Dynamic Depth-Width Optimization for Capsule Graph Convolutional Network

Shangwei Wu[1], Yingtong Xiong[1], Chuliang Weng(✉)[1]

1 School of Data Science and Engineering, East China Normal University, Shanghai 200062, China

A. Main contributions of our work

We propose Dynamic Depth-Width Optimization for Capsule Graph Convolutional Network (DynaCGCN) to achieve a balance between training efficiency and accuracy results. Three main contributions are made in this paper:

- Through observing both the change in validation accuracy and the reduction speed of training loss, we optimize the depth and width of capsule graph convolutional layers across sliding epoch windows. Following this scheme, we can dynamically refine our model while utilizing the weight parameters learned at previous training stages.
- The assistant module is formalized as an RL agent. We define the depth-width setting of capsule graph convolutions as a state and consider the changes in model depth and width as actions. We leverage the RL mechanism to update the rewards corresponding to different actions.
- To ease the computation burden that arises from the assistant module, we split the whole 10-fold cross validation process into multiple tasks to run them synchronously. Experiments imply that the multi-worker paralleling strategy saves much time in finding the optimal depth and width of the model.

Received month dd, 2022; accepted month dd, yyyy

E-mail: clweng@dase.ecnu.edu.cn

B. The effects of different depth-width settings in CapsGNN

To evaluate the effects of the depth and width in CapsGNN, we change them to observe the performance of the model on two graph datasets, ENZYMES and IMDB-MULTI [1], in biochemical and social fields, respectively. It is shown in Figure 1 that if the model has small D and W, it can not sufficiently extract the features of graphs. On the contrary, when the model structure is too complicated, overfitting occurs frequently. Thus, CapsGNN is sensitive to changes in depth and width. Since the impacts of D and W are distinct on different datasets, it is necessary to adaptively adjust the depth-width setting according to graph type.

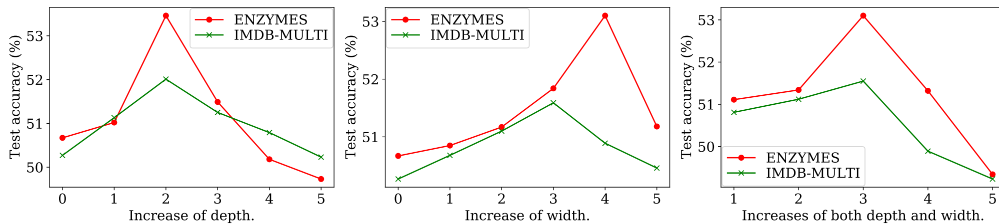


Fig. 1 Results of classification accuracy when increasing D and W. The third subfigure depicts the changes of both D and W, and the x-axis represents the increase of either of them (for simplicity, the growth scales of D and W are set the same here).

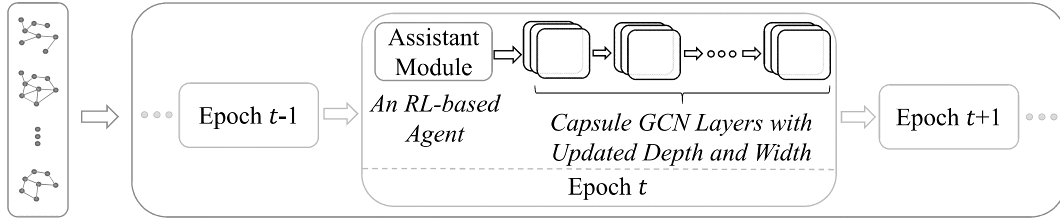


Fig. 2 Architecture of DynaCGCN (non-parallel version).

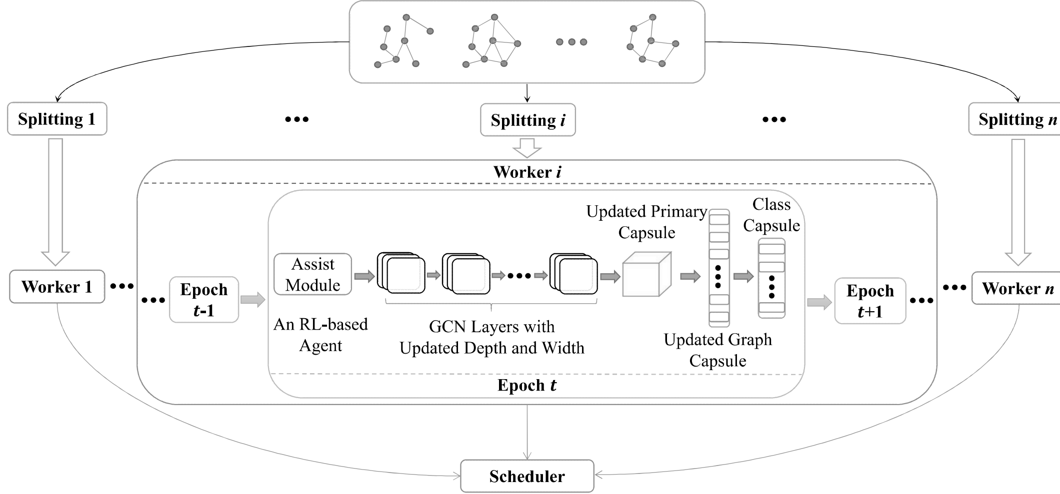


Fig. 3 Architecture of DynaCGCN (parallel version).

C. *Implementation of the assistant module (AM)*

The search process of the optimal D and W is described in Algorithm 1. Note that an action taken in a just-ended sliding window might show a bearable degradation in model performance, which is normal in deep learning. Thus we introduce a discount rate γ and a loss reduction threshold θ to avoid giving a large negative reward for this kind of action. Besides, we also define a scale factor β to normalize the values of $eval(a_{i^*}^{t-3})$ and $\Delta valacc$. $|eval(a_{i^*}^{t-3})|$ is generally larger than $|\Delta valacc|$ in our experiments.

D. *Parallel processing*

To accelerate the selection process in the proposed assistant module, we further assign multiple workers to train in parallel on the GPU. We follow the same validation method as in CapsGNN, i.e., the 10-fold cross validation, which splits the original dataset into ten groups. On each dataset, we take eight groups as the training holds, one group as the validation hold, and the left one group as the test hold. In the end, we would obtain ten test accuracy results on ten different splittings. We take the average of these results as the final test accuracy.

Since the computations on ten splittings are independent of each other, to speed up the 10-fold validation process, we dispatch these splittings to ten workers on one GPU card. Then we dynamically assign an appropriate number of workers by monitoring their running status. We observe that the device memory required by each worker keeps almost the same, so we could estimate the memory requirement of a worker through pre-run.

In Algorithm 2, we describe the strategy for the scheduling of multiple workers. Suppose the maximum number of workers that can run concurrently is K. We launch K workers at the beginning of the training (line 1). Once the running workers are less than K, we select $K-k$ unprocessed workers and launch them on the GPU (lines 2-3). When there are no workers in operation within a time threshold (denoted by T), it means that the training on all workers is over (lines 5-6).

However, when processing multiple workers concurrently on a single dataset, sometimes GPU memory is not fully utilized. Thus we extend Algorithm 2 to do scheduling on multiple datasets, i.e., combinatorially assigning workers on different datasets to maximize the utilization of GPU memory, shown in Algorithm 3.

Algorithm 1: The updating strategy in the assistant module.

Input: The number of training epochs: E, the length of the sliding epoch window: len_{sw} (we let $len_{sw} = 3$ here), the reward discount rate: γ ($0 \leq \gamma \leq 0.5$), the loss decrease threshold: θ ($0 \leq \theta \leq 0.5$), ΔD_list for updating D (we let $\Delta D_list = \{-3, -2, -1, 0, 1, 2, 3\}$ here), the ΔW_list for updating W (we let $\Delta W_list = \{-2, -1, 0, 1, 2\}$ here), the set of losses during E epochs: $\{loss_t\}$, the set of accuracies on the validation data set during E epochs: $\{valacc_t\}$, the set of Q-values of m ($m = len1 \times len2$) actions: $\{Q_i\}$ ($1 \leq i \leq m$).

1. **for** $4 \leq t \leq E$ **do**
2.   **if** $(t-1)\ mod\ 3 = 0$ **then**
3. $\Delta loss_1 \leftarrow loss_{t-2} - loss_{t-1}$, $\Delta loss_2 \leftarrow loss_{t-3} - loss_{t-2}$;
4. $eval(a_{i^*}^{t-3}) \leftarrow \Delta loss_1 / \Delta loss_2$; ▷ loss reduction rate
5. $\Delta valacc \leftarrow valacc_{t-1} - valacc_{t-2}$; ▷ change in validation accuracy
6. $\beta \leftarrow 10$; ▷ a scale factor for normalizing $eval(a_{i^*}^{t-3})$ and $\Delta valacc$
7.   **if** $\Delta valacc > 0$ **then**
8. $flag \leftarrow sgn(\Delta loss_1)$; ▷ $sgn()$ stands for the sign function
9.   **if** $\Delta loss_1 \cdot \Delta loss_2 > 0$ **then**
10.   **if** $eval(a_{i^*}^{t-3}) \geq 1$ **then** ▷ $|\Delta loss_1| \geq |\Delta loss_2|$
11. $Q(a_{i^*}^{t-3}) += flag \cdot (1-\gamma) \cdot (eval(a_{i^*}^{t-3}) + \beta \cdot \Delta valacc)$; ▷ positive or negative reward, depending on whether $\Delta loss_1 > 0$ or not
12.   **else** ▷ $eval(a_{i^*}^{t-3}) \in (0, 1)$, i.e., $|\Delta loss_1| < |\Delta loss_2|$
13.   **if** $\Delta loss_1 < 0$ **then**
14. $Q(a_{i^*}^{t-3}) += \gamma \cdot eval(a_{i^*}^{t-3})$; ▷ minor positive reward due to slight mitigation of overfitting on the training set
15.   **else if** $\Delta loss_1 > \theta$ **then**
16. $Q(a_{i^*}^{t-3}) += \gamma \cdot (\beta \cdot \Delta valacc)$; ▷ minor positive reward due to bearable model performance degradation
17.   **else** ▷ $\Delta loss_1 \in (0, \theta)$
18. $Q(a_{i^*}^{t-3}) -= (1-\gamma) \cdot (\beta \cdot eval(a_{i^*}^{t-3}) + \beta \cdot \Delta valacc)$; ▷ negative reward due to unbearable model performance degradation
19.   **else**
20.   **if** $eval(a_{i^*}^{t-3})) \leq -1$ **then** ▷ $|\Delta loss_1| \geq |\Delta loss_2|$
21. $Q(a_{i^*}^{t-3}) -= flag \cdot (1-\gamma) \cdot (eval(a_{i^*}^{t-3}) + 10\Delta valacc)$; ▷ positive or negative reward, depending on whether $\Delta loss_1 > 0$ or not
22.   **else** ▷ $eval(a_{i^*}^{t-3}) \in (-1, 0)$, i.e., $|\Delta loss_1| < |\Delta loss_2|$
23.   **if** $\Delta loss_1 < 0$ **then**
24. $Q(a_{i^*}^{t-3}) += \gamma \cdot (\beta \cdot \Delta valacc - eval(a_{i^*}^{t-3}))$; ▷ minor positive reward due to bearable model performance degradation
25.   **else**
26. $Q(a_{i^*}^{t-3}) += (1-\gamma) \cdot (\beta \cdot \Delta valacc - \beta \cdot eval(a_{i^*}^{t-3}))$; ▷ positive reward due to the rectification of overfitting on the training set
27.   **else**
28. $Q(a_{i^*}^{t-3}) += (1-\gamma) \cdot (\beta^2 \cdot Min(\Delta valacc, 0))$; ▷ strong negative reward
29. $Q_maxindex \leftarrow$ index of $Max\{Q_i\}$, $action_maxindex \leftarrow actions[Q_maxindex]$;
30.   return $\Delta D_list[p^*] = action_maxindex[0]$, $\Delta W_list[q^*] = action_maxindex[1]$;

Algorithm 2: Scheduling within a dataset.

Input: Number of running workers: k, GPU memory allocated to each worker: M_s, available GPU memory: M_a, number of maximal concurrent workers: $K = \lfloor M_a/M_s \rfloor$.

```
1  Launch K workers and initialize k as K;
2  while k < K do
3      Launch K − k unexecuted workers;
4      Update k;
5      if k = 0 then
6          Break;
```

Algorithm 3: Scheduling among multiple datasets.

Input: The list of datasets: $\{ds_i \mid i \in [1, len_{ds}]\}$ ($len_{ds} \geq 2$), GPU memory allocated to each worker on ds_i: M_i, available GPU memory: M_a.

```
1  for i in [1, len_ds] do
       ▷ schedule on ds_i
2      if training on ds_i not begins then
3          U_i ← number of unexecuted splittings in ds_i;
4          K_i ← min(⌊M_a/M_i⌋, U_i);
5          Initialize K_i workers on ds_i;
       ▷ schedule on ds_j
6      for j in [i, len_ds] do
7          if training on ds_j not begins then
8              if M_a − K_i × M_i ≥ M_j then
9                  U_j ← number of unexecuted splittings
                    in ds_j;
10                 K_j ← min(⌊(M_a − K_i × M_i)/M_j⌋, U_j);
11                 Initialize K_j workers in ds_j;
12 Repeat lines 1 to 11, and schedule on ds_i and ds_j with
    Algorithm 2 until all splittings from all datasets are
    executed.
```

E. DynaCGCN architecture

The overall design of DynaCGCN is depicted in Figure 2 and Figure 3.

F. Environmental settings

Since our work focuses on graph-level classification (not node-level classification or link-level prediction), we experiment on the datasets frequently used in graph classification tasks including four biological datasets (MUTAG, ENZYMES, NCI1, and PROTEINS) and three social network datasets (COLLAB, IMDB-BINARY, and IMDB-MULTI). The details of these datasets are described in Table 1.

DynaCGCN is compared with four graph kernel algorithms (GK [2], WL [3], DGK [4], and AWE [5]), four GNNs-based methods (PATCHY-SAN (PSCN) [6], DGCNN [7], GIN [8], and SOM-GCNN [9]), and two capsule-based GNNs methods (GCAPS-CNN [10] and CapsGNN [11]). We also take the results reported in [12] as a baseline, namely FGNN, which proposed a fair validation method of GNNs for graph classification.

We evaluate DynaCGCN on a machine equipped with dual 2.40 GHz Intel Xeon Gold 6240R processors (24 cores in total), 256 GB main memory, and 1 NVIDIA Tesla V100 GPU (32 GB memory). The installed operating system is Ubuntu 18.04, using CUDA 11.2 and cuDNN 7.6.5. PyTorch 1.8 [13] and Python 3.6.5 are used for training.

G. Search process of the optimal depth-width settings on different datasets

In DynaCGCN, unlike the static depth-width setting used in CapsGNN, the ΔD and ΔW selected by the assistant module during training are displayed in Table 2. Note that the optimal ΔD and ΔW determined by AM are different on the 10 splittings. We take the ΔD and ΔW on the splitting with the best test accuracy for the comparison against CapsGNN.

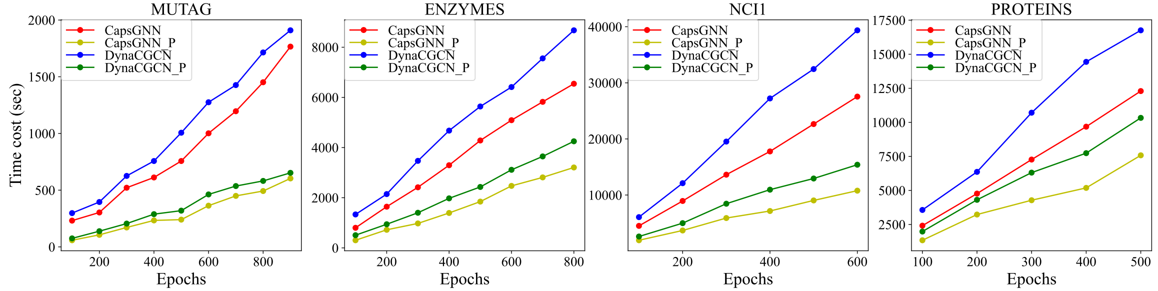
H. Parallel execution

Taking into consideration the computational cost brought by the assistant module, we adaptively adjust the number of workers at runtime. It implies in Figure 4 that when with Algorithm 2, the parallel DynaCGCN on all datasets could averagely speed up the sequential DynaCGCN by up to 2.99×. To make full use of GPU memory, we adopt Algorithm 3 to let workers on different datasets run combinatorially, which further decreases the time overhead, e.g., by 16.7% within 300 training epochs.

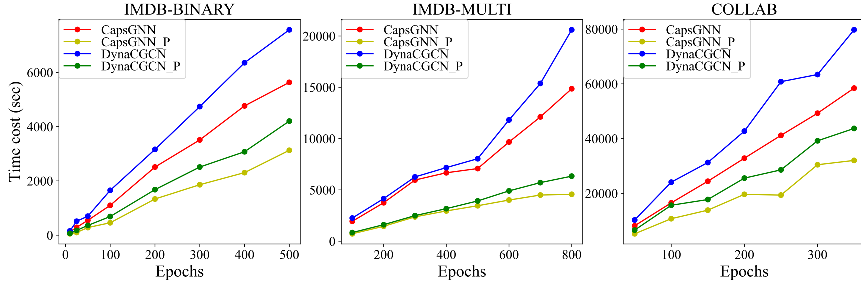
The comparison of memory consumption when experimenting with CapsGNN and our model is recorded in Table 3. It indicates that the assistant module in DynaCGCN requires additional device memory to do training. However, using the parallel strategies proposed in Algorithm 2 and Algorithm 3, DynaCGCN could achieve a good balance between computation efficiency and model accuracy with full utilization of

Table 1 Dataset Information [1].

Description	MUTAG	ENZYMES	NCI1	PROTEINS	IMDB-B	IMDB-M	COLLAB
Type	Bio	Bio	Bio	Bio	Social	Social	Social
#Graphs	188	600	4110	1113	1000	1500	5000
#Graph classes	2	6	2	2	2	3	3
#Nodes Avg.	17.93	32.46	29.87	39.06	19.77	13	74.49
#Edges Avg.	19.79	63.14	32.30	72.81	193.06	131.87	4914.99
#Node Labels	7	6	23	4	-	-	-



(a) Biochemical Datasets.



(b) Social Datasets.

Fig. 4 Time costs of CapsGNN, DynaCGCN, and their parallel versions, i.e., CapsGNN_P and DynaCGCN_P.

Table 2 Comparison of $(\Delta D, \Delta W)$ selected in CapsGNN and DynaCGCN.

Dataset	CapsGNN	DynaCGCN
MUTAG	$(0,0)$	$(-1,0) \rightarrow (1,0) \rightarrow (0,1) \rightarrow (1,-1) \rightarrow (1,1) \rightarrow (0,1)$
ENZYMES	$(0,0)$	$(-3,1) \rightarrow (1,1) \rightarrow (1,1) \rightarrow (1,-1) \rightarrow (1,1) \rightarrow (1,0)$
NCI1	$(0,0)$	$(-2,-1) \rightarrow (1,1) \rightarrow (0,1) \rightarrow (1,0) \rightarrow (1,0) \rightarrow (0,1)$
PROTEINS	$(0,0)$	$(-3,0) \rightarrow (1,2) \rightarrow (0,0) \rightarrow (1,-1) \rightarrow (1,1) \rightarrow (1,-1)$
IMDB-BINARY	$(0,0)$	$(0,0) \rightarrow (0,1) \rightarrow (1,0) \rightarrow (0,-1) \rightarrow (-1,0) \rightarrow (1,-1)$
IMDB-MULTI	$(0,0)$	$(-2,0) \rightarrow (1,1) \rightarrow (1,-1) \rightarrow (0,1) \rightarrow (1,0) \rightarrow (1,-1)$
COLLAB	$(0,0)$	$(-1,0) \rightarrow (1,2) \rightarrow (0,-1) \rightarrow (1,1) \rightarrow (0,-1) \rightarrow (0,1)$

GPU memory.

I. Difference between DynaCGCN and one-shot NAS methods

Prevalent one-shot NAS approaches [14, 15] usually train a supernet that contains all possible architectures in the search space. The performance of a particular architecture can be approximately evaluated without training, i.e., by only preserving the corresponding weights in the supernet and using them in the inference phase. However, the performance of a certain architecture predicted by the supernet is often inaccurate [16].

The main difference between our work and existing one-shot NAS methods is that we reduce the search cost by training only one particular architecture (rather than a huge supernet) in an arbitrary training stage. Besides, we leverage the RL idea to record the accumulated effect of each chosen architecture (w.r.t. model depth and width) by monitoring not only the loss reduction rate but also the accuracy changes on the validation set, which helps to obtain an efficient and accurate search.

J. Related work

Following the success of neural networks on grid data [17, 18], considerable research interest has been devoted to non-grid graph data, i.e., Graph Neural Networks (GNNs) [19],

Table 3 Memory consumption of CapsGNN and DynaCGCN.

Model	MUTAG	ENZYMES	NCI1	PROTEINS	IMDB-BINARY	IMDB-MULTI	COLLAB
CapsGNN	1521Mb	2279Mb	2695Mb	6135Mb	2501Mb	1975Mb	8111Mb
DynaCGCN	1990Mb	3162Mb	5526Mb	11120Mb	4692Mb	4730Mb	10572Mb

especially Graph Convolutional Networks (GCNs) [20, 21]. GNNs have already obtained remarkable achievements in various tasks [22], e.g., graph classification [23], link prediction [24], and node classification [25]. GCNs inherits the convolutional operations in CNNs, however, the simple aggregation in convolutions could not help learn sufficient information about graphs containing multiple node attributes and complicated inter-node connections.

CapsNet was proposed by Hinton's team [26] and improved by them [27, 28] to represent local-global features of images. Inspired by the promising explainability of CapsNet, some studies [10, 11, 29] combine GCNs and CapsNet to extract multi-scale information in graphs. However, the static structure employed in these capsule-based GCNs would restrict their representation ability, which motivates us to explore a dynamic adjustment of the model structure during training.

Concerning the design of the structure of a neural network, it has been turning from manual efforts into automatic machine searches, known as Neural Architecture Search (NAS) [14, 16, 30–32]. There exist mainly three kinds of techniques to search for the best model architecture in NAS, i.e., Reinforcement Learning (RL) based, Evolution Algorithm (EA) based, and gradient-based methods. RL-based methods [14, 30, 33] use RNN as the controller to train different architectures and select the optimal one according to their validation accuracy results. EA-based methods [32, 34–36] utilize the evolutionary algorithm to search for superior model structures from a set of initialized candidate networks. Gradient-based methods [15, 37, 38] construct a SuperNet and leverage the attention mechanism to remove weak connections after searching. The former two methods can be regarded as offline and are often time-consuming. Although the third method is an online method and shows good efficiency due to gradient descent optimization, it lacks variety in searching for the optimal architecture. Thus, it is necessary to make a good trade-off between search efficiency and model performance.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 62141214 and 62272171).

References

1. Kersting K, Kriege N M, Morris C, Mutzel P, Neumann M. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de
2. Shervashidze N, Vishwanathan S, Petri T, Mehlhorn K, Borgwardt K. Efficient graphlet kernels for large graph comparison. In: International Conference on Artificial Intelligence and Statistics. 2009, 488–495
3. Shervashidze N, Schweitzer P, Leeuwen E J V, Mehlhorn K, Borgwardt K M. Weisfeiler-Lehman graph kernels. Journal of Machine Learning Research, 2011, 12(9): 2539–2561
4. Yanardag P, Vishwanathan S. Deep graph kernels. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2015, 1365–1374
5. Ivanov S, Burnaev E. Anonymous walk embeddings. In: International Conference on Machine Learning. 2018, 2191–2200
6. Niepert M, Ahmed M, Kutzkov K. Learning convolutional neural networks for graphs. In: International Conference on Machine Learning. 2016, 2014–2023
7. Zhang M, Cui Z, Neumann M, Chen Y. An end-to-end deep learning architecture for graph classification. In: Proceedings of the Thirty-Second Artificial Intelligence AAAI Conference. 2018, 4438–4445
8. Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? In: International Conference on Learning Representations. 2019
9. Pasa L, Navarin N, Sperduti A. SOM-based aggregation for graph convolutional neural networks. Neural Computing and Applications, 2020, 1–20
10. Verma S, Zhang Z L. Graph capsule convolutional neural networks. In: Joint ICML and IJCAI Workshop on Computational Biology. 2018
11. Zhang X, Chen L. Capsule graph neural network. In: International Conference on Learning Representations. 2019
12. Errica F, Podda M, Bacciu D, Micheli A. A fair comparison of graph neural networks for graph classification. In: International Conference on Learning Representations. 2020
13. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. 2019, 8024–8035
14. Pham H, Guan M Y, Zoph B, Le Q V, Dean J. Efficient neural architecture search via parameter sharing. In: International Conference on Machine Learning. 2018, 4092–4101

15. Liu H, Simonyan K, Yang Y. DARTS: Differentiable architecture search. In: International Conference on Learning Representations. 2019
16. Luo R, Tian F, Qin T, Chen E, Liu T. Neural architecture optimization. In: Advances in Neural Information Processing Systems. 2018, 7827–7838
17. LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series. In: The Handbook of Brain Theory and Neural Networks. Cambridge, MA, USA: MIT Press, 1998, 255–258
18. Mikolov T, Kombrink S, Burget L, Černockỳ J, Khudanpur S. Extensions of recurrent neural network language model. In: International Conference on Acoustics, Speech and Signal Processing. 2011, 5528–5531
19. Gori M, Monfardini G, Scarselli F. A new model for learning in graph domains. In: IEEE International Joint Conference on Neural Networks. 2005, 729–734
20. Bruna J, Zaremba W, Szlam A, Lecun Y. Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations. 2014
21. Henaff M, Bruna J, LeCun Y. Deep convolutional networks on graph-structured data. ArXiv, 2015, abs/1506.05163
22. Abadal S, Jain A, Guirado R, López-Alonso J, Alarcón E. Computing graph neural networks: A survey from algorithms to accelerators. ACM Computing Surveys, 2022, 54(9): 191:1–191:38
23. Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems. 2016, 3844–3852
24. Zhang M, Chen Y. Link prediction based on graph neural networks. In: Advances in Neural Information Processing Systems. 2018, 5165–5175
25. Kipf T, Welling M. Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations. 2017
26. Hinton G E, Krizhevsky A, Wang S D. Transforming auto-encoders. In: International Conference on Artificial Neural Networks. 2011, 44–51
27. Sabour S, Frosst N, Hinton G E. Dynamic routing between capsules. In: Advances in Neural Information Processing Systems. 2017, 3856–3866
28. Hinton G E, Sabour S, Frosst N. Matrix capsules with EM routing. In: International Conference on Learning Representations. 2018
29. Mallea M D G, Meltzer P, Bentley P J. Capsule neural networks for graph classification using explicit tensorial graph representations. ArXiv, 2019, abs/1902.08399
30. Zoph B, Le Q V. Neural architecture search with reinforcement learning. In: International Conference on Learning Representations. 2017
31. Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L, Fei-Fei L, Yuille A L, Huang J, Murphy K. Progressive neural architecture search. In: European Conference on Computer Vision. 2018, 19–35
32. Real E, Aggarwal A, Huang Y, Le Q V. Regularized evolution for image classifier architecture search. In: Proceedings of the Thirty-First Artificial Intelligence AAAI Conference. 2019, 4780–4789
33. Zoph B, Vasudevan V, Shlens J, Le Q V. Learning transferable architectures for scalable image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. 2018, 8697–8710
34. Real E, Moore S, Selle A, Saxena S, Suematsu Y L, Tan J, Le Q V, Kurakin A. Large-scale evolution of image classifiers. In: International Conference on Machine Learning. 2017, 2902–2911
35. Xie L, Yuille A L. Genetic CNN. In: International Conference on Computer Vision. 2017, 1388–1397
36. Shu H, Wang Y, Jia X, Han K, Chen H, Xu C, Tian Q, Xu C. Co-Evolutionary compression for unpaired image translation. In: International Conference on Computer Vision. 2019, 3234–3243
37. Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: IEEE Conference on Computer Vision and Pattern Recognition. 2019, 10734–10742
38. Xie S, Zheng H, Liu C, Lin L. SNAS: Stochastic neural architecture search. In: International Conference on Learning Representations. 2019