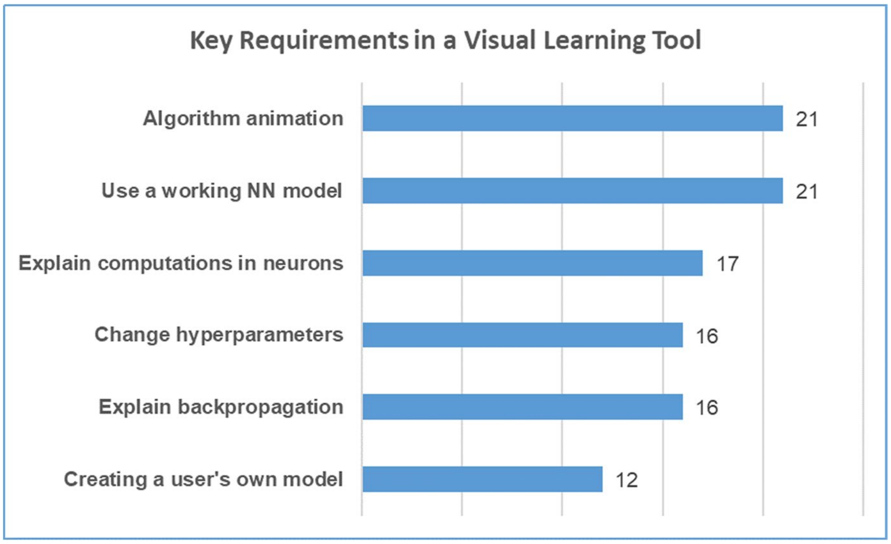


**Fig. 7** Survey results from 31 students who have already learned about NNs. Key requirements in a Visual Learning Tool for NNs

Participants were allowed to choose from a range of options (see Figs. 14 and 15). The aggregated results of this survey are shown in Fig. 6 and 7. (We used Microsoft Excel for data analysis.)

During the development of the AR/VR subsystem, we tried to follow the result of this survey to fulfil the requirements. After completing the current version of the system, we constructed NNs previously seen with varying activation functions and learning rates, as well as additional simple NN with two hidden layers and two output neurons, each using different activation functions and learning rates.

## 4 Results

We conducted an observational study to investigate how our students would use this system to learn about NNs, and also to test the usability of the system. We recruited 21 student participants from our university (3 female, 18 male). Four students were MSc. and the others were BSc. students. All participants were interested in learning NNs, and none of them had known RKNet before. Participants self-reported their level of knowledge on non-neural network machine learning techniques (see Figs. 16 and 17). We also provided a feature checklist, which outlined the main features of our tool and asked them to try the first NN with different activation functions and learning rates, then the second one. During the study, participants were asked to think aloud and share their computer screen with us; they were encouraged to ask questions when necessary. The exit questionnaire included a series of 7-point Likert-scale questions about the utility and usefulness of different views in RKNet (Fig. 8 and 9). All average Likert ratings were above 6 except for the rating of "interactive calculation views". From the high ratings and our observations, participants found our tool easy to use and understand, retained a high engagement level during their session, and eventually gained a better understanding of NN concepts.

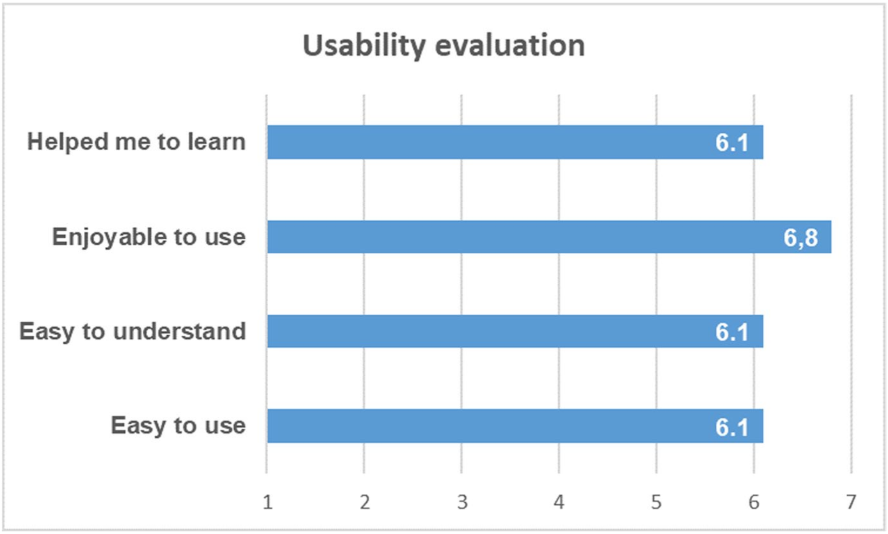


**Fig. 8** Average ratings from 21 participants regarding the usability and usefulness of RKNet. Participants thought RKNet was enjoyable, easy to use and helped them learn about ANs

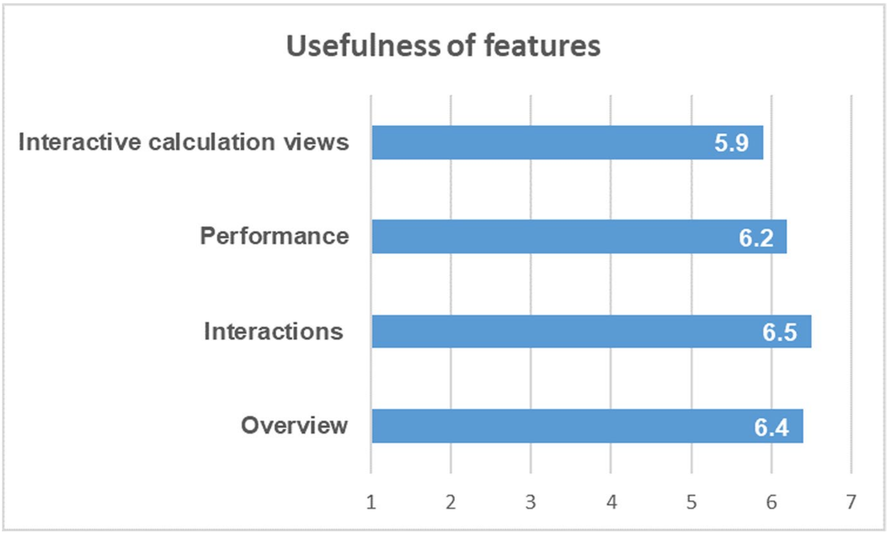


**Fig. 9** Average ratings from 21 participants regarding the usability and usefulness of RKNet. Usefulness, especially animations were rated favourably

Another useful feature of RKNet that participants mentioned was the interactions, which received the highest rating in the exit questionnaire (Fig. 8). We found that interactions helped to increase participants' engagement level (e.g., spending more time and effort) and made RKNet more enjoyable to use.

The main student objection was that hyperparameters cannot be varied in AR.

## 5 Conclusions and future work

RKNET is a framework with which we can create Neural Networks in the Erlang programming language to solve various tasks. Network operation is monitored in a virtual environment. We also made the VR tool suitable for non-experts as it allows an easy and accessible introduction to NNs. We plan to provide the option of changing hyperparameters interactively in VR view. The system is a prototype, so we are constantly developing it, and we add new functions to the system based on needs.

In addition to VR, we also targeted the AR environment. Here, the aesthetic placement of the network in real space caused problems, but we will solve this soon with the help of plane detection. So AR will also become an available option (although currently, it is only available in the test phase, as we focused on VR).

We plan to create a higher level of interaction, where all elements and functions of the network can be created even from the VR environment. We also aim to supplement the framework with a web-based education system. Later, we plan to release RKNet as an open-source software.

## Appendix for codes and data

**Fig. 10** The definition of a perceptron model in an Erlang list

```
network_model() ->
[
{iln1, input, [{hln1, {0.1, 0.4}}]},
{iln2, input, [{hln1, {0.5, 0.2}}]},
{iln3, input, [{hln1, {0.2, 0.6}}]},
{hln1, hidden, [{out1}]},
{out1, output, [{console}]}
].
```

```
network_model() ->
[
{iln1, input, [{hln1, {0.1, 0.4}}], input_neuron},
{iln2, input, [{hln1, {0.5, 0.2}}], input_neuron},
{iln3, input, [{hln1, {0.2, 0.6}}], input_neuron},
{hln1, hidden, [{out1}], hidden_neuron},
{out1, output, [{console}], output_neuron}
].
```

**Fig. 11** The definition of a perceptron model in an Erlang list including the Lambda functions

```
output_neuron({Name, ActValue}) ->
receive
      {input, Value} ->
            [Value]),
            State = ActValue + Value,
            if
                  State >= 0.5 ->
                        mconsole(true);
                  true -> mconsole(false)
            end,
            save_neuron_state(Name, State),
            output_neuron({Name, State});
      stop -> stop;
      _ -> output_neuron({Name, ActValue})
end.
```

**Fig. 12** The definition of a perceptron model in an Erlang list including the lambda functions

```
neurons_back_prop() ->
[
{x1, input, [{h3, 1, 0.12}, {h4, 1, 0.22}], bp_input],
{x2, input, [{h3, 2, 0.21}, {h4, 2, 0.6}], bp_input},
{h3, hidden, [{o5, 0.14}], bp_hidden},
{h4, hidden, [{o5, 0.15}], bp_hidden},
{o5, output, [{x1, x2, h3, h4}], bp_output]
].
```

**Fig. 13** The definition of an NN (without biases) by an Erlang list

| Students: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gender(Male/Female)** | M | F | M | F | F | M | M | M | M | M | M | M | M | M | M | **M** | M | M | M | M | M | M | M | M | M | **M** | F | M | M | M | M |
| **MSc (M) or BSc(B)** | B | M | B | M | B | B | B | B | B | B | M | M | B | B | B | B | B | M | B | B | B | B | B | B | B | B | B | B | B | B | B |
| **The process for training NN (Yes or No)** | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| **Backpropagation (Yes or No)** | Y | Y | Y | Y | Y | Y | N | Y | N | Y | Y | N | Y | Y | Y | Y | Y | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| **Connection of math & structure (Yes or No)** | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | Y | Y | N | Y | N | Y | Y | N | N | Y | Y | Y | Y | Y | Y | Y | Y |
| **Math behind neurons and layers** | Y | Y | N | N | Y | N | Y | Y | N | Y | Y | N | Y | N | Y | N | Y | Y | N | N | N | N | N | Y | N | Y | N | Y | N | N | Y |
| **Layer and weight dimensions (Yes or No)** | N | N | N | N | N | N | Y | N | N | N | N | N | N | Y | N | N | Y | N | N | N | N | N | N | N | Y | N | Y | N | N | Y | N |
| **NN Structure (Yes or No)** | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | Y | N | N | N | N | N | N | N | N | N | Y | N | N | Y | N |

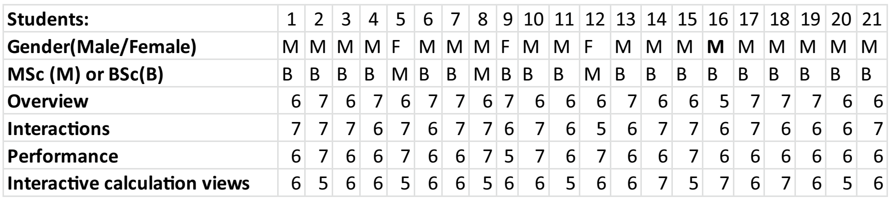
**Fig. 14** Data source of "Major difficulties in learning NNs"

| Students: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gender(Male/Female)** | M | F | M | F | F | M | M | M | M | M | M | M | M | M | M | **M** | M | M | M | M | M | M | M | M | M | **M** | F | M | M | M | M |
| **MSc (M) or BSc(B)** | B | M | B | M | B | B | B | B | B | B | M | M | B | B | B | B | B | M | B | B | B | B | B | B | B | B | B | B | B | B | B |
| **Algorithm animation** | Y | N | Y | Y | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | Y | N | Y | N | Y | Y | Y | N | N | Y | Y | Y | N | N | Y | N |
| **Use a working NN model** | Y | Y | Y | Y | Y | Y | N | Y | N | Y | Y | N | Y | N | N | Y | N | Y | N | Y | Y | Y | Y | N | Y | Y | N | Y | N | Y | Y |
| **Explain computations in neurons** | Y | N | N | N | Y | N | Y | N | Y | Y | Y | Y | Y | N | Y | Y | N | Y | N | Y | Y | N | N | Y | N | N | N | Y | Y | N | Y |
| **Change hyperparameters** | Y | Y | N | N | Y | N | Y | Y | N | Y | Y | N | Y | Y | Y | N | N | Y | N | Y | N | N | N | Y | N | Y | N | Y | N | N | Y |
| **Explain backpropagation** | N | Y | Y | Y | N | Y | Y | N | Y | N | N | N | Y | Y | Y | N | Y | N | N | N | N | Y | Y | N | Y | N | Y | N | Y | Y | N |
| **Creating a user's own model** | N | N | Y | Y | N | Y | N | Y | N | N | N | Y | N | N | N | Y | Y | N | Y | N | N | Y | N | N | Y | N | Y | N | N | Y | N |

**Fig. 15** Data source of "Key Requirements in a Visual Learning Tool"

| **Students:** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gender(Male/Female)** | M | M | M | M | F | M | M | M | F | M | M | F | M | M | M | **M** | M | M | M | M | M |
| **MSc (M) or BSc(B)** | B | B | B | B | M | B | B | M | B | B | B | M | B | B | B | B | B | B | B | B | B |
| **Easy to use** | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| **Easy to understand** | 6 | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 |
| **Enjoyable to use** | 7 | 7 | 7 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 7 | 7 | 7 | 7 | 7 | 6 | 7 | 6 |
| **Helped me to learn** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 | 6 | 6 | 6 | 7 | 6 | 6 | 6 | 6 | 6 | 6 |

**Fig. 16** Data source of "Usability evaluation"



| Students: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gender(Male/Female)** | M | M | M | M | F | M | M | M | F | M | M | F | M | M | M | **M** | M | M | M | M | M |
| **MSc (M) or BSc(B)** | B | B | B | B | M | B | B | M | B | B | B | M | B | B | B | B | B | B | B | B | B |
| **Overview** | 6 | 7 | 6 | 7 | 6 | 7 | 7 | 6 | 7 | 6 | 6 | 6 | 7 | 6 | 6 | 5 | 7 | 7 | 7 | 6 | 6 |
| **Interactions** | 7 | 7 | 7 | 6 | 7 | 6 | 7 | 7 | 6 | 7 | 6 | 5 | 6 | 7 | 7 | 6 | 7 | 6 | 6 | 6 | 7 |
| **Performance** | 6 | 7 | 6 | 6 | 7 | 6 | 6 | 7 | 5 | 7 | 6 | 7 | 6 | 6 | 7 | 6 | 6 | 6 | 6 | 6 | 6 |
| **Interactive calculation views** | 6 | 5 | 6 | 6 | 5 | 6 | 6 | 5 | 6 | 6 | 5 | 6 | 6 | 7 | 5 | 7 | 6 | 7 | 6 | 5 | 6 |

**Fig. 17** Data source of ”Usefulness of Features”

**Funding** Open access funding provided by Eszterhazy Karoly Catholic University.

**Data availability** The author confirms that all data generated or analysed during this study are included in this published article. Furthermore, primary and secondary sources and data supporting the findings of this study were all publicly available at the time of submission.

## Declarations

**Conflict of interest** We have no conflict of interest to declare.

## References

Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/. Accessed 20 Oct 2022.

Abulrub, A. G., Attridge, A. N., & Williams, M. A. (2011). Virtual Reality in engineering education: The future for creative learning. *IEEE Global Engineering Education Conference (EDUCON), 2011*, 751–757. https://doi.org/10.1109/EDUCON.2011.5773223

Alhalabi, W. (2016). Virtual reality systems enhance students' achievements in engineering education. Behaviour & Information Technology 35, 11 (July 2016), 919–925. https://doi.org/10.1080/0144929X.2016.1212931

Babić, K., and Meštrović, A. (2019). Visualizations of the training process of neural networks. 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2019, pp. 1619-1623. https://doi.org/10.23919/MIPRO.2019.8757142

Behler, J. (2015). Constructing high-dimensional neural network potentials: A tutorial review. *Wiley Online Library*. https://doi.org/10.1002/qua.24890

Bilal, A., Jourabloo, A., Ye, M., Liu, X. and L. Ren (2018). Do Convolutional Neural Networks Learn Class Hierarchy? IEEE Transactions on Visualization and Computer Graphics, 24(1):152–162, Jan. 2018. https://doi.org/10.1109/TVCG.2017.2744683

Bock, M. and Schreiber, A. (2019). Visualization of neural networks in virtual reality using UNREAL ENGINE. In Proceedings of the 24th ACM Symposium on Virtual Reality Software and

Technology, ser. VRST ’18. New York, NY, USA: ACM, 2018, pp. 132:1–132:2. [Online]. Available at: https://doi.org/10.1145/3281505.3281605. Accessed 25 Oct 2022

Carmigniani, J., et al. (2011). Augmented reality technologies, systems and applications. *Multimedia Tools and Applications, 51*(1), 341–377. https://doi.org/10.1007/s11042-010-0660-6

Chollet F. et al (2015). Keras. https://keras.io. Accessed 20 Oct 2022.

Gao, S., Caines, P.E. (2022). Transmission Neural Networks: From Virus Spread Models to Neural Networks. IEEE (Version: August 5, 2022). https://doi.org/10.48550/arXiv.2208.03616

Garcia, R., Telea, A. C., da Silva, B. C., Torresen, J., & Dihl Comba, J. L. (2018). A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers & Graphics, 77*, 30–49. https://doi.org/10.1016/j.cag.2018.09.018

Gorbachev, V., et al. (2022). Artificial Neural Networks for Predicting Food Antiradical Potential. *Appl. Sci. 2022, 12*(12), 6290. https://doi.org/10.3390/app12126290

Harley, A. (2015). An interactive node-link visualization of convolutional neural networks. Springer, Cham. https://doi.org/10.1007/978-3-319-27857-5_77

Hohman, F., Kahng, M., Pienta, R. and D. H. Chau (2019). Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2674–2693, Aug. 2019. https://doi.org/10.1109/TVCG.2018.2843369

Jin, M. et al. (2020). An Enhanced Convolutional Neural Network in Side-Channel Attacks and Its Visualization. https://doi.org/10.48550/arXiv.2009.08898

Kahng, M., Andrews, P. Y., Kalro, A. and Chau, D. H. (2018). ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, Jan. 2018. https://doi.org/10.1109/TVCG.2017.2744718

Kahng, M., Thorat N., Chau D. H., Viegas, F. B. and Wattenberg M. (2019). GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):310–320, Jan. 2019. https://doi.org/10.1109/TVCG.2018.2864500

Karpathy, A. (2016). ConvNetJS MNIST demo. Available at: https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html Accessed 29 Oct 2022.

Kumar, D. J., Bhunia, P, Adhikary, S.D, Bej, B. (2022). Optimization of Effluents Using Artificial Neural Network and Support Vector Regression in Detergent Industrial Wastewater Treatment, Cleaner Chemical Engineering. Volume 3, 2022, 100039, ISSN 2772-7823. https://doi.org/10.1016/j.clce.2022.100039

Legaard, C. M., et al. (2021). Constructing neural network-based models for simulating dynamical systems. *ACM Computing Surveys, 55*(11), 1–34. https://arxiv.org/pdf/2111.01495.pdf, https://doi.org/10.1145/3567591. Accessed 20 Oct 2022.

Li, M., Zhao, Z., and Scheidegger, C. (2020). Visualizing Neural Networks with the Grand Tour. https://doi.org/10.23915/distill.00025

Liu, M. et al. (2017a). Towards Better Analysis of Deep Convolutional Neural Networks *in IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91-100, Jan. 2017. https://doi.org/10.1109/TVCG.2016.2598831

Liu, S., et al. (2017b). Visualizing High-Dimensional Data: Advances in the Past Decade. *IEEE Transactions on Visualization and Computer Graphics,* 23(3):1249–1268, Mar. 2017. https://doi.org/10.1109/TVCG.2016.2640960

López, O.A., López, A., Crossa, J. (2022). Fundamentals of Artificial Neural Networks and Deep Learning. *In: Multivariate Statistical Machine Learning Methods for Genomic Prediction. Springer, Cham.* https://doi.org/10.1007/978-3-030-89010-0_10

Meissler, N., Wohlan, A., Hochgeschwender, N. (2019). Using Visualization of Convolutional Neural Networks in Virtual Reality for Machine Learning Newcomers. *IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, San Diego, CA, USA, 2019, pp. 152-1526. https://doi.org/10.1109/AIVR46125.2019.00031

Mohamed, E., Sirlantzis, K., Howells, G. (2022). A review of visualisation-as-explanation techniques for convolutional neural networks and their evaluation. *Displays, Volume 73, 2022*, 102239, ISSN 0141-9382. https://doi.org/10.1016/j.displa.2022.102239

Norton, A. P. and Qi, Y. (2017). Adversarial-Playground: A visualization suite showing how adversarial examples fool deep learning. *In 2017 IEEE Symposium on Visualization for Cyber Security*

*(VizSec)*, pp. 1–4. IEEE, Phoenix, AZ, USA, Oct. 2017. https://doi.org/10.48550/arXiv.1708.00807

Olah, C. (2014). Neural Networks, Manifolds, and Topology. Available at: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/. Accessed 20 Nov 2022.

Ma, P. et al. (2022). Neural network in food analytics, Critical Reviews in Food Science and Nutrition. *Taylor and Francis Online*. https://doi.org/10.1080/10408398.2022.2139217

Psotka, J. (1995). Immersive training systems: Virtual reality and education and training. *Instructional Science,* vol. 23, no. 5-6, pp. 405–431, Nov. 1995

Queck, D., Wohlan, A., Schreiber, A. (2022). Neural Network Visualization in Virtual Reality: A Use Case Analysis and Implementation. In: Yamamoto, S., Mori, H. (eds) Human Interface and the Management of Information: Visual and Information Design. HCII 2022. *Lecture Notes in Computer Science*, vol 13305. Springer, Cham. https://doi.org/10.1007/978-3-031-06424-1_28

Sarker, I.H. (2021). Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective. *SN COMPUT. SCI. 2,* 154 (2021). https://doi.org/10.1007/s42979-021-00535-6

Sarvamangala, D. R., & Kulkarni, R. V. (2022). Convolutional neural networks in medical image understanding: a survey. *Evol. Intel., 15*, 1–22. https://doi.org/10.1007/s12065-020-00540-3

Schreiber A. and Bock, M. (2019). Visualization and exploration of deep learning networks in 3d and virtual reality. *In HCI International 2019* - Posters, C. Stephanidis, Ed. Cham: *Springer International Publishing*, 2019, pp. 206–211. https://doi.org/10.1007/978-3-030-23528-4_29

Shahroudnejad, A. (2021). A Survey on Understanding, Visualizations, and Explanation of Deep Neural Networks. https://arxiv.org/abs/2102.01792. Accessed 20 Dec 2022.

Smilkov, D., Carter, S., Sculley, D., Viegas, F. B. and Wattenberg, M. (2017). Direct- ´ Manipulation Visualization of Deep Networks. *arXiv*:1708.03788, Aug. 2017. https://arxiv.org/abs/1708.03788. Accessed 20 Oct 2022.

Tang, J, et al. (2019). Bridging Biological and Artificial Neural Networks with Emerging Neuromorphic Devices: Fundamentals, Progress, and Challenges. First published: 24 September 2019. https://doi.org/10.1002/adma.201902761

Taye, M. M. (2023). Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Computation, 2023*(*11*), *52*. https://doi.org/10.3390/computation11030052

Tulbure, A., Tulbure, A., & Dulf, E. (2022). A review on modern defect detection models using DCNNs – Deep convolutional neural networks. *Journal of Advanced Research, 35*(2022), 33–48. https://doi.org/10.1016/j.jare.2021.03.015

Wang, Z. J. et al. (2021). CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. in *IEEE Transactions on Visualization and Computer Graphics,* vol. 27, no. 2, pp. 1396-1406, Feb. 2021. https://doi.org/10.1109/TVCG.2020.3030418

Wieczorek, M., Siłka, J., & Woźniak, M. (2020). Neural network powered COVID-19 spread forecasting model, Chaos. *Solitons & Fractals, 140*(2020), 110203, ISSN 0960-0779. https://doi.org/10.1016/j.chaos.2020.110203

Xiang, L., Qian, D., & Jian-Qiao, S. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety, 172*(2018), 1–11. https://doi.org/10.1016/j.ress.2017.11.021

Xin, S. et al. (2021). ATNN: Adversarial Two-Tower Neural Network for New Item's Popularity Prediction in E-commerce. *IEEE 37th International Conference on Data Engineering (ICDE)*, Chania, Greece, 2021, pp. 2499-2510. https://doi.org/10.1109/ICDE51399.2021.00282.

Zhang, C., Okafuji, Y., and Wada, T. (2021). Evaluation of visualization performance of CNN models using driver model. 2021 IEEE/SICE International Symposium on System *Integration (SII),* Iwaki, Fukushima, Japan, 2021, pp. 739-744. https://doi.org/10.1109/IEEECONF49454.2021.9382776

## Authors and Affiliations

**Roland Kiraly**[1] · **Sandor Kiraly**[1] · **Martin Palotai**[1]

✉ Roland Kiraly
kiraly.roland@uni-eszterhazy.hu

✉ Sandor Kiraly
kiraly.sandor@uni-eszterhazy.hu

✉ Martin Palotai
palotaimartinm@gmail.com

1 Department of Information Technology, Eszterházy Károly Catholic University, Eger, Hungary