W254–W260 *Nucleic Acids Research, 2022, Vol. 50, Web Server issue* *Published online 12 May 2022*
https://doi.org/10.1093/nar/gkac369

DLEB: a web application for building deep learning models in biological research

Suyeon Wy†, Daehong Kwon†, Kisang Kwon and Jaebum Kim*

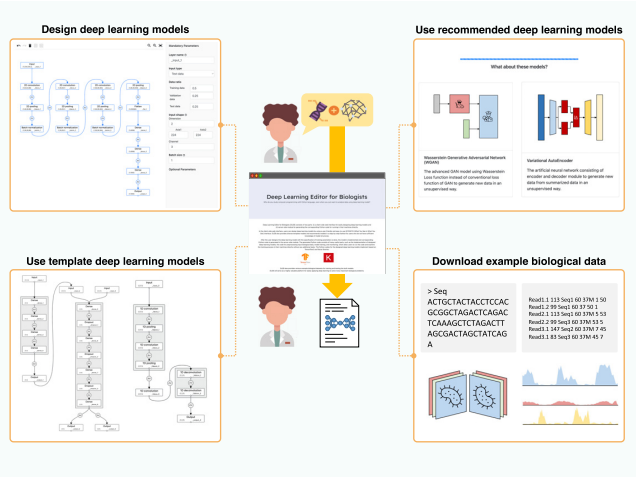
Department of Biomedical Science and Engineering, Konkuk University, Seoul 05029, Republic of Korea

Received March 11, 2022; Revised April 14, 2022; Editorial Decision April 27, 2022; Accepted April 28, 2022

ABSTRACT

Deep learning has been applied for solving many biological problems, and it has shown outstanding performance. Applying deep learning in research requires knowledge of deep learning theories and programming skills, but researchers have developed diverse deep learning platforms to allow users to build deep learning models without programming. Despite these efforts, it is still difficult for biologists to use deep learning because of limitations of the existing platforms. Therefore, a new platform is necessary that can solve these challenges for biologists. To alleviate this situation, we developed a user-friendly and easy-to-use web application called DLEB (Deep Learning Editor for Biologists) that allows for building deep learning models specialized for biologists. DLEB helps researchers (i) design deep learning models easily and (ii) generate corresponding Python code to run directly in their machines. DLEB provides other useful features for biologists, such as recommending deep learning models for specific learning tasks and data, pre-processing of input biological data, and availability of various template models and example biological datasets for model training. DLEB can serve as a highly valuable platform for easily applying deep learning to solve many important biological problems. DLEB is freely available at http://dleb.konkuk.ac.kr/.

GRAPHICAL ABSTRACT



INTRODUCTION

Deep learning is one class of machine learning algorithms that can extract important features from raw data by themselves in an end-to-end method (1). Dramatic improvement of computational power and learning algorithms, as well as the accumulation of enormous sets of biological data, have enabled the use of deep learning for solving many biological problems (2) including transcription factor binding site prediction (3,4), variant detection (5,6), and biomedical image diagnosis (7,8). Diverse libraries based on Python programming language, including Tensorflow (https://www.tensorflow.org/), Keras (https://keras.io/) and PyTorch (https://pytorch.org/), have been developed to help implement deep learning models. However, they are difficult to use without knowledge of deep learning theories and programming skills.

Accordingly, diverse deep learning platforms such as Deep Cognition (https://deepcognition.ai/), Watson Studio Neural Network Modeler (https://www.ibm.com/cloud/watson-studio), and Neural Network Console (https://dl.sony.com) have been developed to allow users to build deep learning models without programming. Nevertheless, biol-

*To whom correspondence should be addressed. Tel: +82 2 450 0456; Fax: +82 2 444 3490; Email: jbkim@konkuk.ac.kr
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.

ogists are still discouraged from applying deep learning in their research because existing deep learning platforms only focus on implementing general deep learning models. To successfully apply deep learning models in research, it is important to choose the most appropriate model for the prepared input data and the specific learning task (2). For example, the convolutional neural network (CNN) is useful for extracting and using local patterns in image data (9), and the recurrent neural network (RNN) is a good model for processing sequential or signal data (1). However, existing platforms do not provide enough example models or recommend appropriate models specialized for a specific learning task that users want to solve. Therefore, it is difficult for biologists to choose appropriate deep learning models in those platforms.

Existing deep learning platforms also do not support preprocessing of biological data, an essential step when using such data in deep learning. However, it is difficult to be carried out by biologists, especially if they do not have enough programming skills. The input data for a deep learning model is typically represented as a matrix with numerical or categorical values. However, because most biological data is not in the shape of a matrix, it has to be preprocessed to be converted into an allowed format. Thus, a pipeline for pre-processing the biological data can be useful for biologists. Furthermore, recent biological studies have applied various deep learning models with complicated structures including a generative adversarial network (GAN) (10) and a Wasserstein generative adversarial network (WGAN) (arXiv preprint arXiv:1712.06148). However, building such complex deep learning models in existing platforms is still limited and furthermore, most existing platforms are not freely available to users. Therefore, a new deep learning platform that can solve the above limitations especially for biologists is necessary.

We developed a user-friendly and easy-to-use web application called DLEB (Deep Learning Editor for Biologists) for building deep learning models specialized for biologists. DLEB can help researchers (i) easily design deep learning models and (ii) generate corresponding Python code for running in their machines directly. DLEB provides additional useful features for biologists, such as recommending deep learning models for specific learning tasks and data, pre-processing input biological data, making available of various template models, and example biological datasets for model training. DLEB can serve as a highly valuable platform for easily applying deep learning to solve many important biological problems.

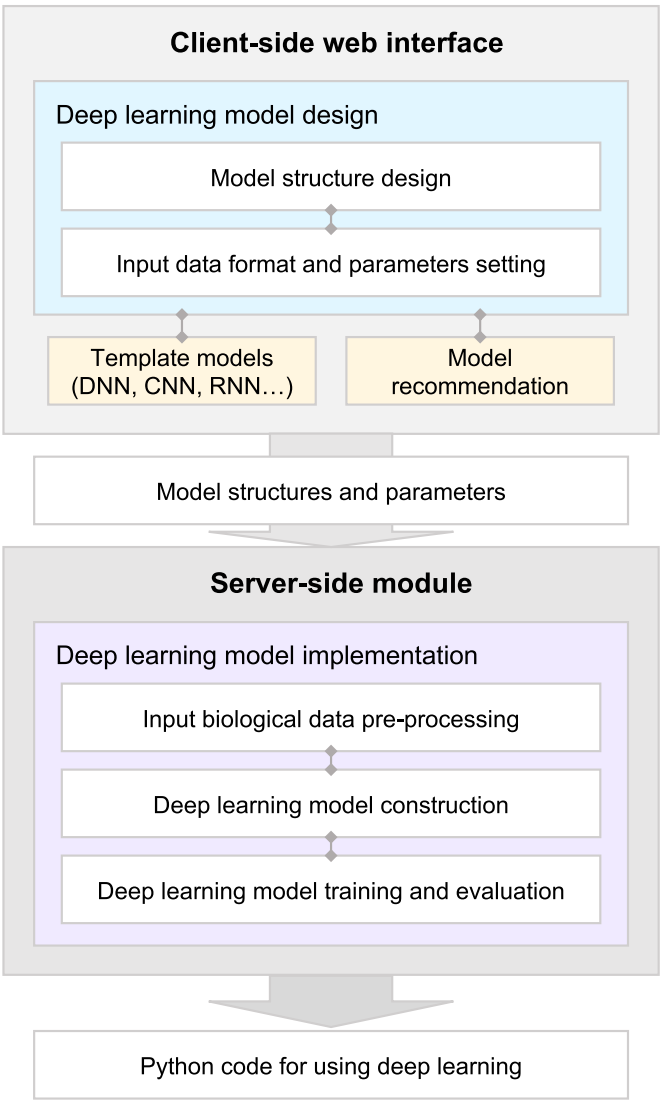


Figure 1. DLEB workflow. In the client-side web interface, the structure of deep learning models is designed using an easy-to-use web interface of DLEB (top panel). The format of input biological data and parameters are also set in the web interface. Users can use template models and the model recommendation function as they design models (yellow boxes in the top panel). In the server-side module, models designed in the client-side web interface are imported into Python code using the Tensorflow2 and Keras deep learning libraries (bottom panel). The final Python code generated by the server-side module in DLEB includes all required code, ranging from input data pre-processing to model training and validation.

MATERIALS AND METHODS

Web application overview

DLEB consists of two parts: a client-side web interface and a server-side module (Figure 1). In the client-side web interface, users can design deep learning models using a user-friendly and easy-to-use WYSIWYG (What You See Is What You Get) interface. Diverse template models and step-by-step model recommendations are also provided for users who lack adequate knowledge of model structures. After the users design the deep learning model with the specified training parameters, DLEB implements the model and generates a Python code in the server-side module. The generated Python code consists of many useful parts, such as implementation of designed deep learning models, code for pre-processing input biological data, model training, and monitoring, that allow users to run the code and examine the training process in their machines directly without any additional tasks. The details of functions provided by DLEB are described in the following subsections.

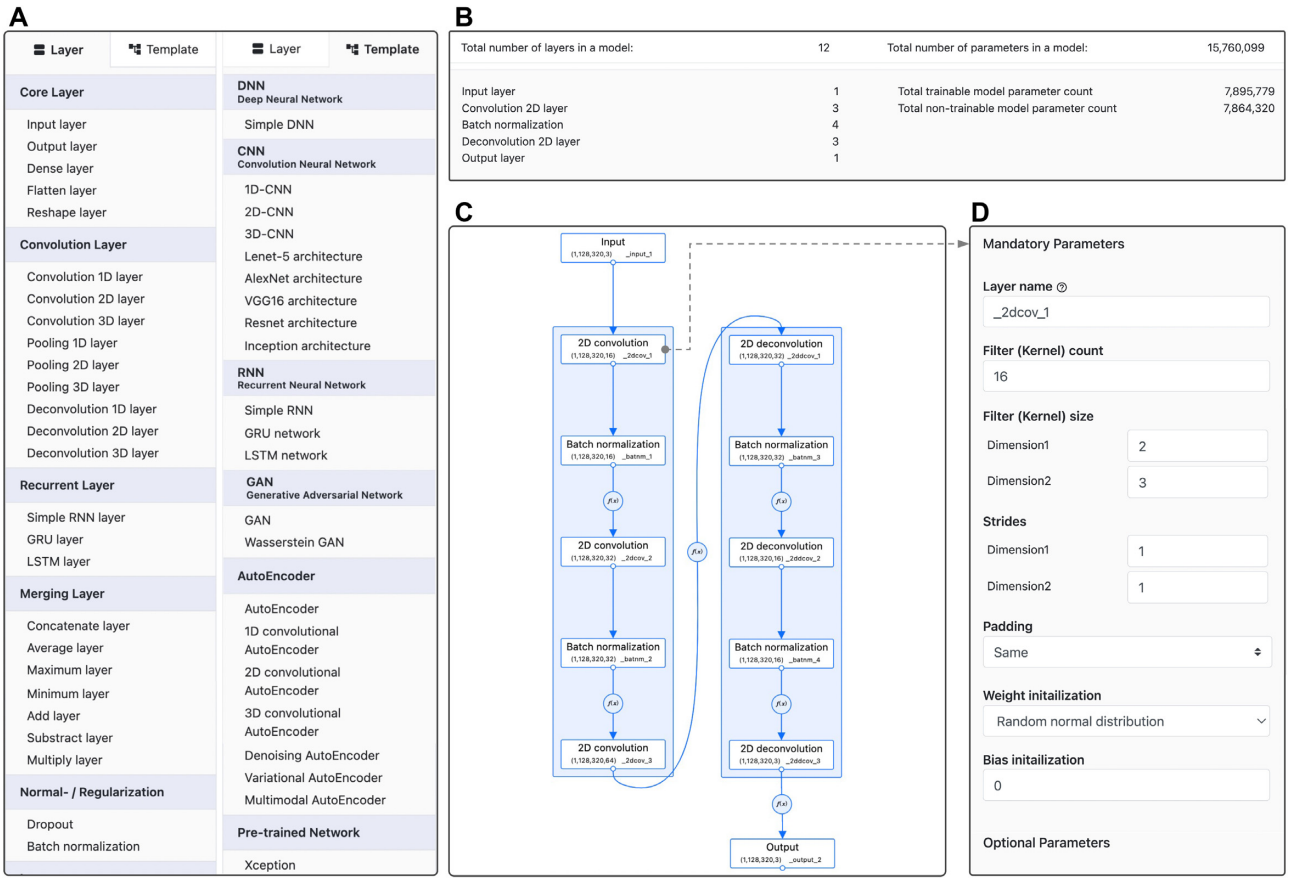


Figure 2. Client-side web interface of DLEB. (**A**) Layer and Template panel for adding layers in models. (**B**) Information panel for displaying the number of layers and parameters in current models. The numbers are changed in real-time. (**C**) Designed models are visualized as in graph form in which nodes and edges represent layers and connections between layers, respectively. Multiple layers can be grouped and visualized as a layer group (blue boxes). Activation functions are presented as circles with a function icon positioned on edges. (**D**) Parameter panel for setting various parameters of each layer and activation functions.

Deep learning model design in the client-side web interface

DLEB provides an easy-to-use web interface for designing and constructing deep learning models using simply mouse click and drag (Figure 2). DLEB supports diverse types of layers, activation functions, and pre-trained models (layer panel in Figure 2A). Users can easily set various parameters for each layer such as an activation function in the parameter panel (Figure 2D). For user's convenience, DLEB provides commonly used parameter values as defaults. Users can also add custom layers and define their operation. Designed models are visualized as a graph structure where nodes and edges represent layers consisting of the model and connections among them, respectively (Figure 2C). Additionally, multiple layers designed for working together for the same function such as encoder and decoder can be grouped and visualized as a layer group (blue boxes in Figure 2C). Activation functions, which transform outputs of layers, are represented as circles with a function icon positioned on edges.

For researchers with little experience in designing deep learning models, DLEB provides diverse template models (Template panel in Figure 2A) and recommends deep learning models (Figure 3). DLEB supports widely used deep learning models as templates including DNN (Deep Neural Network), CNN, RNN, GAN, WGAN and autoencoder as well as popular pre-trained neural networks including Xception (11), VGG (arXiv preprint arXiv:1409.1556) and ResNet (12). DLEB recommends deep learning models appropriate for a selected learning task with its available input data (Figure 3A–C). When the users choose a model among the recommended ones, DLEB generates a template for the model in the web interface for further customization (Figure 3D).

Moreover, DLEB provides various information on constructed models that could be used to build complete models. For example, DLEB can calculate and display numbers of layers and parameters in deep learning models that are updated in real time during model construction (Figure 2B). Those numbers can be valuable information for users to reduce the size of the model or the time needed for training. DLEB also checks the integrity of model structures and parameter values before generating corresponding Python code.

Deep learning model implementation in the server-side module

Given a model structure and user-defined parameters, DLEB generates Python code in the server-side module for running in users' machines directly without additional programming. The Python code consists of multiple parts:

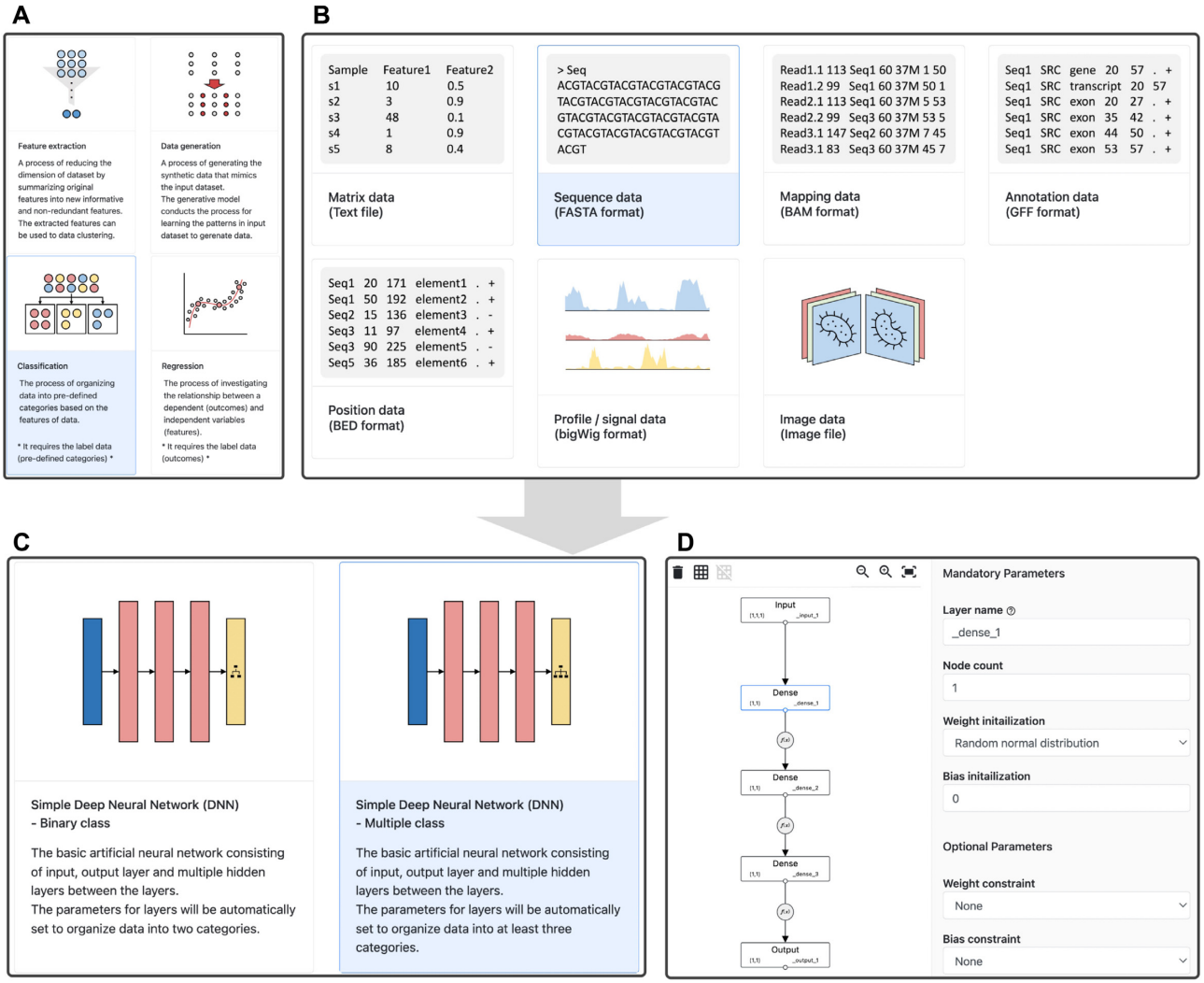


Figure 3. Model recommendation function of DLEB. (**A**) Web interface for selecting a learning task. Users can choose one of the learning tasks including 'Feature extraction', 'Data generation', 'Classification', and 'Regression'. (**B**) Web interface for selecting the type of available data. (**C**) Examples of models recommended by DLEB. (**D**) Web interface for customizing a recommended model.

pre-processing input biological data, implementing the designed model, and training and monitoring the model. It is difficult to directly use data in formats specialized for biological data, such as FASTA, BAM, bigWig, VCF and GFF, as input data for deep learning models. Therefore, DLEB generates Python code for calculating biological features in each genomic region of interest and summarizing them in a matrix format using the Python library 'janggu' (13) and in-house Python scripts. Image and text data can also be pre-processed into pixels and string arrays, respectively, which are summarized in NumPy (14) array format that can be directly used in deep learning models. For model training and evaluation, the pre-processed dataset can be divided into training, validation, and test datasets. The generated code can be also used to post-process the output of output layers by converting it into the original format of the input data such as a sequence or an image. The generated Python code includes code for all of the above tasks.

The designed deep learning models are implemented in the server-side module using Tensorflow2 and Keras, the two most popular deep learning libraries. For handling models with complex structures including multiple inputs or outputs and non-linear topology, the Keras functional API, which is more flexible than the Keras sequential API, is used. To identify the order of layers in the model structure efficiently, the depth-first search algorithm is used for traversing the graph structure of models. Furthermore, the final code includes code for training, testing, monitoring, and saving models. Therefore, all processes from data pre-processing to model training can be executed directly by running just the generated Python code. Additionally, users can obtain intermediate outputs generated by hidden layers by using a command-line option of the generated Python code.

Example biological datasets

There are various public example datasets for evaluating the performance of deep learning models, such as MNIST (15), Street View House Numbers (16), and the 1 Bil-

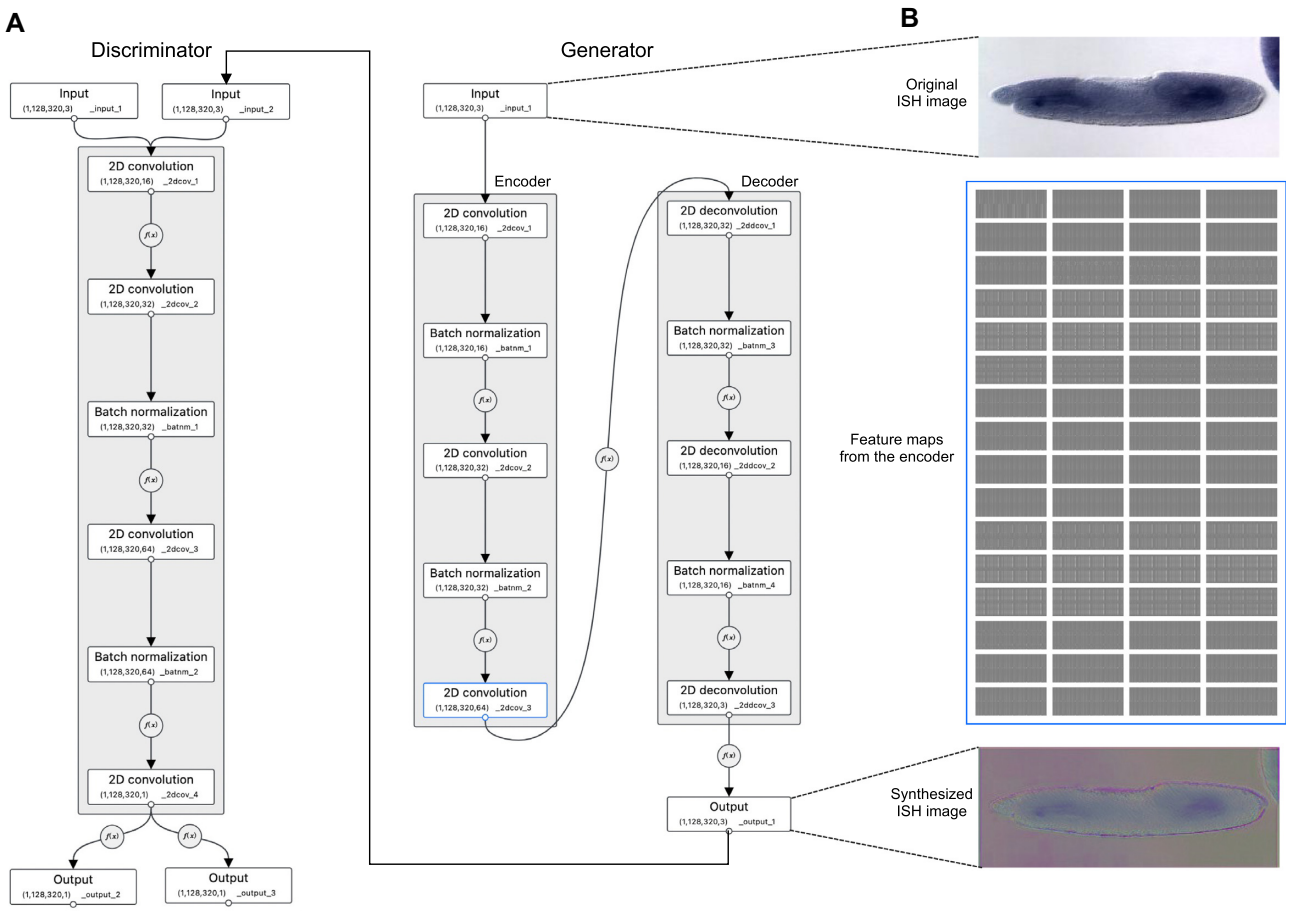


Figure 4. Example generative adversarial network (GAN) model designed using DLEB. (**A**) The model structure of GAN consisting of a discriminator and a generator. (**B**) Images of the input and the output of the generator in the GAN model. Feature maps of the last hidden layer in the encoder are also visualized as gray-scale images (blue box).

lion Word Language Model Benchmark datasets (arXiv preprint arXiv:1312.3005). Even though there are unique characteristics of biological data compared with other datasets including high complexity (17) and variability (18), there are no example datasets specialized for evaluating deep learning models designed for biological input data. Therefore, DLEB provides example biological datasets that researchers have used in deep learning models in recent biological studies (5,19–21). DLEB supports example datasets with diverse types including sequence, alignment, signal, and image and provides example deep learning models for each data type.

Implementation

The server-side module was written by Python (version 3.8.10). The client-side web interface was implemented using HTML5 (https://www.w3.org/TR/html5/), Bootstrap (version 4.6, https://getbootstrap.com/), and JavaScript (https://www.javascript.com/) along with several libraries, such as jQuery (https://jquery.com/) and d3.js (https://d3js.org/). PHP (version5.3.3, https://php.net) was used for Python code generation.

RESULTS

To illustrate the capability and usefulness of DLEB, we built a GAN model that generates in situ hybridization (ISH) images of Drosophila embryo (Figure 4). In this experiment, we randomly selected 1,000 ISH images obtained from the Berkeley Drosophila Genome Project (BDGP) dataset (22) available in the DLEB website to use as training data. The GAN model was designed to contain two sub-models: a generator that could generate the ISH images and a discriminator that could distinguish real images from images the generator provided (Figure 4A). The generator comprises two parts: encoder and decoder. The encoder has three convolutional layers and two batch normalization layers with a LeakyReLU activation function, and the decoder consists of three deconvolution layers and two batch normalization layers with a ReLU activation function. To generate image vectors with values ranging from –1 to 1, a Tanh activation function is fed to the result of the final decoder block. The discriminator consists of four convolutional layers followed by the LeakyReLU activation function and batch normalization. The resulting feature vector is fed to a sigmoid activation function for binary classification. To train this model, we set the batch size to 1 and the number of

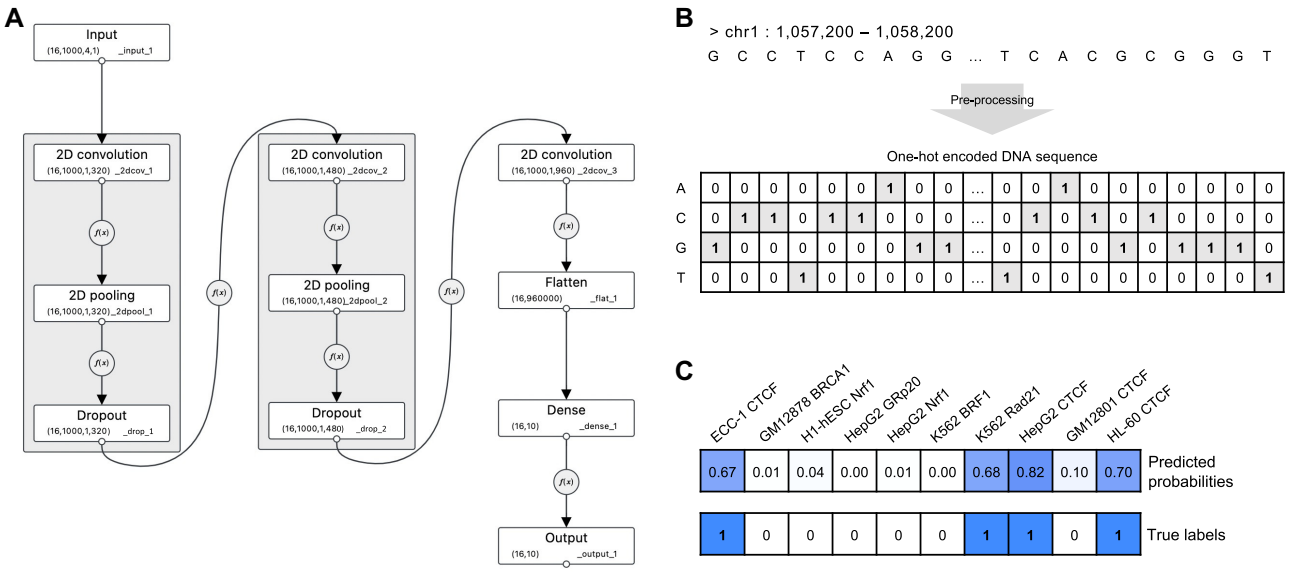


Figure 5. Example convolutional neural network (CNN) model designed using DLEB. (**A**) The architecture of the CNN model. (**B**) Pre-processing of an input DNA sequence. A DNA sequence in 1000 bp length is converted to a one-hot encoded matrix using a source code provided by DLEB. (**C**) Predicted probabilities of chromatin features (top row) and true labels (bottom row; 1 means the input sequence is related to the corresponding chromatin feature).

epochs to 100, and used the Adam optimizer with a learning rate of 1E–5 and a cross entropy loss function. Figure 4B shows an example of input ISH image, the feature maps of the last hidden layer in the encoder, and the synthesized ISH image from the generator, which was similar to the input image.

We next built a CNN model that predicts chromatin features from a given DNA sequence as designed and described in (21) (Figure 5A). For training and testing the model, we collected 99 298 DNA sequences in 1000 bp length with labels of 10 chromatin features, and split them to training (80%), validating (10%), and testing (10%) dataset. The DNA sequences in FASTA format were converted to matrices containing one-hot encoded nucleotides using a source code for pre-processing input data provided by DLEB (Figure 5B). The CNN model consists of two 2D convolution blocks, a 2D convolution layer, and a fully connected layer (Figure 5A). The 2D convolution block contains a 2D convolution layer with a ReLU activation function followed by a max pooling and a dropout layer. The output of the second 2D convolution block is fed to the 2D convolution layer and then to a fully connected layer. The final output vector is then transformed with a sigmoid activation function for multi-label classification of 10 chromatin features. For training this model, the batch size was set to 16 and the SGD optimizer was used with a learning rate of 1E-3 and a mean squared error loss function. Figure 5C shows an example of the probability of each chromatin feature predicted by the CNN model (top row) and a true label (bottom row; 1 means the input sequence is related to the corresponding chromatin feature). By using 0.5 as the probability cutoff for classification, we can observe perfect agreement between the predicted and true labels.

The deep learning models shown in Figures 4A and 5A were solely designed by using the client-side web interface in DLEB. The generated Python code for those models (Supplementary Data S1 and S2) was successfully run in a machine with the NVIDIA Quadro RTX 5000 GPU. These experiments clearly showed the applicability of DLEB for solving biological problems using deep learning.

CONCLUSIONS

DLEB is a user-friendly web application for designing and implementing deep learning models for biologists. It helps users design and implement deep learning models with an easy-to-use web interface and various useful functions. DLEB can serve as a highly valuable platform for easily applying deep learning to solve many important biological problems. Because various biological data can be presented in graph form, future updates in DLEB will include the support of graph-based neural networks such as graph convolutional networks.

DATA AVAILABILITY

DLEB is freely available at http://dleb.konkuk.ac.kr/.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

FUNDING

Ministry of Science and ICT, Republic of Korea [2021M3H9A2097134, 2014M3C9A3063544, 2019R1F1A1042018]. Funding for open access charge: Ministry of Science and ICT, Republic of Korea.
Conflict of interest statement. None declared.

REFERENCES

1. Goodfellow,I., Bengio,Y. and Courville,A. (2016) In: *Deep Learning*. MIT Press.
2. Eraslan,G., Avsec,Z., Gagneur,J. and Theis,F.J. (2019) Deep learning: new computational modelling techniques for genomics. *Nat. Rev. Genet.*, **20**, 389–403.
3. Koo,P.K. and Ploenzke,M. (2020) Deep learning for inferring transcription factor binding sites. *Curr. Opin. Syst. Biol.*, **19**, 16–23.
4. Quang,D. and Xie,X. (2019) FactorNet: a deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data. *Methods*, **166**, 40–47.
5. Poplin,R., Chang,P.C., Alexander,D., Schwartz,S., Colthurst,T., Ku,A., Newburger,D., Dijamco,J., Nguyen,N., Afshar,P.T. *et al.* (2018) A universal SNP and small-indel variant caller using deep neural networks. *Nat. Biotechnol.*, **36**, 983–987.
6. Sahraeian,S.M.E., Liu,R.L., Lau,B.Y., Podesta,K., Mohiyuddin,M. and Lam,H.Y.K. (2019) Deep convolutional neural networks for accurate somatic mutation detection. *Nat. Commun.*, **10**, 1041.
7. Esteva,A., Kuprel,B., Novoa,R.A., Ko,J., Swetter,S.M., Blau,H.M. and Thrun,S. (2017) Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, **542**, 115–118.
8. Kermany,D.S., Goldbaum,M., Cai,W., Valentim,C.C.S., Liang,H., Baxter,S.L., McKeown,A., Yang,G., Wu,X., Yan,F. *et al.* (2018) Identifying medical diagnoses and treatable diseases by image-Based deep learning. *Cell*, **172**, 1122–1131.
9. Yamashita,R., Nishio,M., Do,R.K.G. and Togashi,K. (2018) Convolutional neural networks: an overview and application in radiology. *Insights Imaging*, **9**, 611–629.
10. Gupta,A. and Zou,J. (2019) Feedback GAN for DNA optimizes protein functions. *Nat. Mach. Intell.*, **1**, 105–111.
11. Chollet,F. (2017) Xception: deep learning with depthwise separable convolutions. *Proc. Cvpr. IEEE*. pp. 1800–1807.
12. He,K.M., Zhang,X.Y., Ren,S.Q. and Sun,J. (2016) Deep residual learning for image recognition. *2016 Ieee Conference on Computer Vision and Pattern Recognition (Cvpr)*. pp. 770–778.
13. Kopp,W., Monti,R., Tamburrini,A., Ohler,U. and Akalin,A. (2020) Deep learning for genomics using janggu. *Nat. Commun.*, **11**, 3488.
14. Harris,C.R., Millman,K.J., van der Walt,S.J., Gommers,R., Virtanen,P., Cournapeau,D., Wieser,E., Taylor,J., Berg,S., Smith,N.J. *et al.* (2020) Array programming with NumPy. *Nature*, **585**, 357–362.
15. Deng,L. (2012) The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag*. Vol. **29**, pp. 141–142.
16. Netzer,Y., Wang,T., Coates,A., Bissacco,A., Wu,B. and Ng,A.Y. (2011) Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.
17. Xu,C.M. and Jackson,S.A. (2019) Machine learning and complex biological data. *Genome Biol.*, **20**, 76.
18. Fay,D.S. and Gerow,K. (2013) A biologist's guide to statistical thinking and analysis. *WormBook: the Online Review of C. elegans Biology*, 1–54.
19. Cai,L., Wang,Z., Kulathinal,R., Kumar,S. and Ji,S. (2021) Deep low-shot learning for biological image classification and visualization from limited training samples. *IEEE Trans. Neural Netw. Learn. Syst*. https://doi.org/10.1109/TNNLS.2021.3106831.
20. Ernst,J. and Kellis,M. (2012) ChromHMM: automating chromatin-state discovery and characterization. *Nat. Methods*, **9**, 215–216.
21. Zhou,J. and Troyanskaya,O.G. (2015) Predicting effects of noncoding variants with deep learning-based sequence model. *Nat. Methods*, **12**, 931–934.
22. Mace,D.L., Varnado,N., Zhang,W.P., Frise,E. and Ohler,U. (2010) Extraction and comparison of gene expression patterns from 2D RNA in situ hybridization images. *Bioinformatics*, **26**, 761–769.