Computers, Materials & Continua
DOI:10.32604/cmc.2021.016803
Article

Tech Science Press

GUI-Based DL-Network Designer for KISTI's Supercomputer Users

Jaegwang Lee, Jongsuk R. Lee and Sunil Ahn[*]

Korea Institute Science and Technology Information, Daejeon, 34141, Korea
[*]Corresponding Author: Sunil Ahn. Email: siahn@kisti.re.kr
Received: 12 January 2021; Accepted: 07 March 2021

Abstract: With the increase in research on AI (Artificial Intelligence), the importance of DL (Deep Learning) in various fields, such as materials, biotechnology, genomes, and new drugs, is increasing significantly, thereby increasing the number of deep-learning framework users. However, to design a deep neural network, a considerable understanding of the framework is required. To solve this problem, a GUI (Graphical User Interface)-based DNN (Deep Neural Network) design tool is being actively researched and developed. The GUI-based DNN design tool can design DNNs quickly and easily. However, the existing GUI-based DNN design tool has certain limitations such as poor usability, framework dependency, and difficulty encountered in changing GUI components. In this study, a deep learning algorithm that solves the problem of poor usability was developed using a template to increase the accessibility for users. Moreover, the proposed tool was developed to save and share only the necessary parts for quick operation. To solve the framework dependency, we applied ONNX (Open Neural Network Exchange), which is an exchange standard for neural networks, and configured it such that DNNs designed with the existing deep-learning framework can be imported. Finally, to address the difficulty encountered in changing GUI components, we defined and developed the JSON format to quickly respond to version updates. The developed DL neural network designer was validated by running it with KISTI's supercomputer-based AI Studio.

Keywords: Deep neural network design; ONNX; GUI design tool; deep learning

1 Introduction

DL (Deep Learning) is a method of constructing a network that is similar to the structure of the human brain. Machine learning has limited ability to process natural data in their original format [1]. Deep learning is composed of multiple processing layers, and the computational model can learn data representations with multiple levels of abstraction. This method has improved the technology in various fields such as speech recognition, image recognition, and dielectrics [2]. Recently, research on DL research has increased, and various frameworks such as TensorFlow, Keras, PyTorch, and Caffe have been proposed. The DL framework provides a variety of pre-trained DL models and algorithms with pre-defined libraries [3]. This helps users to quickly and easily use repetitive tasksrmrj r so that users can focus on developing important algorithms [4]. However, the DL framework requires a high degree of understanding using libraries and various pre-trained DL models and algorithms. In addition, the code-based DNN design requires a high degree of understanding by users owing to typos and errors [5]. Recently, DL has been used in various fields such as materials, biotechnology, dielectrics, and new drugs, and research on methods that enable users to easily and conveniently design DNNs according to diversification is being actively conducted [6,7].

The existing GUI (Graphical User Interface)-based DNN (Deep Neural Network) design tool uses the drag and drop method and selection method to design DNNs. However, traditional GUI-based DNN design tools have disadvantages including poor usability, framework dependency, and difficulty encountered in changing GUI components [8]. Existing GUI-based DNN design tools have the disadvantage of requiring the same level of understanding as that for the framework. Furthermore, it cannot be reused as the user must create the neural network from scratch each time [9]. Next, most GUI-based DNN design tools are designed based on specific frameworks and algorithms, and they framework dependencies. It is difficult to use the pre-trained deep-learning model when sharing algorithms; it encounters difficulty because it uses a different framework for each user. In other words, to apply other user's deep-learning models and algorithms, a certain high level of understanding of two or more frameworks is required [10]. Finally, existing GUI-based DNN design tools encounter difficulty in changing GUI components. The GUI-based DNN design tool has the disadvantages of high time consumption and incurred cost because it requires considerable work such as design change of the tool and node modification to change GUI components when updating the version [11].

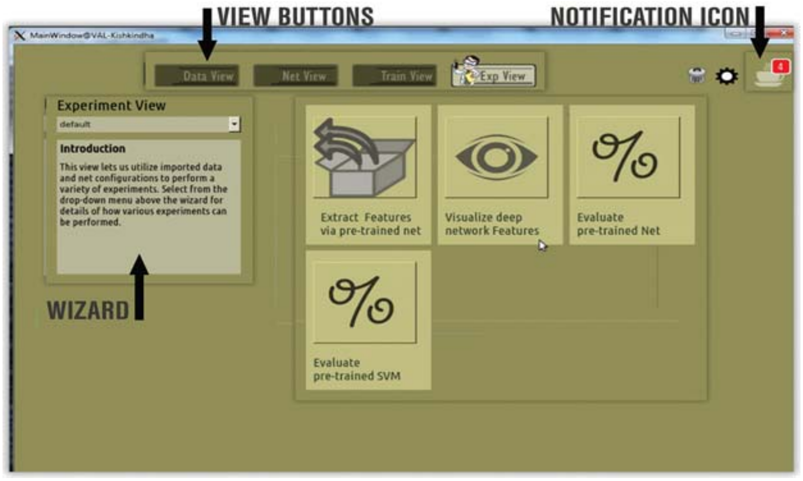
A DL neural network designer was developed to solve the problems of existing GUI-based DNN design tools including the poor usability, framework dependency, and difficulty in changing GUI components. We constructed a template of the known deep learning algorithm and used the drag and drop method to improve accessibility for users. DL neural network designer also allows the designed DNNs to be shared with other users. We implemented a function that can be used as a template by storing only necessary parts of the designed DNN. The reusability problem was solved by implementing this function. To solve the framework-dependency problem, ONNX (Open Neural Network Exchange) [12], which is a neural network exchange standard, was applied to the DL neural network designer. To eliminate the difficulty encountered in changing GUI components, each GUI component was modularized, so that it can respond quickly to the version change, and the JSON format for each module was defined. The ONNX operator and the PyTorch library parser were developed to convert the model file to JSON format.

Herein, Chapter 2 analyzes the existing GUI-based DNN design tool, and Chapter 3 defines the problem. Chapter 4 designs a system to solve the defined problems, and Chapter 5 examines the part that developed the designed DL neural network designer. Finally, Chapter 6 presents the results.

2 Related Research

2.1 Expresso

Expresso is a GUI tool written in Python to provide a user-friendly interface for designing, training, and developing deep learning frameworks. It is built on Caffe, which is a widely used framework for developing CNNs (Convolutional Neural network). Expresso provides a convenient wizard-like graphical interface that guides the user through various processes. The following tasks are executed in Expresso: importing data, building and training deep networks, performing various experiments, and analyzing and visualizing experimental results. The multithreaded nature of Expresso allows for the concurrency and notification of events related to the scenarios described earlier. Expresso is developed based on a specific framework and a specific algorithm, and it suffers from dependency problems [13]. Fig. 1 shows the main screen of Expresso.



Introduction

This view lets us utilize imported data and net configurations to perform a variety of experiments. Select from the drop-down menu above the wizard for details of how various experiments can be performed.

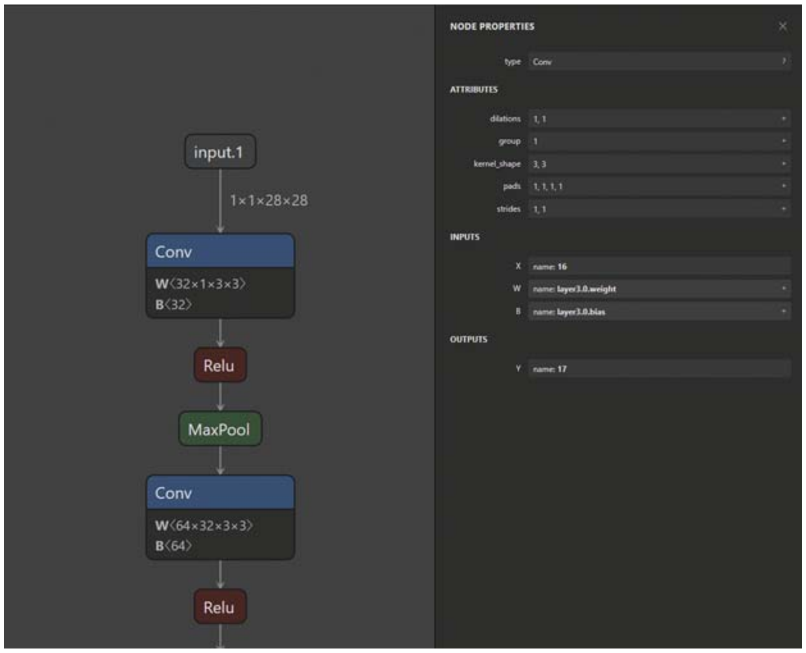
Figure 1: Main screen of Expresso [13]

2.2 *Netron*

Netron is a GUI-based viewer for deep-learning and machine-execution models. The program is developed with Electron and has the advantages of the availability of a PC installation version (macOS, Linux, and Windows support) and web-based execution. Netron supports 17 official frameworks (including Keras, Core ML, MXNet, and TensorFlow Lite) and 14 informal frameworks (including TensorFlow, PyTorch, TorchScript, and Chainer). It also shows that each block uses not only input and output forms but also some activation functions. However, in the case of Netron, the model cannot be modified, which is a drawback [14]. Fig. 2 shows the interface of Netron.

2.3 *Barista*

Barista is a Caffe-based GUI tool, which is an open software deep-learning framework. Caffe is one of the most popular frameworks used for DNN training. However, using this tool to edit a prototype text file to specify the net architecture and hyperparameters is tedious and prone to errors. Barista is a graph-based net topology editor that provides a complete graphical user interface. It provides an end-to-end educational facility for DNNs, allowing researchers to focus on solving problems without having to write codes and edit text files, or edit them [6]. Fig. 3 shows the interface of the Barista.



NODE PROPERTIES

type Conv

ATTRIBUTES

dilations 1, 1
group 1
kernel_shape 3, 3
pads 1, 1, 1, 1
strides 1, 1

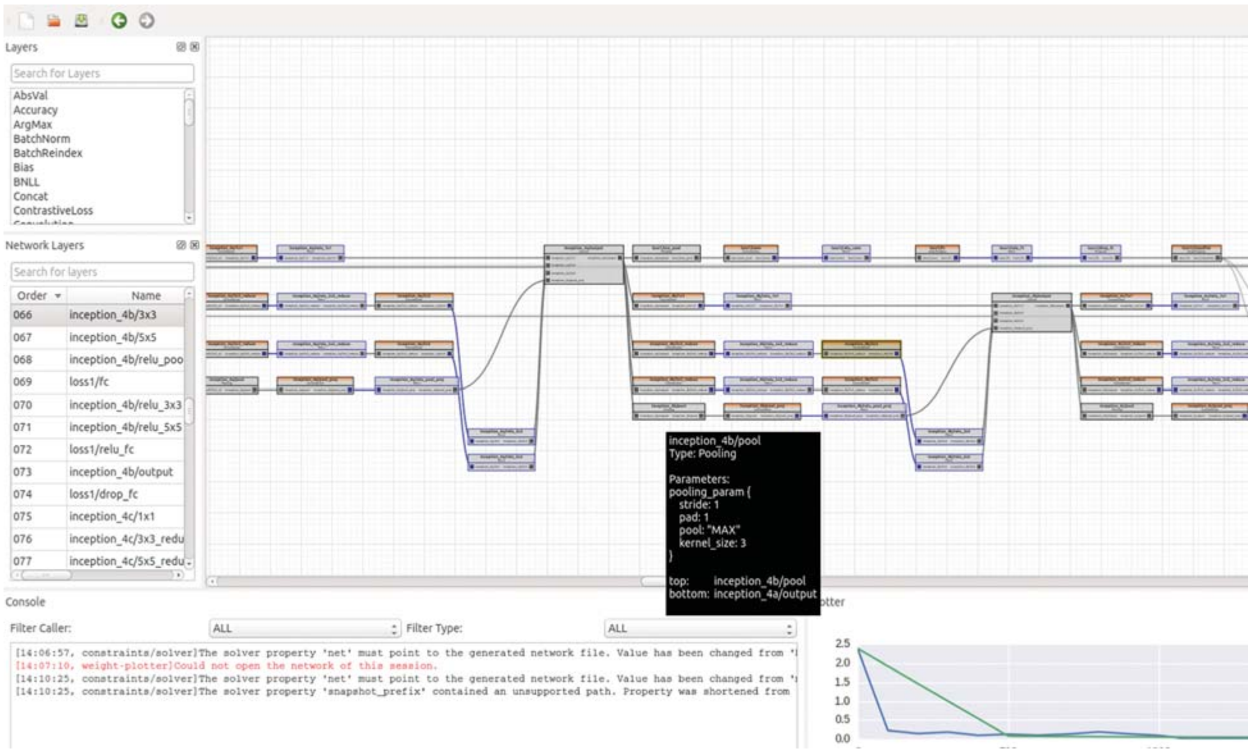
INPUTS

X name: 16
W name: layer3.0.weight
B name: layer3.0.bias

OUTPUTS

Y name: 17

Figure 2: Deep-learning model loading using Netron [14]



Order	Name
066	inception_4b/3x3
067	inception_4b/5x5
068	inception_4b/relu_poo
069	loss1/fc
070	inception_4b/relu_3x3
071	inception_4b/relu_5x5
072	loss1/relu_fc
073	inception_4b/output
074	loss1/drop_fc
075	inception_4c/1x1
076	inception_4c/3x3_redu
077	inception_4c/5x5_redu

```
inception_4b/pool
Type: Pooling

Parameters:
pooling_param {
  stride: 1
  pad: 1
  pool: "MAX"
  kernel_size: 3
}

top:      inception_4b/pool
bottom:  inception_4a/output
```

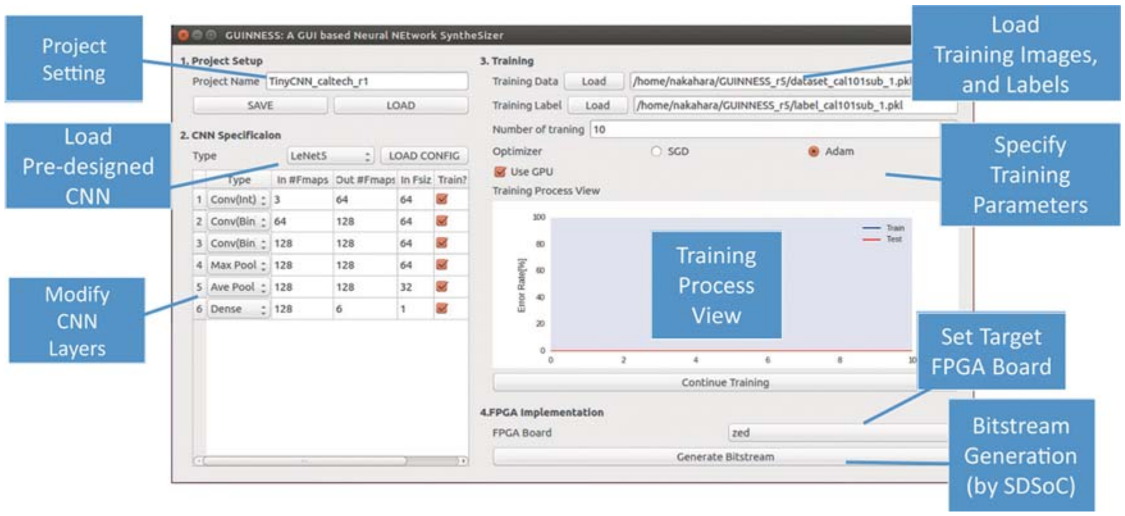
Filter Caller: ALL Filter Type: ALL

```
[14:06:57, constraints/solver]The solver property 'net' must point to the generated network file. Value has been changed from '
[14:07:10, weight-plotter]Could not open the network of this session.
[14:10:25, constraints/solver]The solver property 'net' must point to the generated network file. Value has been changed from '
[14:10:25, constraints/solver]The solver property 'snapshot_prefix' contained an unsupported path. Property was shortened from
```

Figure 3: Interface of the Barista [6]

2.4 GUINNESS

GUINNESS (GUI-based binarized neural network synthesizer) is an open-source tool for binarized DNNs for GUI-based FPGA implementations that include both training for GPUs and FPGA inference. GUINNESS has implemented a VGG-11 based on Digilent Inc. Zedboard's. Therefore, the software designer does not need to create a script for designing the structure of the neural network and the training operation. Only the hyperparameter values are specified and used. At the end of the training, GUINNESS will automatically generate C++ code and use the Xilinx SDSoC system design tool flow to synthesize the bitstream. Therefore, it is suitable for software programmers who are not familiar with the FPGA design. GUINNESS relies the problem of relying on specific algorithms for GPU education and FPGA inference, which is a problem [15]. Fig. 4 shows the interface of the GUINNESS.



Project Name TinyCNN_caltech_r1

Type LeNet5

	Type	In #Fmaps	Out #Fmaps	In Fsiz	Train?
1	Conv(Int)	3	64	64	✓
2	Conv(Bin	64	128	64	✓
3	Conv(Bin	128	128	64	✓
4	Max Pool	128	128	64	✓
5	Ave Pool	128	128	32	✓
6	Dense	128	6	1	✓

Training Data /home/nakahara/GUINNESS_r5/dataset_cal101sub_1.pkl

Training Label /home/nakahara/GUINNESS_r5/label_cal101sub_1.pkl

Number of traning 10

Optimizer SGD [✓ Adam]

✓ Use GPU

FPGA Board zed

Figure 4: Interface of the GUINNESS [15]

3 Problem Definition

The existing DNN design method can be divided into code-based and GUI-based DNN designs. In the former case, usability largely depends on the user's understanding of the framework. Even if they have a high degree of understanding of the framework, errors such as incorrect variable usage and typos occur when developing the code. Various GUI-based DNN design tools have been developed to solve these problems. However, existing GUI-based DNN design tools have the limitations of poor usability, framework dependency, and difficulty encountered in changing GUI components.

3.1 Poor Usability

The existing GUI-based DNN design tool has been developed to enhance the user's convenience and productivity. However, the existing GUI-based DNN design tool requires a higher degree of understanding of the DL algorithm, which is a problem for beginners using the tool. In the conventional code-based method, the DNN can be copied and used. However, the GUI-based DNN design tool has the drawback that the code-based neural network needs to be redesigned into a GUI-based neural network. Redesigning consumes a considerable amount of time as the corresponding nodes need to be arranged and connected sequentially. The existing GUI-based DNN design tool has been developed to enhance e convenience and productivity; however, reusability is an issue that needs to be addressed. As users become more familiar with the DL algorithm, reusing user-created DNNs or modifying parts of DNNs is a requirement. The existing GUI-based DNN design tool has the disadvantage that it cannot be reused or used with other external DNNs. These problems were solved by using a Template, node module, and sharing function with a DL neural network designer.

3.2 Framework Dependency

Several frameworks have been developed and used depending on factors such as the research method and discipline. The existing GUI-based DNN design tool has framework dependencies. It is difficult to use a pre-trained deep-learning model when using other frameworks for sharing algorithms. In other words, to use another user's DL model or DL neural network, a high level of understanding of frameworks and DL algorithms is required. In addition, the existing GUI-based DNN design tool has the disadvantage that it is not possible to express or modify the DL neural network of other users. It is also not possible to reuse or use only a few parts of a DL neural network; thus, a significant amount of time will be spent on creating DL neural networks and performing correction operations. The DL neural network designer applies ONNX, which is a DNN exchange standard, to solve the framework dependency. It can be converted to the JSON format defined using the ONNX operator, which is a PyTorch library-based parser. We defined the GUI JSON format of the DL neural network designer and designed the DNN such that its necessary parts could be modularized and saved to be used as a template.

3.3 Difficulty in Changing GUI Components

Existing GUI-based DNN design tools are based on specific frameworks and algorithms, which renders GUI-component modification a difficult task. Version update, framework addition, algorithm addition, and GUI-component change require tool design and structural changes; high-speed response is not possible, which is a disadvantage. In addition, when applying ONNX to support various frameworks, the GUI-based DNN design tool should be designed to support quick ONNX version updates. The DL neural network designer is developed to respond quickly to version updates through functional modularization.

4 DL Neural Network Designer Design

Fig. 5 illustrates the system flow of the DL neural network designer. The user scenario is as follows. When the user imports the model created by the code-based framework, a ".onnx" file is generated through the ONNX converter. The generated ".onnx" file is optimized for the JSON format used by the DL neural network designer, and then, the DNN is generated in the designer. The generated DNN can be modified, and recreated, and then exported. At the time of export, a file is created via a code converter, and the user can check the training progress and can download or share the file using the linked AI Studio.

The hardware of the proposed DL neural network designer is based on KISTI's supercomputer. The DL neural network designer is developed as a tool used in the AI Studio platform to solve the problems of poor usability, framework dependencies, and the difficulty encountered in changing GUI components. It can easily and quickly respond to the aforementioned problems through functional modularity. Fig. 6 shows its configuration.

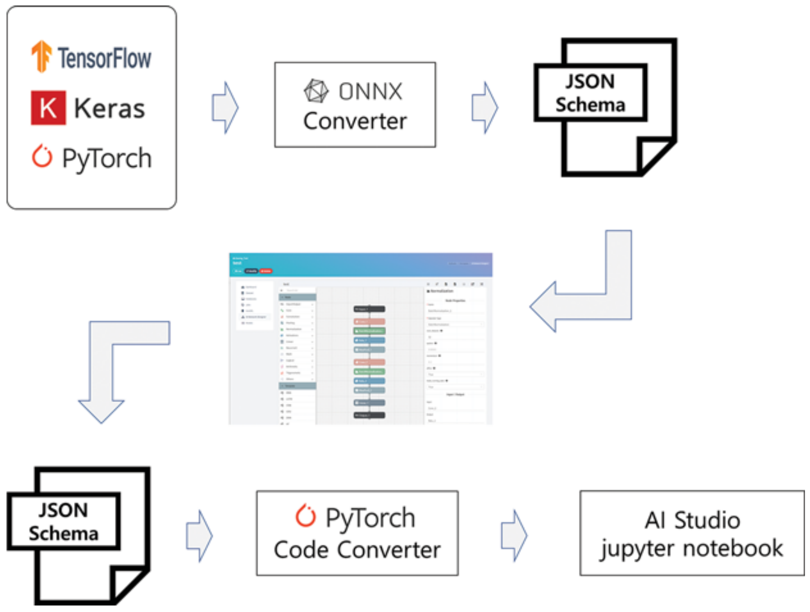


Figure 5: System flow of the DL neural network designer

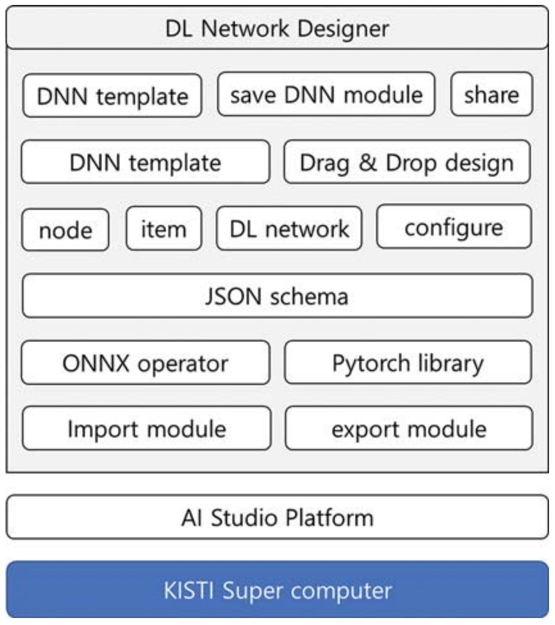


Figure 6: Configuration of the DL neural network designer

4.1 Usability Troubleshooting

We propose a template method for a faster DNN design. In addition, we propose the sharing and saving of the parts of DNNs to improve reusability.

4.1.1 Template

The template is a pre-designed DNN. It was proposed to reduce the time spent by the users of GUI-based DNN design tools to design deep neural networks and to improve the understanding of DL algorithms. It is developed by the drag and drop method; even beginners can quickly and easily configure a DNN.

The template node is composed of a CNN, a DNN, an RNN (Recurrent Neural Network), an LSTM (Long Short-Term Memory), a GRU (Gated Recurrent Units), an AE (Auto Encoder), a DBN (Deep Belief Network), and a DBN (Deep Belief Network). The template is of two types: open type and close types. The open-type template has a form in which several items are unfolded, whereas and the close-type template only displays one item. In the open-type template, parameters, such as num_layer, input_channel, and output_channel, can be entered, indicating whether repetitions are to be several times based on the basic layer. It is also possible to change open item modifications to new DNNs and design complex DNNs. Because the close-type template is provided by the PyTorch library and ONNX operator, it was designed to be used for one item. The use of the template method is faster and easier than that of to use when compared to traditional GUI-based DNN design tools, which help in enhancing the convenience and productivity of the user. Tab. 1 lists the DL algorithm template properties.

Table 1: DL algorithm template properties

Template node type	DL algorithm	Correction and rewrite	Parameter modification
Open type	CNN	O	O
	DNN	O	O
	AE	O	O
	DN	O	O
Close type	RNN	X	O
	LSTM	X	O
	GRU	X	O
	DBN	X	O

4.1.2 DNN Save Module

The DNN save module was proposed to enhance reusability and convenience. It is possible to save the entire or a part of the designed DNN. The saved DNN can be used as a template. The designed DNN is saved and managed in ".json" via the DNN save module, and it can be deleted when the user does not need it. Fig. 7 presents a flow chart of the DNN that saves the module.

4.1.3 Share Module

The Share module was proposed to improve the reusability of DNNs through sharing between users. A Share module that shares a user-designed DNN creates a JSON file and saves it in a DB File. DNN files stored in the DB File are listed on the Share page for the user to use the shared DNN.

There are two methods of using a shared DNN:

1. Create a new project by pressing the Import button of the shared DNN

2. Click the Import button of the shared deep neural network to add it to the My Favorite Network of the user's existing project.

Fig. 8 shows the flow chart of the sharing and reuse neural networks.

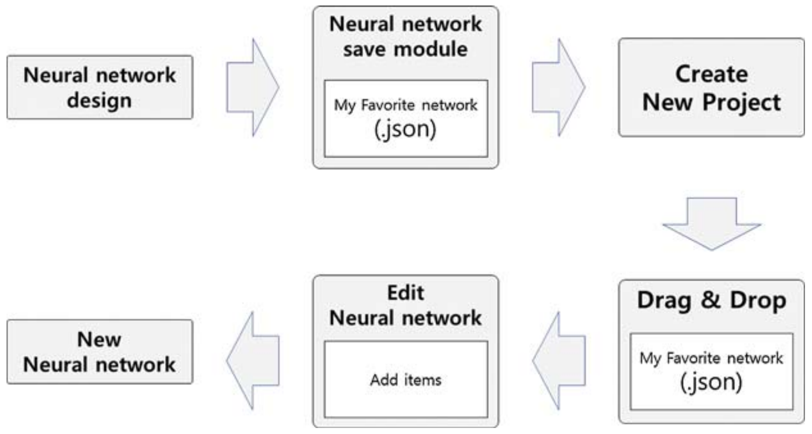


Figure 7: Flow chart of the DNN that saves the module

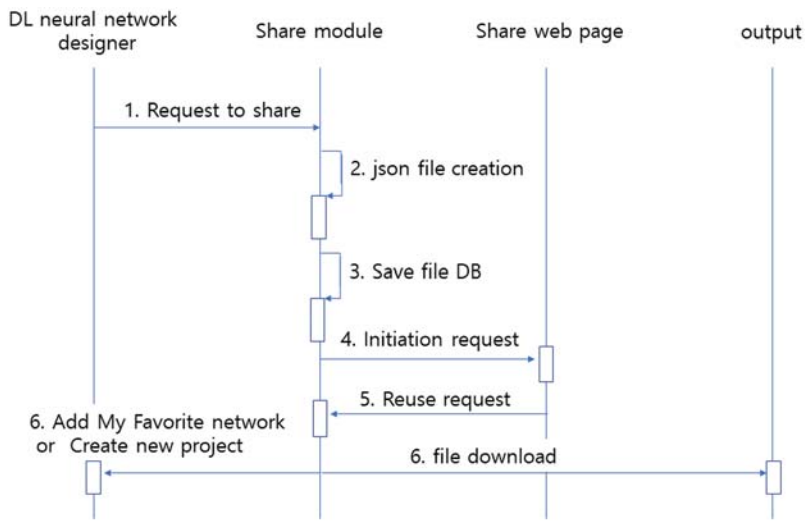


Figure 8: Flow chart of the sharing and reuse neural networks

4.2 Suggesting of a Method of Solving Framework Dependency

The framework dependency is a pre-trained deep learning model, which requires a significant amount of time and a high level of understanding of several frameworks for reusing the algorithm. To solve this problem, the DL neural network designer has applied the ONNX standard to support various frameworks. ONNX defines operators and data types and supports interoperability between different frameworks. This solves the issue of learning multiple frameworks and makes it easy to apply the DL algorithm. This section details the design strategies for resolving framework dependencies.

The Import module has a framework discrimination function, an ONNX conversion function, and a GUI-based JSON conversion function. The GUI JSON format is divided into a Node, Item, Template, Configure, and GUI DNN. The ONNX converter uses the extension to determine the framework and converts it to ".onnx" using the ONNX API that matches the framework. The converted ".onnx" file is forwarded to the GUI JSON format defined using the JSON parser and is converted to ".json" as shown in the Designer View. The GUI JSON format is defined by the required value of the parameter, option value, Node (which is a group of Item), position of Item, and Link (which is a connection between Items), based on the ONNX operator and the library of PyTorch. Fig. 9 shows the concept of the Import module.

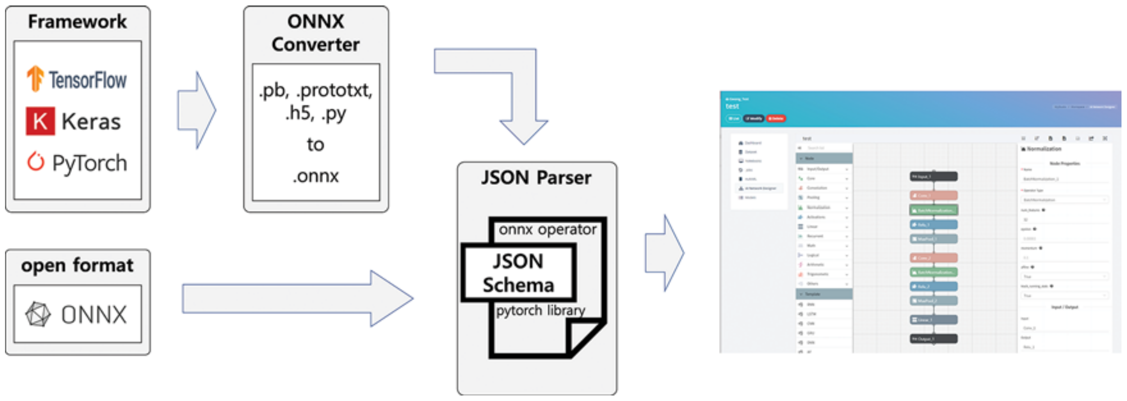


Figure 9: Process of an import module

The Export module is responsible for converting DNNs designed using a code converter to PyTorch-class files. The code converter relocates ".json" files using the JSON parser and subsequently generates ".py" files, including the PyTorch-class. The generated ".py" file then generates a ".ipynb" via a file converter based on the user's request (download, with training through AI Studio). Below, Fig. 10 presents the concept of an Export module.

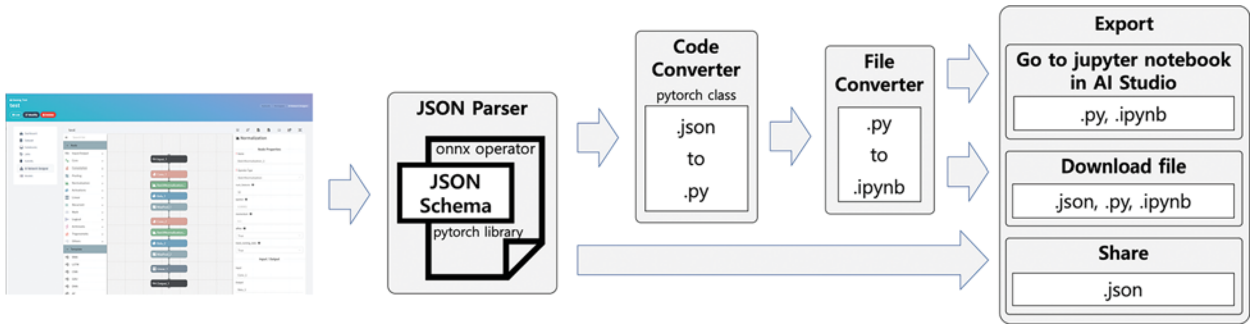


Figure 10: Concept of export module

4.3 Suggestion of a Method of Solving Difficulty Encountered in Changing GUI Components

The DL neural network designer was designed to solve the difficult problem concerning the changing of GUI components by partitioning the GUI JSON format and modifying it easily and quickly via modularization. These designers can be divided into three categories: Component, Designer, and Configure. The Component consists of the Node, Item, and Template. Nodes are divided into 13 groups including Core, Convolution, and Pooling. The Node's JSON format is configured to be modified only when a group is added. The Configuration contains the ID of the Item that belongs to the group. The Item is composed of Convolution, ReLU, MaxPool, etc. as the basic components of DL. The Item sets the prerequisites, options, and default values of parameters based on the ONNX operator and PyTorch library as the elements that constitute the actual DNN. The Item contains information such as the Item label, parameter label, data type, required type, default, and description. Finally, the Template is registered in the Node and Configuration and contains some Item information, connection information between Items, and Item position information. Below, Fig. 11 shows the linkage information between modules of the modularized DL neural network designer.

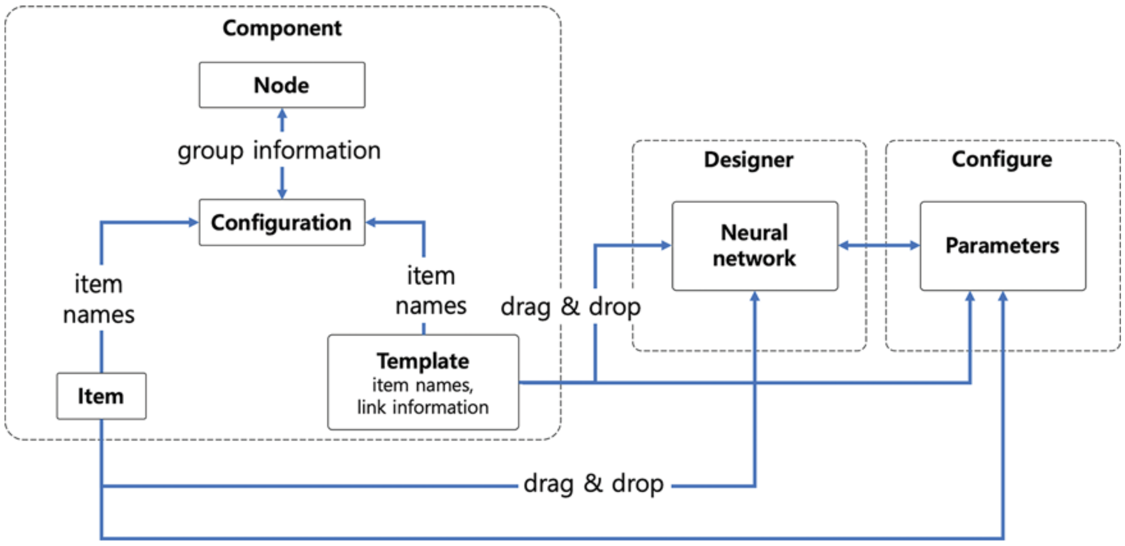
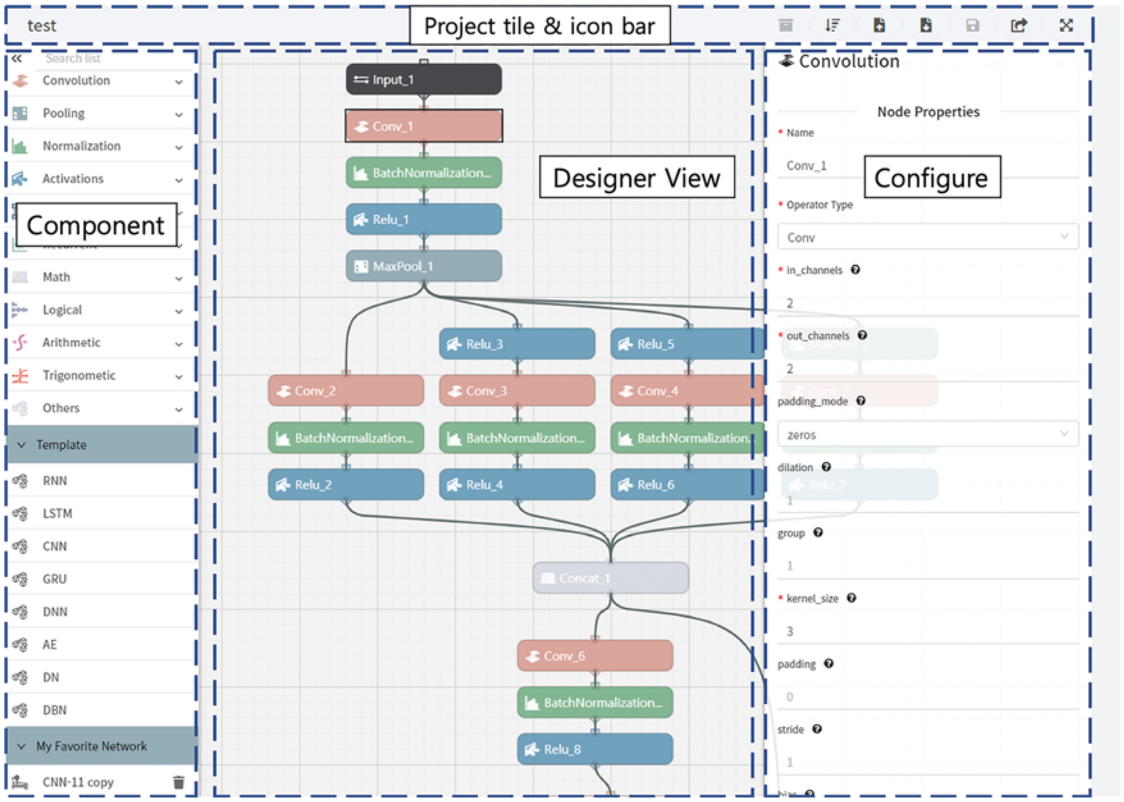


Figure 11: Linkage information between modules of the modularized DL neural network designer

5 Construction of DL Neural Network Designer

Fig. 12 shows the developed DL neural network designer screen. It consists of the Project name, Component, Designer View, Configure, and Icon bar, and it possesses a user-friendly structure. The Component consists of the Item, Template, and My Favorite Network, where the Configure was developed such that parameters can be easily inputted. Fullscreen, Save, Share, Import, Export, and Sort icons are arranged in the Icon bar. The Designer View is where a DNN can be designed via a "drag and drop" process of the components.



Name: Conv_1
Operator Type: Conv
in_channels: 2
out_channels: 2
padding_mode: zeros
dilation: 1
group: 1
kernel_size: 3
padding: 0
stride: 1

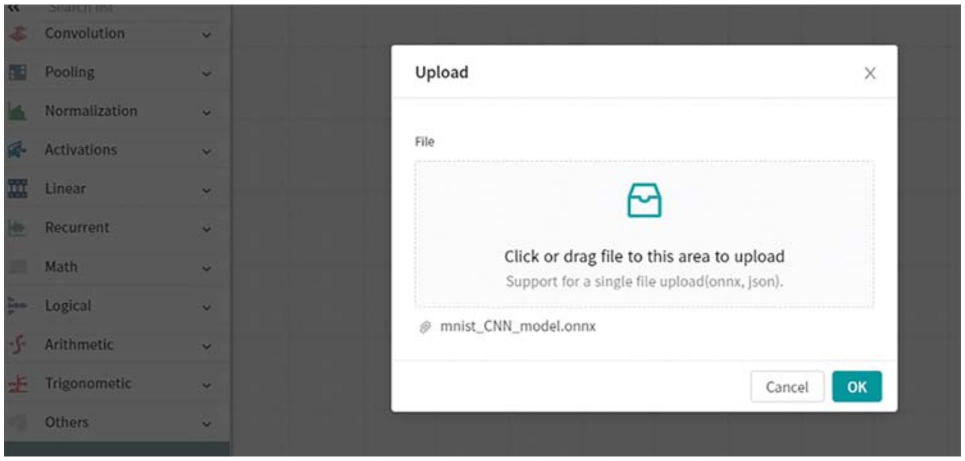
Figure 12: DL neural network designer screen

5.1 Import/Export Module

5.1.1 Import Module Development

When the user uploads a model or deep neural network file, the Import module determines the framework, as per the extension (TensorFlow (.pb), Keras (.h5), and PyTorch (.pth)). Subsequently, this file is converted to the ".onnx" format via the ONNX API (tf2onnx, keras2onnx, pytorch2onnx). Below, Fig. 13 shows the screen corresponding to when a model file is to be uploaded in the GUI.

The converted ".onnx" file contains data that are not required to represent the deep neural network in the GUI with the DL neural network designer. Unnecessary data result in the consumption of unnecessary time when calling the model. The JSON parser optimizes this process by removing such unnecessary data and rearranging the required data into the defined GUI JSON format to reduce the unnecessary time consumption. Fig. 14 presents the JSON parser optimization source code, and Fig. 15 shows the JSON format capture screens before and after optimization.



Click or drag file to this area to upload
Support for a single file upload(onnx, json).
mnist_CNN_model.onnx

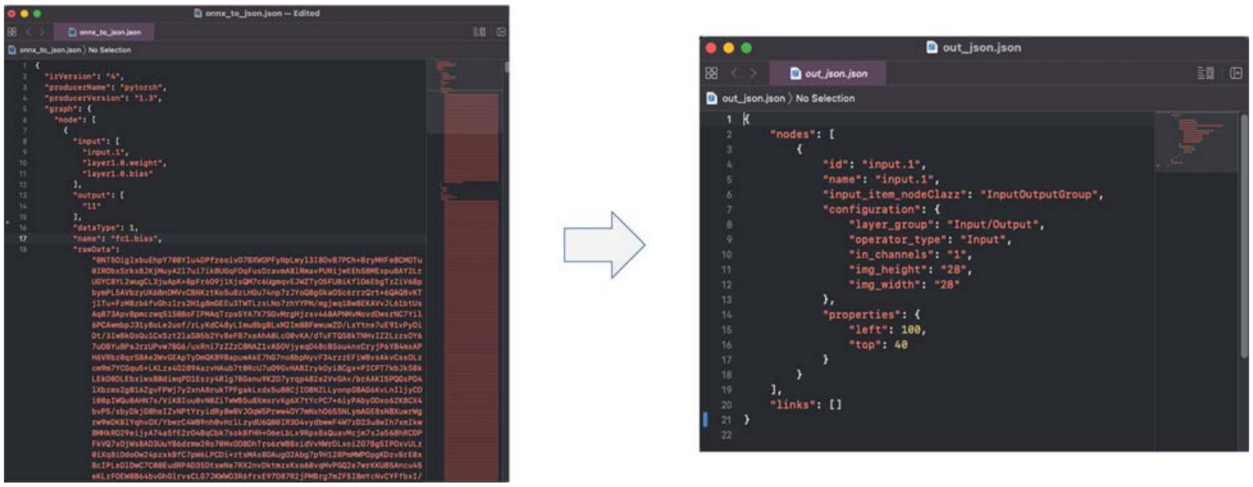
Figure 13: Screen corresponding to when a model file is to be uploaded in the GUI

```
def op_conv2d(node_data, file_data, node_xy, node_id, node_name, node_size, i):
…
    try:
        for j in range(node_size):
            if node_attribute[j]["name"] == "dilations":
                dilations = node_attribute[j]["ints"]
                conv2d_node_configuration["dilation"] = dilations

            elif node_attribute[j]["name"] == "strides":
                strides = node_attribute[j]["ints"]
                conv2d_node_configuration["stride"] = strides

            elif node_attribute[j]["name"] == "kernel_shape":
                kernel_shape = node_attribute[j]["ints"]
                conv2d_node_configuration["kernel_size"] = kernel_shape

            elif node_attribute[j]["name"] == "pads" or "auto_pad":
                if node_attribute[j]["name"] == "auto_pad":
                    pads = ["1", "1"]
                    conv2d_node_configuration["padding"] = pads
                elif node_attribute[j]["name"] == "pads":
                    pads = node_attribute[j]["ints"]
                    conv2d_node_configuration["padding"] = pads

            elif node_attribute[j]["name"] == "group":
                group = node_attribute[j]["i"]
                conv2d_node_configuration["groups"] = group

…
```

Figure 14: Optimization using JSON parser

5.1.2 Export Module Development

The Export module is responsible for code conversion and AI Studio interlocking. Fig. 16 shows the screen visible at the time of export. It can be downloaded in the GUI JSON format (.json) or as the PyTorch class (.py) file. Training advancement using KISTI's supercomputer export is possible via the Jupyter notebook linked with AI Studio. A code converter can be used to convert a DNN to code. The code converter with a built-in JSON parser can convert ".json" files in the GUI JSON format to ".py" files. The ".py" file contains the PyTorch class code.


onnx_to_json.json — Edited

```
{
  "irVersion": "4",
  "producerName": "pytorch",
  "producerVersion": "1.3",
  "graph": {
    "node": [
      {
        "input": [
          "input.1",
          "layer1.0.weight",
          "layer1.0.bias"
        ],
        "output": [
          "11"
        ],
        "dataType": 1,
        "name": "fc1.bias",
        "rawData":
          [illegible]
```

out_json.json

```
{
    "nodes": [
        {
            "id": "input.1",
            "name": "input.1",
            "input_item_nodeClazz": "InputOutputGroup",
            "configuration": {
                "layer_group": "Input/Output",
                "operator_type": "Input",
                "in_channels": "1",
                "img_height": "28",
                "img_width": "28"
            },
            "properties": {
                "left": 100,
                "top": 40
            }
        }
    ],
    "links": []
}
```

Figure 15: JSON format capture screens before (left) and after (right) optimization

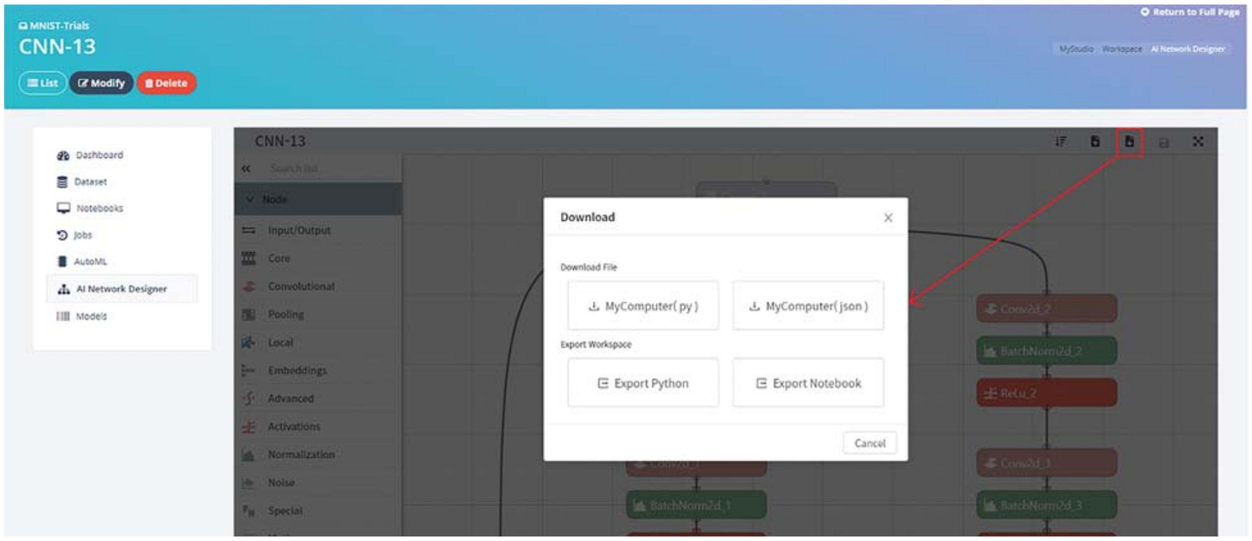
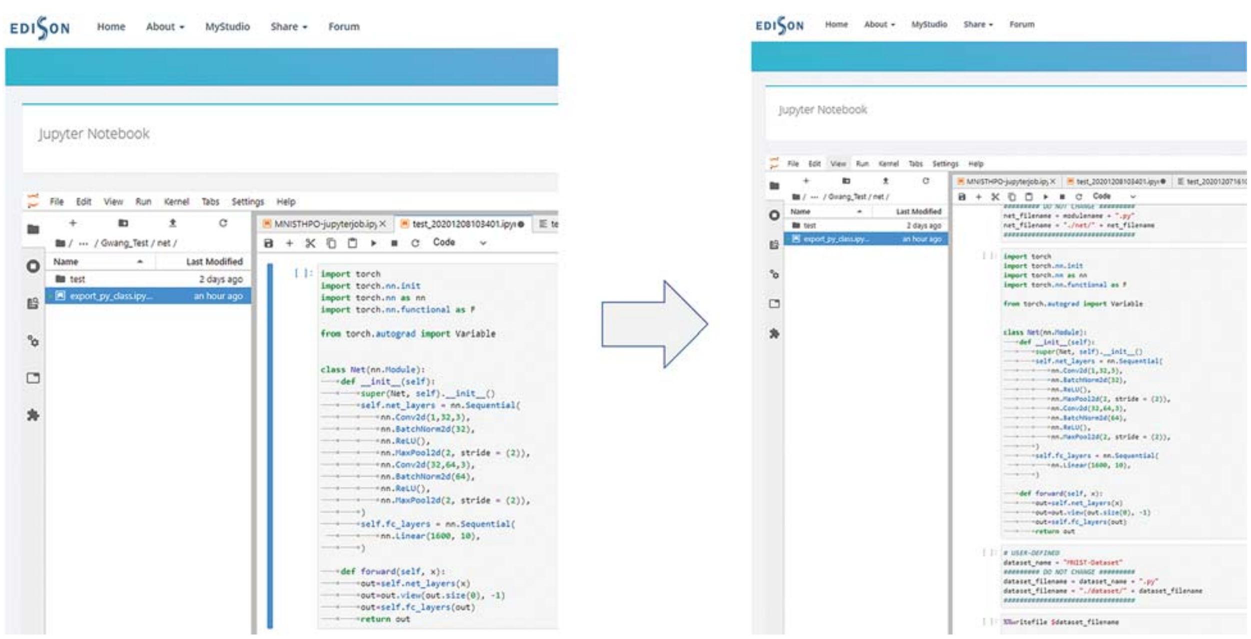


Figure 16: DL neural network designer export screen

To use the API of AI Studio, a file format compatible with Jupyter notebook is needed, so that the ".py" file format can be converted to ".ipynb". In addition, for the convenience of the user, the library Import function is needed such that the API of AI Studio can be loaded automatically. These functions are supported by the file converter. Fig. 17 shows a ".ipynb" file that combines the ".ipynb" file created by the code converter and the API converted by the file converter.



```
import torch
import torch.nn.init
import torch.nn as nn
import torch.nn.functional as F

from torch.autograd import Variable


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.net_layers = nn.Sequential(
            nn.Conv2d(1,32,3),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = (2)),
            nn.Conv2d(32,64,3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = (2)),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(1600, 10),
        )

    def forward(self, x):
        out=self.net_layers(x)
        out=out.view(out.size(0), -1)
        out=self.fc_layers(out)
        return out
```

```
######### DO NOT CHANGE #########
net_filename = modulename + ".py"
net_filename = "./net/" + net_filename
##################################
```

```
import torch
import torch.nn.init
import torch.nn as nn
import torch.nn.functional as F

from torch.autograd import Variable


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.net_layers = nn.Sequential(
            nn.Conv2d(1,32,3),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = (2)),
            nn.Conv2d(32,64,3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = (2)),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(1600, 10),
        )

    def forward(self, x):
        out=self.net_layers(x)
        out=out.view(out.size(0), -1)
        out=self.fc_layers(out)
        return out
```

```
# USER-DEFINED
dataset_name = "MNIST-Dataset"
######### DO NOT CHANGE #########
dataset_filename = dataset_name + ".py"
dataset_filename = "./dataset/" + dataset_filename
##################################
```

```
%%writefile $dataset_filename
```

Figure 17: ".py" file converted to code and ".ipynb" file including API

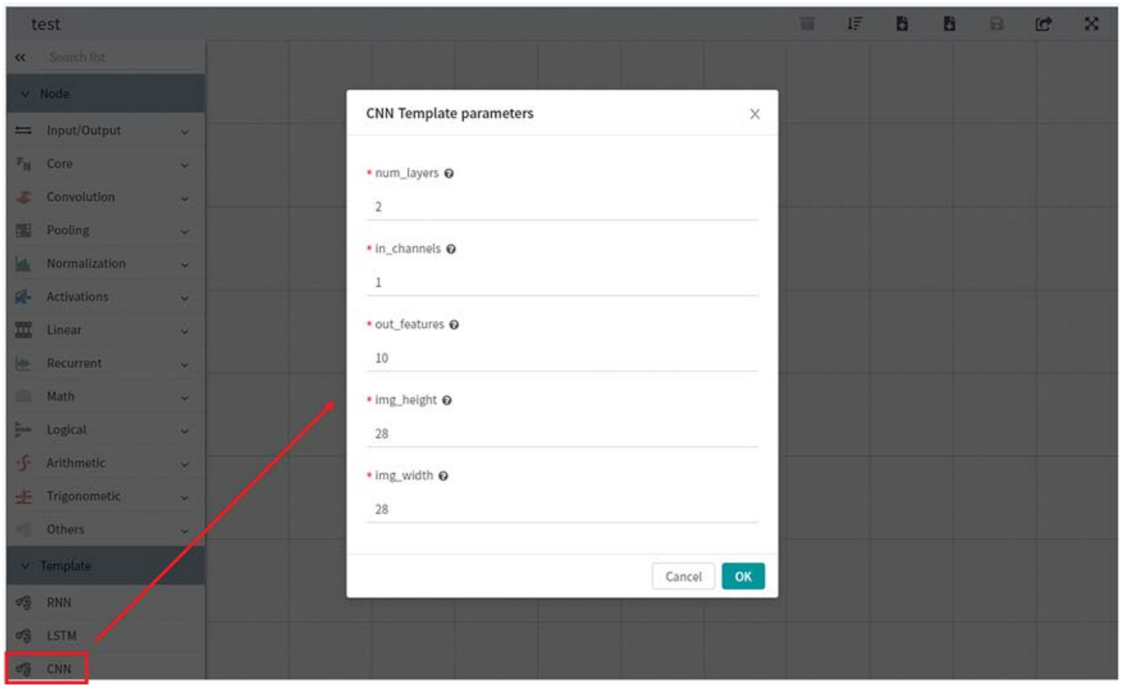
5.2 *Component and Configure*

5.2.1 *JSON Format Definition*

In the DL neural network designer, the Item can be viewed as the basic component of the graph, the Configure as the parameter set of the basic component, the Node as the group that classifies the Item, and Configuration as the seam between the Node and Item. The Nodes are categorized for user convenience. Because color and image information is contained such that the user can easily distinguish between the Items, readability is improved. Herein, an Item was created via the ONNX operator parser, and the parameters of the PyTorch components were compared and rearranged. The Item is of 94 Item types. The parameters of the ONNX operator parser and PyTorch components were compared, and the sorting module was implemented such that it could respond quickly to version updates. The Configure shows the parameter data regarding the "form" of the Item. The Configuration matches Items to Nodes. Thus, it indicates the Node that each Item belongs to.

5.2.2 *Template Development*

The DL neural network designer supports the CNN, DNN, RNN, LSTM, GRU, AE, DN, and DBN. The open-type template consists of a Net layer and a fully connected layer. One Net layer of the CNN contains the structure of Convolution, Batch normalization, ReLU, and MaxPool. This is a method wherein the basic layers that constitute two or more Net layers are repeated, and it could input and calculate the number of layers, the number of input channels, number of outputs, and width and height of the images that constitute the network using the Template. In the DNN, one Net layer is linear and sigmoid, and the fully connected layer is linear. In the AE, one Net layer is linear and leaky ReLU, and the fully connected layer is linear and sigmoid. The DN has a structure that forms the following pair: Convolution, ReLU, and MaxPool, and ConvTranspose, ReLU, and MaxUnpool. The close-type RNNs, LSTM, GRU, and DBN are configured with one Item for easy use. Fig. 18 shows a screen corresponding to the designing of a network using a CNN template, and Fig. 19 shows an open-type template and a close-type template.



CNN Template parameters

* num_layers: 2
* in_channels: 1
* out_features: 10
* img_height: 28
* img_width: 28

Figure 18: Screen corresponding to the designing of a network using a CNN template

5.2.3 My Favorite Network and Share Development

My Favorite Network and Share are features proposed to improve reusability, enabling users to configure frequently used networks and use them as templates. My Favorite Network can convert the part of the network needed by the user into a module. Alternatively, a DNN that is not in the template can be designed and reused. Fig. 20 shows a modular custom model.

Share is a function that shares a deep neural network designed by the DL neural network designer. The Neural Network Viewer can be used to view the deep neural network shared by the user and copy it to the user's new project or My Favorite Network. Fig. 21 shows a list of shared deep neural networks.

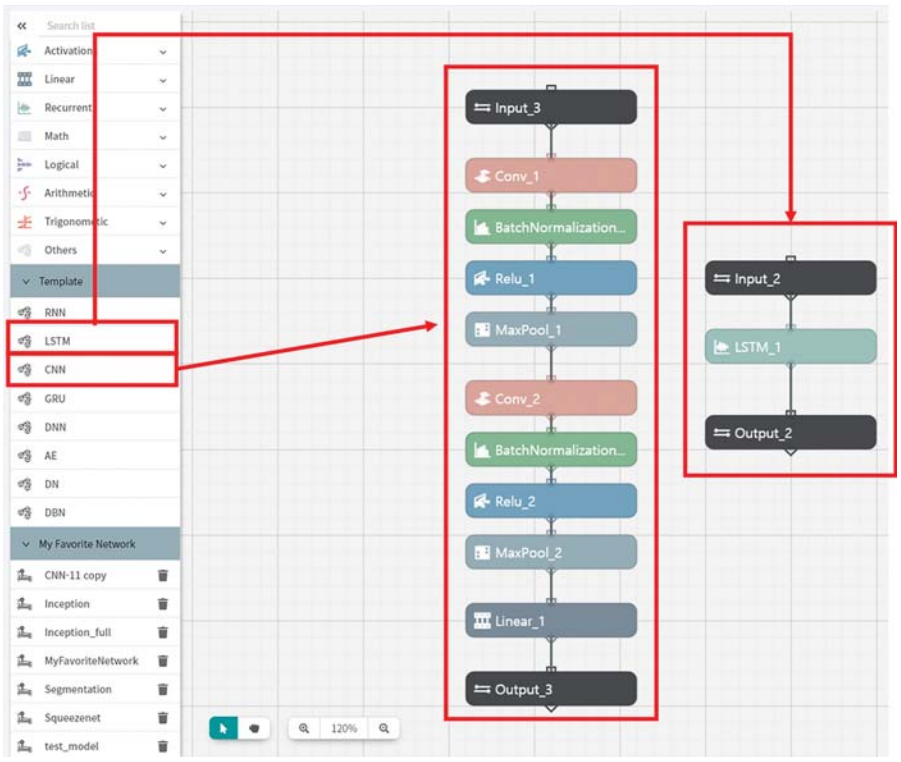


Figure 19: Open-type template and close-type templates

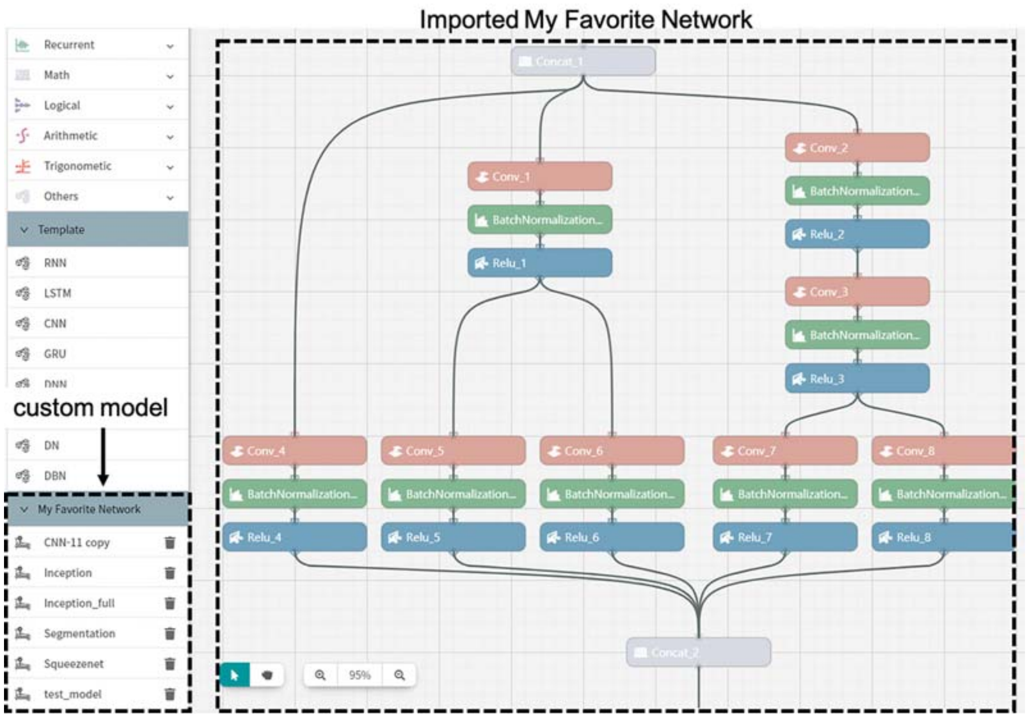


Figure 20: Using a custom model (Import My Favorite Network)

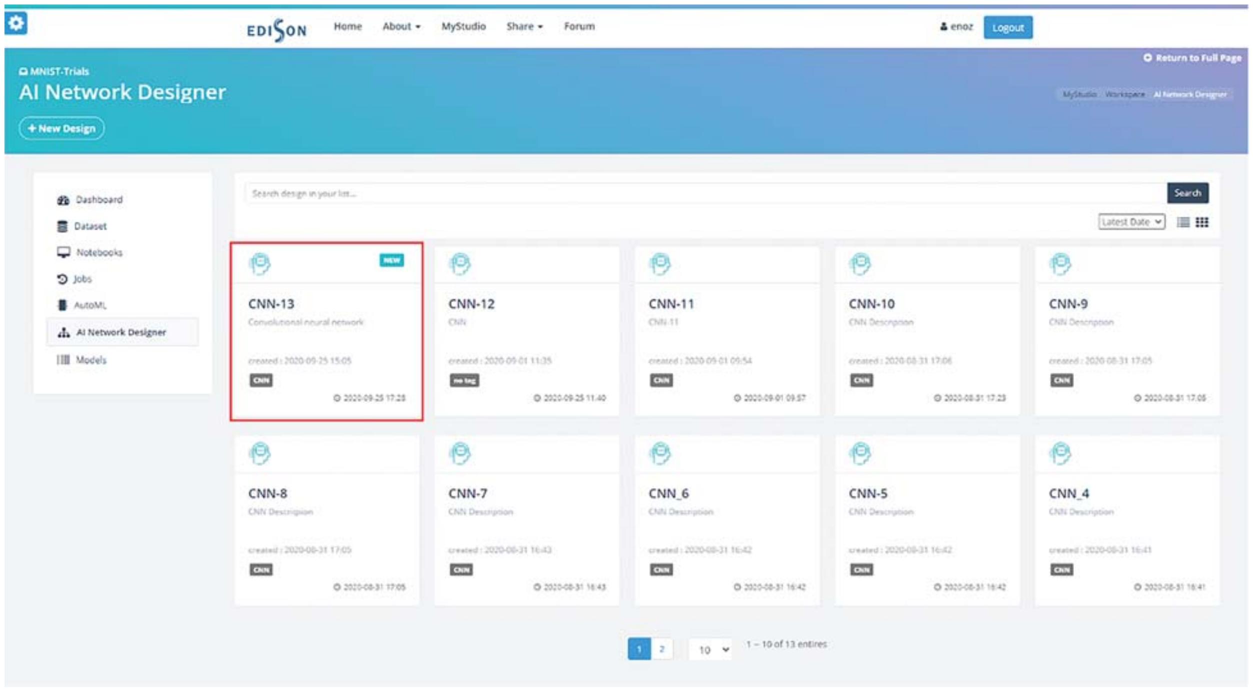


Figure 21: List of shared deep neural networks

6 Conclusion

DL is applied in various fields such as voice recognition, image recognition, and dielectrics. As a result, various frameworks for these have been proposed; among these, the DL framework provides libraries, DL models, and algorithms. However, it has the drawback of requiring a high degree of understanding for its effective utilization. An easy-to-use GUI-based DNN design tool that is easy to use to overcome these problems is thus being researched and developed.

This GUI-based DNN design tool can quickly design the DNN using the drag and drop method and select methods. However, the existing GUI-based DNN design tool has three drawbacks. First, there exists a problem concerning usability because the tool requires a high level of understanding similar to that needed for using the framework; moreover, it needs to be created from scratch every time. Second, it is designed based on a specific framework and a specific algorithm, and it possesses framework dependencies. Third, it cannot change GUI components easily. This is because a significant amount of work is required (e.g., a change in the design of the tool and modification of the Node to change the GUI component, in turn to change the version, algorithm, and component).

In this study, we developed a user-friendly GUI-based DNN design tool to solve these problems. First, we developed a Template to solve the usability issues and enhance the convenience of users. We also developed a deep neural network Save module and a Share module to improve reusability. Second, it appeared possible to apply ONNX to resolve the framework dependencies and define a JSON format and design a deep neural network without framework knowledge. Finally, each function was modularized to solve the challenging problem of changing the GUI components. In addition, the basic components constituting the deep neural network were developed by modularizing groups of elements, such that they could respond quickly to version updates. The developed GUI-based DNN design tool serves as a platform for KISTI's supercomputer users, and it is being verified in cooperation with KISTI's EDISON. In the future, we plan to make such a GUI-based DNN design tool available and extend it to reflect the generative AI algorithm and composite AI.

Funding Statement: This research was supported by the KISTI Program (No. K-20-L02-C05-S01) and the EDISON Program through the National Research Foundation of Korea (NRF) (No. NRF-2011-0020576). A grant was also awarded by the Ministry of Science and ICT (MSIT) under the Program for Returners for R&D.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.
[2] Z. Shen, F. Ding and Y. Shi, "Digital forensics for recoloring via convolutional neural network," *Computers, Materials & Continua*, vol. 62, no. 1, pp. 1–16, 2020.
[3] Y. S. Lee and P. J. Moon, "A comparison and analysis of deep learning framework," *The Journal of the Korea Institute of Electronic Communication Sciences*, vol. 12, no. 1, pp. 115–122, 2017.
[4] I. Hussain, J. Zeng, X. Hong and S. Tan, "A survey on deep convolutional neural networks for image steganography and steganalysis," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 3, pp. 1228–1248, 2020.
[5] Y. M. Park, E. J. Lim, S. Y. Ahn, W. Choi and T. W. Kim, "DL-dashboard: User-friendly deep learning model development environment," in *Proc. ACM*, New York, NY, USA, pp. 73–74, 2019.
[6] S. Klemm, A. Scherzinger, D. Drees and X. Y. Jiang, "Barista—A graphical tool for designing and training deep neural networks," arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1802.04626v1.
[7] K. Devi, D. Paulraj and B. Muthusenthil, "Deep learning based security model for cloud based task scheduling," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 9, pp. 3663–3678, 2020.
[8] W. Fang, F. Zhang, Y. Ding and J. Sheng, "A new sequential image prediction method based on LSTM and DCGAN," *Computers, Materials & Continua*, vol. 64, no. 1, pp. 217–231, 2020.
[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen *et al.,* "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," arXiv, 2016. [Online]. Available: https://arxiv.org/abs/1603.04467v2.
[10] A. Akundi, T. L. B. Tseng, Z. Cao and H. J. Kim, "A deep learning graphical user interface application on MATLAB," in *Proc. ASEE*, Salt Lake, UT, USA, 2018.
[11] M. G. Fernandes, "A live IDE for deep learning architecture," M. S. dissertation, University of Porto, Portugal, 2019.
[12] W. F. Lin, D. Y. Tsai, L. Tang, C. T. Hsieh, Y. C. Chou *et al.,* "ONNC: A compilation framework connecting ONNX to proprietary deep learning accelerators," in *Proc. AICAS*, Hsinchu, Taiwan, pp. 214–218, 2019.
[13] R. K. Sarvadevabhatla and R. V. Babu, "Expresso: A user-friendly GUI for designing, training and exploring convolutional neural networks," arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1505.06605v2.
[14] C. Versloot, "Visualizing your Neural Network with Netron," MACHINECURVE, 2020. [Online]. Available: https://www.machinecurve.com/index.php/2020/02/27/visualizing-your-neural-network-with-netron/.
[15] H. Nakahara, H. Yonekawa, T. Fuji, M. Shimoda and S. Sato, "GUINNESS: A GUI based binarized deep neural network framework for software programmers," *IEICE Transactions on Information and Systems*, vol. E102-D, no. 5, pp. 1003–1011, 2019.