Building Block Identification from Deep Neural Network Codes for Deep Learning Modeling Support

[1]Ki Sun Park, Kyoung Soon Hwang, Keon Myung Lee

Abstract: ***Background/Objectives: Recent successful applicationsof deep learning techniques have been attracting many attentions of developers in various domains. There are many publically available codes of deep learning models to which developers refer. This paper introduces a method to identify building blocks of deep neural network models for reuse and visualization in model development. Methods/Statistical analysis: Deep learning models have a layered architecture the number of which layers can be tens to hundreds. Developers design their own deep learning model by composing it with layers. GUI-based modeling tool can be very useful to design such model with which a new model is created or existing deep learning model codes are imported and modified. To reuse useful building blocks in existing models and visualize models in abstract and hierarchical view, a building block identification method is proposed which uses a graph structure analysis technique and a frequent subsequence mining technique. It first transforms a deep neural network model into a graph structure, then identify macro-blocks, and after that mines the frequent consecutive subsequences. Findings:The macro-blocks in deep neural networks are detected by the proposed algorithm which isolates subgraphs that starts with a node with a single fan-in, ends with a node with fan-out 1, and has nodes with more than one fan-in or fan-out between the start and the end nodes. The frequent consecutive subsequences of nodes are recognized as candidate building blocks for deep learning model construction. Building blocks may contain smaller building blocks so that a hierarchical and abstract entailment can be implemented in the visual representation of deep learning models with many layers. As the building blocks, the macro-blocks and the frequent consecutive blocks are chosen and registered in the GUI-based deep learning modeling tool. With the GUI-based model tool, developers can easily design deep learning models using the building blocks and can visualize conveniently the deep learning models with many layers. Improvements/Applications: The proposed building block identification method helps developers build deep learning models by providing useful building blocks identified from existing deep learning models and enables to visualize deep learning models with many layers by automatically organizing the hierarchies on such models.***

Keywords: Deep learning modeling, Code reuse, Model visualization, Frequent Pattern Mining

I. INTRODUCTION

Deep learning is an advanced learning technique originated from the neural networks, which has demonstrated surprising performances compared to existing machine learning techniques in various domains.[1,2,3] Many developers have interest in applying deep learning techniques to their own problems of their domains. Deep learning is realized using deep neural network models which are composed of many layers. The architectures of deep neural network models have strong influence of the performance of the models. The developers usually try various models to find good deep learning model which is appropriate to their problem. To get a quality model, developers should have sufficient understandings on deep learning models and plentiful of practicing experiences on deep learning model developments.

Deep learning models are organized into a layered architecture with tens or hundreds layers. To help model developments, there have been developed some GUI(graphical user interface)-based tools in which developers can design the model architecture in a drag-and-drop manner.[4] It is convenient to use such tools when the number of layers is small enough to draw on the window of the tools. For the models with many layers, it needs different approach to handling many layers. A promising approach is to provide the hierarchical view for the deep learning models each of level represents an abstraction level. To enable the hierarchical view, there needs some method to enforce the abstraction levels in the modeling tool. The functionality can be realized by providing the macro-block modules in the model design tool. A macro-block represents a module that contains a subgroup of a deep learning model which can be a sequence of layers or a structured module organized in an acyclic graph of layers. A macro-block may contains another macro-block so that a hierarchical design and view can be allowed. In the GUI-based modeling tool, the developers are asked to design the models in the top-down manner from the most abstract modules to more and more concrete submodules. For the successful and effective design of deep learning models, the developers should have good insights on the model architecture on their application problems. Therefore, the developers should be experienced and have good knowledge on deep learning applications.

It is helpful for model designing to provide the developers with the candidate building blocks for constructing deep learning models. The availability of such building blocks help the developers amplify their imagination and creativity in the model design. Such functionality will be especially beneficial to non-experienced developers. The issue is here for such design tools to be equipped with the building blocks for deep learning models.

Revised Manuscript Received on January 03, 2019.

Ki-Sun Park and **Kyoung Soon Hwang**, Dept. of Computer Science, Chungbuk National University, Cheongju, 28644 Korea

Keon Myung Lee, Corresponding author, Dept. of Computer Science, Chungbuk National University, Cheongju, 28644 Korea

Many excellent deep learning models have been published in the open source to the public on the Internet. Developers sometimes refer to such models when developing their own models. Hence, the publicly available source codes of deep learning models are useful resources from which some valuable building blocks are extracted. This paper is concerned with identifying such building blocks by analyzing the source codes.

The deep learning models can be expressed as an acyclic graphical model with a single starting node and a single ending node. The starting node corresponds to the input at the problem, and the ending node corresponds to the output of the model. Some repeating subgraphs and articulable subgraphs can be a building block. In a deep learning model, each layer is characterized by its own parameters. For example, a convolution layer in a convolutional neural network is characterized by dimension and channel of the input, number of filters, filter size, padding, and stride. On identifying the repeating subgraphs, the information of layers are generalized to ignore fine details and capture the essential characteristics. In the graphical representation of a deep learning model, each node is expressed in terms of the input dimension and channel, layer operation, output dimension and channel. It means that two nodes are regarded as being repeated in the graph when they have the same information for those characteristics.

In a graphical representation of the deep learning models, an articulable subgraph indicates a subgroup in which there are a single starting node and a single ending node, and there also exist many other nodes between the two nodes. In a deep learning model, some specialized modules carry out specialized roles like extracting some distinctive features. An articulable subgraph may contain other articulable subgraphs in itself. Such inclusive articulable subgraph may appear in different articulable subgraphs. The frequent articulable subgraphs can be useful as a building block for deep learning model construction. As the representative examples of articulable model, there are the inception module in GoogleNet[5] and the residual module in ResNet[6]. We propose a method to identify the articulable subgraphs from source codes of deep learning models. The proposed method first spots all articulable subgraphs from the graph structure for a deep learning model, then converts a graph structure into a sequence of generalized node labels, and after that identifies the meaningful building blocking based on their frequencies and size.

There have been some works to assist developers to design deep learning models. A GUI-based model construction tool was proposed which developers can design deep learning models without direct code editing and can train the designed model.[7] Their tool does not yet support the hierarchical representation and design of deep learning models. A genetic programming-based approach was proposed for designing convolutional neural network (CNN) architectures.[8] The method makes various CNN models composed with functional modules like convolutional blocks and tensor concatenation blocks, and then trained in an evolutionary manner over the generations. A document analysis method was proposed to extract and understand deep learning diagrams and tables from the published articles and transform them into an executable code.[9] Theapproach is ideal but yet restricted to extract sufficient information for deep learning model construction from research articles which describes the model architecture in a brief manner. Some design principle for deep learning model was proposed in which the number of layers are increased along with the extension of the width of each layer while monitoring the performance increase.Along with the principle, a concept of SimpNet was introduced, butthe hierarchical and abstract levels was not considered in its model design.[10]An automated construction method for CNN model was introduced which used a hyperparameter optimization technique with some new objective function.[11]There has been some work for autonomic machine learning framework on which the framework gradually takes care of the burdens of the developers in an automated manner.[12] The framework does not have the functionalities for assisting the developers with building blocks to design deep learning models and for visualizing the models in a hierarchical manner.

The remainder of this paper is organized as follows: Section 2 presents the methods to generalize layers of deep learning models into an abstract level, to represent the deep learning models in a graph structure, to identify articulable subgraphs, to find the frequent subgraphs, and to determine the candidate building blocks. Section 3 shows how the proposed methods have been implemented and adopted in a GUI-based tool. Finally, Section 4 draws the conclusions.

II. MATERIALS AND METHODS

We propose a technique which detects the macro-blocks, identifies the repeating patterns of consecutive layers, and proposes the candidate building blocks for deep learning model construction and hierarchical model visualization.

2.1. *Graphical Representation of Deep Learning Models*

Deep learning models are organized into a layered architecture each of which layer performs specific operations for feature extraction and desired output generation. Each layer is connected to some preceding layer(s) from which it receives its input. A deep learning model can be hence represented by a graph $G = (V, E)$which consists of nodes V and edges E as follows: Each layer corresponds to a node and the connections from a layer to another layer is represented by a directed edge from the corresponding node to the other corresponding node. We want to find frequent or important subgroups which can be building blocks. Each layer has its own characteristics such as its operation and association parameters, e.g., input dimension and channels, filter size, stride and padding in case of a convolution layer of CNN models. We generalize the layers into generic labels by suppressing the parametric factors not associated with model architecture, because they can be adjusted at the later phase of model design. As such suppressed factors in convolution layers, there are filter size, stride, and padding information. Each node keeps the information about its operation, dimensions and channels of its input and output, respectively. The nodes with the same information are labelled with the same symbols.

Once a deep learning model is converted into a graph, it is assumed that all nodes are labelled with symbols like A, B, C and so on.

2.2. ***Detection of Articulable Subgraph***

The graph model for a deep learning model can be non-linear, that is, it may have some structure at which some nodes have fan-in of more than one or fan-out of more than one. Here fan-in indicates the number of preceding nodes for a node and fan-out indicates the number of following nodes for a node in a graph. An articulable subgraph is defined as a subgraph which starts with a single node, ends with a single node, and there are multiple paths from the starting node to the ending node. The articulable subgraphs are regarded as a functional module which may be used as a building block in other deep learning model. An articulable subgraph may uses other articulable subgraphs and hence the nested structure can be constructed. If an articulable subgraph is treated as an abstract node, a graph model for a deep learning model can be viewed as a sequence of nodes. When an articulable subgraph contains other articulable subgraphs, the abstract node is detailed by a sequence of nodes in which some nodes are abstract nodes corresponding to nested articulable subgraphs. This abstraction organization and linearization provides the hierarchical view and development for deep learning models.

An articulable subgraph at the top level can be identified by thefollowing procedure *Find-Articulable-Subgraph*. As the input, the procedure takes the graph model $G = (V, E)$of a deep learning model and its starting node N_s and sends as the output the set $AS$of articulable subgraphs.

Procedure*Find-Articulable-Subgraph*:

Input :the graph model $G = (V, E)$, starting node N_s

Output :the set AS of the articulable subgraphs at the top level, the modified graph model $G' = (V', E')$

begin

1.**foreach** N_i

2.$N_i.token \leftarrow 0$

3.make an empty queue Q

4.$Q.push(N_s)$

5.**while**Q is not empty

6.$N_j \leftarrow Q$.pop( )

7.**foreach**N_k adjacent to N_j (i.e.,$(N_j, N_k) \in E$)

8.$N_k.token \leftarrow \sum^{N_j.fan\_in} \frac{N_j.token}{N_j.fan\_out}$

9.$Q.push(N_k)$

10.$Q \leftarrow \emptyset$

11.$Q.push(N_s)$

12. **while**Q is not empty

13. $N_j \leftarrow Q$.pop( )

14. **foreach**N_k adjacent to N_j (i.e.,$(N_j, N_k) \in E$)

15. **if**$N_k.token = 1$

16. $SG \leftarrow Q.popall( )$

17. **if** |SG| > 3

18. G ←replace the subgraph corresponding to SG with an abstract node A_k

19. **else**

20. Q.push(N_k)

21.**return**AS as the articulable subgraphs **and** G as the output G'

end.

To detect the nested articulable subgraphs, we apply the above procedure for each already-detected articulable subgraphs by the following procedure *Find-Nested-Articulable-Subgraphs*:

Procedure*Find-Nested-Articulable-Subgraph*:

Input : the set of articulable subgraphs AS

Output :the set AS of the articulable subgraphs including nested structure

begin

1.**foreach**$AS_i \in AS$

2.N_s ←find the starting node at AS_i

3.**foreach**$N_k$adjacent to N_s (i.e.,$(N_s, N_k) \in E$)

4. $N_k.token \leftarrow 1$

5. $NS, g \leftarrow Find\text{-}Articulable\text{-}Subgraph(AS_i, N_i)$

6.**foreach**$NS_i \in NS$

7.**if**$NS_i \in AS$

8.**break**

9.$AS \leftarrow AS \cup NS$

10. **return**ASas the articulable subgraphs

end.

The above procedure returns the set of all articulable subgraphs and the set of all linearized sequences with abstract nodes along with primitive nodes corresponding to layers in the deep learning model, when it is called as *Find-Nested-Articulable-Subgraph*(G, N_o) where G is the graph model for a deep learning model and $N_o$is the starting node of the each articulable subgraph model.

2.3. ***Identification of Candidate Building Blocks***

One aim of this work is to identify candidate building blocks from the existing source codes of deep learning models. Articulable subgraphs can be candidate building blocks when they satisfy some criteria such as their size and their frequency. If the size of an articulable subgraph, i.e., the number of the subgraph, is greater than the pre-specified threshold, it is chosen as a candidate building block. Although the size of an articulable subgraph is small, it can be a building block if it happens frequently in a deep learning model or across deep learning models.

In order to count the frequency of articulable subgraphs, it needs a method to evaluate the equivalence of two subgraphs. Articulable subgraphs start with a node with fan-in of one and end with a node with fan-out of one. Hence, an articulable subgraph can be a collection of all possible paths from the staring node to the ending nodes. When evaluating the equivalence of two articulable subgraphs, their paths sets are checked to see they are the same set which can be easily performed in the Union-Find data structure[13].

The frequent consecutive subsequences can be candidate building blocks. Such consecutive subsequences are be found by a frequent subsequence mining method which is modified to consider only consecutive subsequences in the mining process. PrefixSpan[14] is one of frequent subsequence mining algorithms which can be used in finding frequent consecutive subsequences. To find frequent articulable subgraphs,

their supports are just counted by comparing the subgraphs with the same size. To find frequent subsequence in the linearized sequence, the subsequences are generated from the end of the sequence at which the corresponding node is an ending node. The ending nodes are either a fully connected layer, a pooling layer, or an RNN (recurrent neural network) layer. From an ending node, consecutive subsequences are generated of which size is three to the pre-specified number. To those subsequences, the above-mentioned modified frequent subsequence mining is applied. For each one of the returned subsequences, their non-overlapped occurrences are counted for the sequence themselves. The frequent consecutive subsequences of layers are recommended as candidate building blocks, where a candidate building block may contain some abstract node(s). Once an abstract node is included in some candidate building block, the corresponding articulable subgraph is also recommended as a building block.

III. RESULTS AND DISCUSSION

3.1. Implementation of the Proposed Method

The proposed method has been implemented to apply the Python codes which use Keras[15] or TensorFlow[16] framework. Figure 1 shows the operations carried out in the proposed method. First, a Python code including a deep learning model is parsed and the model is extracted and exported into a JSON format which is an expression to use a nested list for structured data description. Then the model is converted into a graph model according to the method presented in Section 2.1. From the graph model, the articulable subgraphs and the linearized sequences are found using the procedure Find-Nested-Articulable-Subgraph presented in Section 2.2. For the articulable subgraphs, their size and frequency are checked to determine whether they are recommended as candidate building blocks. For each linearized sequence, the subsequences are generated with respect to ending nodes from right to left presented in Section 2.3. For the collection of the generated subsequences, a modified PrefixSpan algorithm is applied to find the frequent subsequences. For each frequent subsequence, its non-overlapping occurrence is counted for the collection of linearized sequences. The subsequences of non-overlapping count greater than the threshold are recommended as candidate building blocks.

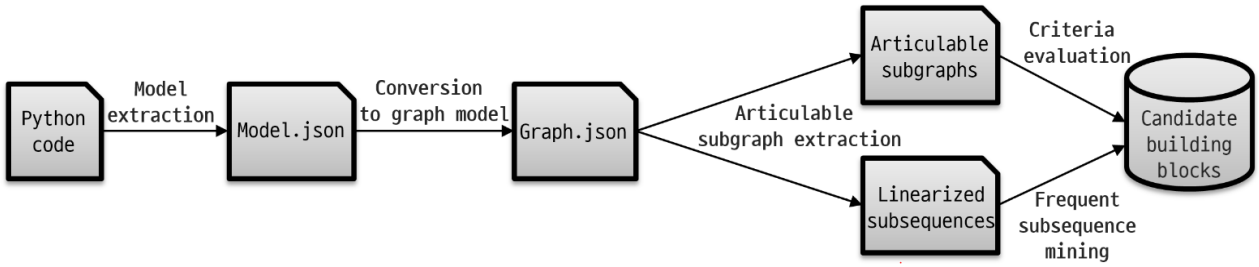

Figure 1. The Operations in the Proposed Method

In Figure 2, (a) shows the GoogleNet model, and(b) shows the articulable subgraphsdetected for the GoogleNet by the proposed method. (c) shows the GoogleNet structure at which the articulable subgraphs are depicted with abstract nodes, and (d) shows its more detained structure at which the articulable subgraphs are drawn as abstract nodes in its nesting abstract node.

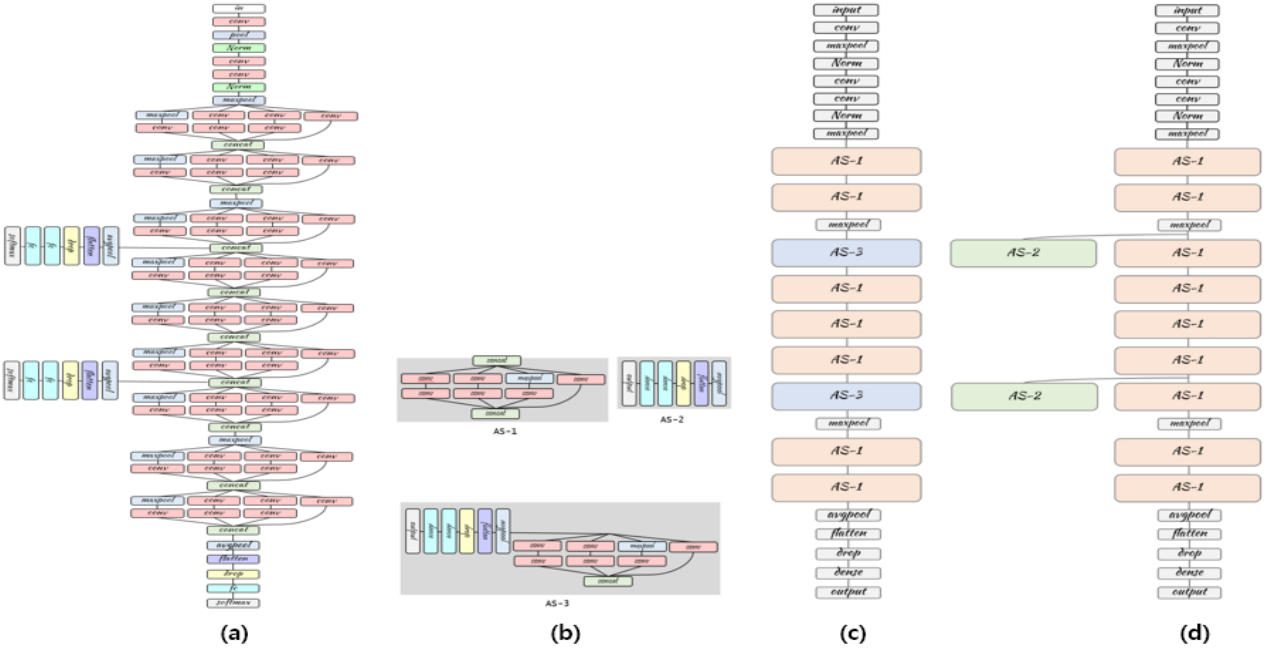

Figure 2. GoogleNet and its articulable subgraphs

The introduction of abstract nodes helps the deep learning models with many layers displayed in a limited

window on the screen. Figure 3 shows a snapshot of the deep learning model editor which implemented the proposed method so that it can import a Python code into itself while identifying the candidate building blocks, and display the imported model in a diagram.

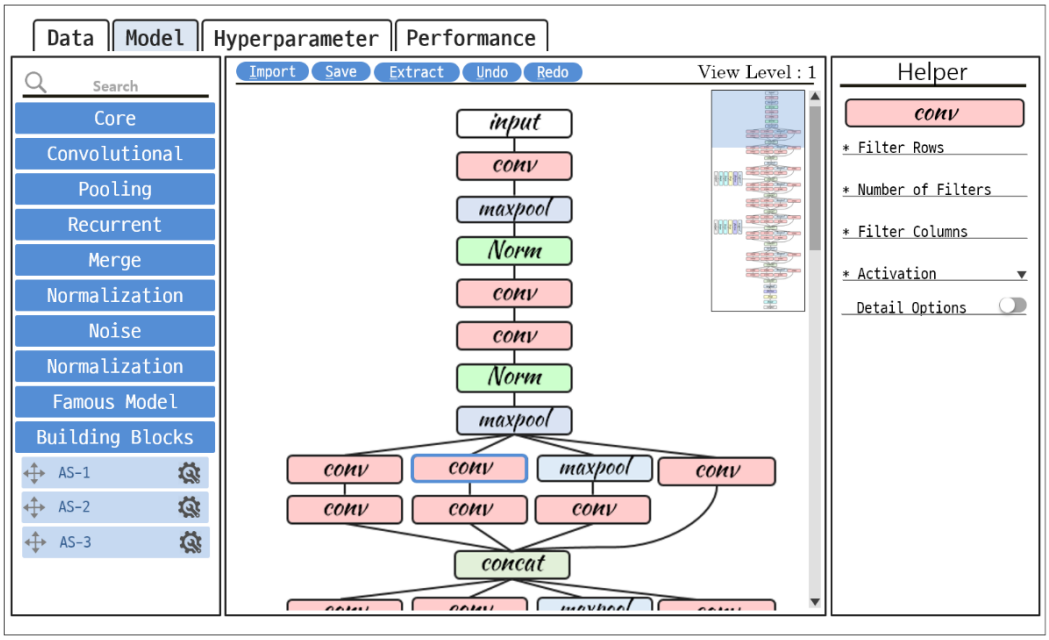

Figure 3. A snapshot of the deep learning modeling tool that implemented the proposed method

IV. CONCLUSION

Deep learning modeling task requires sufficient knowledge on deep learning algorithms and enough practicing experiences. To make a GUI-based deep learning model tool provide useful building blocks so that developers can use them in a drag-and-drop manner, the paper proposed a technique to identify candidate building blocks from existing source codes of deep learning models. It parses and extracts the model architecture from a source code, and converts it into a graph structure. It detects the articulable subgraphs and repeating patterns from the graph structure and identifies the candidate building blocks. The identified building blocks are registered into the GUI-based modeling tool as items in the available building block menu. The building blocks are displayed as an icon and the building block containing nested building blocks can be expanded to show the detailed architecture or can be shrunk to show it in a concise form. A prototype of the GUI-based modeling tool was developed which implemented the proposed method. The identification of building blocks from deep learning model codes enables to provide useful building blocks for non-expert developers to design a deep learning model, and to visualize the imported deep learning models in a hierarchical view which is useful to handle a deep learning model with many layers.

ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (No.: NRF-2017M3C4A7069 432)

REFERENCES

1. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015; 521(7553), 436.
2. Lee H.-W, Kim N.-R., Lee J.-H. Deep Neural Network Self-training Based on Unsupervised Learning and Dropout. International Journal of Fuzzy Logic and Intelligent Systems. 2017;17(1):1-9.
3. Choi H. CNN Output Optimization for More Balanced Classification. International Journal of Fuzzy Logic and Intelligent Systems. 2017;17(2):98-106.
4. Suganuma M, Shirakawa S, NagaoT. A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the Genetic and Evolutionary Computation Conference, ACM. 2017 July; 497-504.
5. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2015; 1-9.
6. He K, Zhang X, Ren S, SunJ. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2016; 770-778.
7. Klemm S, Scherzinger A, Drees D, Jiang X.Barista - a Graphical Tool for Designing and Training Deep Neural Networks.arXiv:1802.04626. 2018;1-8.
8. Suganuma M, Shirakawa S, Nagao T. A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the Genetic and Evolutionary Computation Conference, ACM. 2017 July; 497-504.
9. Sethi A, Sankaran A, Panwar N, Khare S, Mani S. DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers. arXiv preprint arXiv:1711.03543. 2017; 1-8.
10. Hasanpour SH, Rouhani M, Fayyaz M, Sabokrou M, AdeliE. Towards Principled Design of Deep Convolutional Networks: Introducing SimpNet. arXiv preprint arXiv:1802.06205. 2018
11. Albelwi S, MahmoodA. A framework for designing the architectures of deep convolutional neural networks. Entropy. 2017; 242-262.
12. LeeKM, KimKI, YooJ. Autonomicity Levels and Requirements for Automated Machine Learning. In Proceedings of the International Conference on Research in Adaptive and Convergent Systems, ACM. 2017, Sept.; 46-48.
13. CormenTH, Leiserson CE, Rivest RL, SteinC. Introduction to algorithms. MIT press; 2009.

14. PeiJ, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, HsuMC. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings 17th International Conference on Data Engineering. 2001, Apr.; 215-224.
15. Chollet F. Keras: The Python deep learning library. Astrophysics Source Code Library; 2018.
16. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. TensorFlow: a system for large-scale machine learning. arXiv:1605.08695. 2016, Nov; 16:265-283.