GRAPH AUTOENCODER FOR GRAPH COMPRESSION AND REPRESENTATION LEARNING

Yunhao Ge,* **Yunkui Pang,*** **Linwei Li & Laurent Itti**
Department of Computer Science
University of Southern California
Los Angeles, CA
{yunhaoge,yunkuipa,lli06371,itti}@usc.edu

ABSTRACT

We consider the problem of graph data compression and representation. Recent developments in graph neural networks (GNNs) focus on generalizing convolutional neural networks (CNNs) to graph data, which includes redesigning convolution and pooling operations for graphs. However, few methods focus on effective graph compression to obtain a smaller graph, which can reconstruct the original full graph with less storage and can provide useful latent representations to improve downstream task performance. To fill this gap, we propose Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE), which, instead of compressing nodes/edges separately, utilizes the node similarity and graph structure to compress all nodes and edges as a whole. Similarity attention graph pooling selects the most representative nodes with the most information by using the similarity and topology among nodes. Our multi-kernel Inductive-Convolution layer can focus on different aspects and learn more general node representations in evolving graphs. We demonstrate that MIAGAE outperforms state-of-the-art methods for graph compression and few-shot graph classification, with superior graph representation learning. (The code is released at this URL†)

1 INTRODUCTION AND RELATED WORKS

Graph Neural Networks (GNNs) (Scarselli et al., 2009; Kipf & Welling, 2016a; Gilmer et al., 2017) extend the idea of using convolution on grid-structure data (images) in Convolutional neural networks (CNNs) (LeCun et al., 2012) to graph data. They achieve great success on many tasks, such as node classification and link prediction (Veličković et al., 2017; Hamilton et al., 2017a). The size of graphs presents a big obstacle to understanding the essential information they contain. Graph compression (Navlakha et al., 2008) can solve this problem and be used to improve visualization, to understand the graph's high-level structure, or as a pre-processing step for other data mining methods (Zhou, 2015).

However, few methods focus on graph compression and reconstruction with GNNs. Although CNN autoencoders are an efficient compression framework for images, research on Graph autoencoder (GAE) structure methods is rare. Variational Graph Autoencoder (Kipf & Welling, 2016b) focuses on

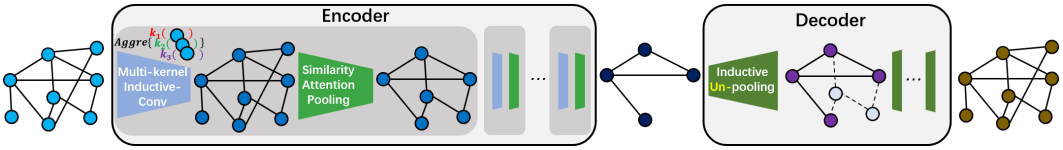

Figure 1: Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE), which consists of an encoder E and a decoder D. E consists of pairs (gray) of Multi-kernel Inductive Graph convolution layer and Similarity Attention Graph Pooling layer. D consists of inductive Un-pooling layers.

*Yunhao Ge and Yunkui Pang contributed equally to this work.
†The code is released at https://github.com/Pangyk/Graph_AE.

link representation while it cannot obtain a compressed graph. Graph Unet (Gao & Ji, 2019) achieves a graph autoencoder structure with gPool but needs extra skip feature connections between encoder and decoder, which have high storage cost. Here, we fill this gap and propose Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE), a GAE structure with GNN for graph compression and graph representation learning (Hamilton et al., 2017b; Goyal & Ferrara, 2018) (Fig. 1). MIAGAE can be used in graph compression and latent graph inference, high-level structure mining, and other unsupervised learning or self-supervised learning (Table 1). The biggest challenge for GAE design is graph pooling (Dhillon et al., 2007; Defferrard et al., 2016). DiffPool (Ying et al., 2019) is a differentiable pooling method by learning an assignment matrix. SAGPool (Lee et al., 2019) uses self-attention to achieve SOTA performance on various tasks. However, these pooling methods need extra trainable parameters and are not designed for graph compression tasks.

Our contributions are: We propose Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE), with (1) Similarity Attention Graph Pooling (SimAGPool) keeps representative nodes with a high contribution for graph compression and needs no extra trainable parameter. (2) Multi-kernel Inductive Graph convolution layer (MI-Conv) to suitable for learning in evolving graphs; (3) State-of-the-art graph compression and representation learning performance, with best storage-efficient; (4) A new image to scene graph pipeline* and synthesized scene graph dataset, Img2SceneGraph-ACSG, which allow rapid idea prototyping for graph compression and representation learning.

Table 1: Comparision of CNN-based Image autoencoder and GCN-based graph autoencoder

	CNN Autoencoder	Variational Graph AE	Graph Unet	MIAGAE (Ours)
Compressed latent inference	✔	✗	✔	✔
Multi-kernel	✔	✗	✗	✔
Pooling layer	✔	✗	✔	✔
Un-Pooling layer	✔	✗	✔	✔
Skip connections between E and D **not** needed	✔	✔	✗	✔
Compatible with graph data	✗	✔	✔	✔

2 Multi-Kernel Inductive Attention Graph Autoencoder

2.1 Multi-kernel Inductive Convolutional layer

Our MIAGAE consists of an encoder network E and a decoder network D (Fig. 1). To obtain graph compression and reconstruction ability, E first learns to eliminate graph nodes and edges to get a compressed smaller graph, then D learns to add new nodes and reconstruct the original graph. Since, during learning, compressed graphs assume dynamic node and edge numbers, we use inductive GCN layers (Hamilton et al., 2017a) instead of transductive layers (Kipf & Welling, 2016a). On the one hand, the node representations are influenced when adding and eliminating nodes, which makes it hard to learn individual representations for each node; on the other hand, learning an inductive method to summarize the representation of new nodes by aggregating their neighbors' feature is more general and easier to generalize across graphs and even datasets (Sec. 3.2).

Our single-kernel Inductive-Conv layer is inspired by GraphSAGE (Hamilton et al., 2017a), which is:

$$f_{k+1}^{i} = W_1 f_k^i + W_2 \cdot mean_{j \in \mathcal{N}(i)} f_k^j \tag{1}$$

where $mean$ is the aggregation function, f_k^i denotes the feature of node v_i in layer k, W_1 and W_2 denote the shared linear transformation parameter for center node v_i and neighbor node v_j respectively, $\mathcal{N}(i)$ denotes the neighboring node sets connected to node i. Inspired by CNN layers in the image domain, which have multiple convolution kernels, and each kernel may extract different aspects of a feature, we propose the Multi-kernel Inductive-Conv layer (MI-Conv). Instead of using a single set of transformation weights W_1 and W_2, MI-Conv uses multiple kernels, and each kernel has its corresponding W_1 and W_2. Features extracted by the first kernel and final features aggregating all m kernels become:

$$f_{k+1,1}^{i} = W_1^1 f_k^i + W_2^1 \cdot mean_{j \in \mathcal{N}(i)} f_k^j \tag{2}$$

$$f_{k+1}^{i} = Aggre(\sigma(f_{k+1,1}^{i}), \sigma(f_{k+1,2}^{i}), ..., \sigma(f_{k+1,m}^{i})) \tag{3}$$

*http://ilab.usc.edu/datasets/i2sg

where m is the number of kernels; σ represents a non-linear activation function (we use ReLU); $Aggre$ represents an aggregate function to combine the multi-kernel results (we use addition).

2.2 Similarity Attention Graph Pooling

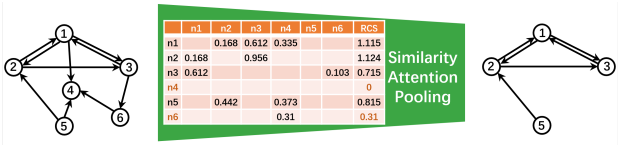

Figure 2: Similarity Attention Graph Pooling.

We propose a Similarity Attention Graph Pooling (SimAGPool) layer (Fig. 2) to achieve down-sampling on graph data and to adaptively select a subset of nodes/edges to form a new but smaller graph through E. To this end, we define Representativeness and Contribution Score (RCS) to help SimAGPool select the most representative (higher RCS) nodes which contain the majority of the graph's information, while eliminating the less representative (lower RCS) nodes and their corresponding edges. RCS is related to both the similarity between node features and the structure of the graph. To calculate RCS, we first calculate the inter product $v_i^T v_j$ as similarity score, which captures the similarity of node features between nodes n_i and n_j. For each node n_i, the value of RCS is the sum of all similarity scores between n_i and its neighbor nodes n_j, if there exist a directed edge e_{ij} form n_i to n_j.

$$RCS_{n_i} = \sum_{j \in \mathcal{N}(i)} (v_i^T v_j) \qquad s.t. \quad \exists e_{ij} \tag{4}$$

The intuition behind SimAGPool and RCS is to keep the nodes with majority graph information, according to two properties: (1) More representative: higher RCS means node n_i is more similar to all the neighbors which represent the majority feature representation in a local subgraph. (2) More contribution: the higher the RCS that n_i has, the more information n_i contributes to its neighbor nodes during message passing in MI-Conv through directed edges e_{ij}. The pooling ratio $p \in (0; 1]$ is a hyperparameter that determines the number of nodes to keep (keep top p percent of nodes by RCS).

Un-Pooling layer forms D, aiming to reconstruct the original graph given the connection information of missing nodes. D and E have a symmetric structure where the number of Inductive Un-pooling layers is the same as the number of SimAGPool layers. The inductive Un-pooling layer has two inputs: the output graph from the previous layer and the edges information of new nodes, which is the same as the eliminated edges in the corresponding pooling layer in E. The inductive Un-pooling layer has the same parameter settings as the Inductive-Conv layer. (More architecture details in Appendix).

3 Experiments

We evaluate our SimAGPool and MI-Conv layers based on the MIAGAE proposed in Sec. 2 on two tasks: (1) Graph compression and reconstruction; (2) Few-shot graph classification to test graph representation learning performance. We compare our networks with previous state-of-the-art GAE models and graph pooling methods that GAE models can use. Table. 2 summarizes the datasets used for envaluation: **COLORS-3** (Knyazev et al., 2019), **FRANKENSTEIN** (Orsini et al., 2015), **Visual Genome** (kri) and **Img2SceneGraph-ACSG** which is a dataset we contribute. (More details in Appendix).

Table 2: Dataset information

	COLORS-3	FRANKENSTEIN	Visual Genome	Img2SceneGraph-ACSG
Number of graphs	10,500	4377	10,000	7,000
Number of classes	11	2	No label	80
Node feature dimension	5	780	500	500

3.1 Graph compression and reconstruction

We use the graphs from all four datasets in Table. 2 to evaluate graph compression and reconstruction performance. Table. 3 compares our MIAGAE and five baseline methods on three main dimensions:

Compression ratio represents the ratio between the uncompressed size and compressed size. A higher ratio represents a more powerful compression ability. MSE represents the reconstruction mean square error between the original graph and the reconstructed graph by GAE. A smaller value means less information loss during compression. Storage represents the storage size that reconstruction needs (except for the latent space, which is the same for all methods), including the decoder network's size and extra information (skip connection features, edge information); smaller is better. Compared with the baseline method Graph Unet under the same compression ratio setting, we obtain better reconstruction MSE while using much less storage for reconstruction, likely mostly because we do not need skip connection feature communication between E and D. Compared with the other four baselines (which can be treated as ablation studies), our MIAGAE outperforms all of them in terms of requiring the smallest storage; On MSE, we perform better than gPool-GAE, GCN-GAE, and GAT-GAE on all four datasets. For SAGPool-GAE, we have smaller MSE on most datasets except COLORS-3 (0.026 vs. 0.028). (More training and architecture details in Appendix).

Table 3: Graph compression performance comparison on test set. We have seven total methods (including five baselines): the first one is graph Unet (Gao & Ji, 2019), the following two are graph pooling methods gPool (Gao & Ji, 2019) and SAGPool (Lee et al., 2019), we use them to substitute our SimAGPool in MIAGAE and form gPool-GAE and SAGPool-GAE, the following two are graph convolution layers GCN (Kipf & Welling, 2016a) and GAT (Veličković et al., 2017), we use them to substitute our MI-Conv and form GCN-GAE and GAT-GAE. The last two are our methods MIAGAE with single and two kernels. Note: we can treat the last four baselines as an ablation study.

Dataset	FRANKENSTEIN		Visual Genome		Img2SceneGraph-ACSG		COLORS-3	
Compression ratio	21.6 : 1		14.86 : 1		14.86 : 1		1.9 : 1	
Attribute	**M**SE	**S**torage	**M**	**S**	**M**	**S**	**M**	**S**
Graph Unet (Gao & Ji, 2019)	0.028	62.81 Mb	0.86	583.28Mb	0.85	238.88 Mb	0.058	646.55Mb
gPool-GAE (Gao & Ji, 2019)	0.021	14.90Mb	0.71	280.86Mb	0.71	112.18Mb	0.031	287.35Mb
SAGPool-GAE (Lee et al., 2019)	0.018	7.12Mb	0.54	124.30 Mb	0.51	54.36 Mb	**0.026**	64.85Mb
GCN-GAE (Kipf & Welling, 2016a)	0.046	8.13Mb	0.92	140.56Mb	0.92	48.87Mb	0.096	136.80Mb
GAT-GAE (Veličković et al., 2017)	0.037	5.50Mb	0.81	126.67Mb	0.81	48.70Mb	0.108	91.46Mb
MIAGAE 1 kernel (Ours)	**0.016**	**5.32 Mb**	0.53	**93.72 Mb**	0.50	**43.73Mb**	0.029	**42.16Mb**
MIAGAE 2 kernel (Ours)	**0.016**	5.77 Mb	**0.51**	139.72 Mb	**0.48**	58.99Mb	0.028	55.65Mb

3.2 Graph representation learning and Few-shot Graph classification

In this section, we evaluate graph representation learning performance. GAE structure networks can be used as an unsupervised or self-supervised learning framework to learn useful graph representations on an unlabeled dataset, and then transfer the representation knowledge to downstream tasks. We use a few-shot graph classification task as a downstream task which is a challenging task with only a small amount of labeled data. Especially when the graph size to be classified is large, a conflict occurs because the large graph sample requires a large classifier with more parameters. At the same time, limited labeled data can not support the training. So a compressed latent graph representation may alleviate this conflict.

We first randomly divide the FRANKENSTEIN dataset into two datasets: unlabeled dataset U to pretrain the graph compression model and labeled dataset L for a few-shot graph classification task (L consists of 100 training and 100 test graphs). After training the graph compression models with U, we use them to compress the graphs in L, and then train a GCN classifier on 100 training graphs and test it on 100 test graphs. One of our baselines is directly using the original training graphs of L to train a GCN classifier without compression. The other baselines have the same settings as Sec. 3.1. Table. 4 shows classification accuracy for the downstream GCNs. (More training details in Appendix). Our method outperforms all baselines.

Generalizing across datasets. To evaluate the graph representation learning performance across datasets, we pretrain the graph compression model on an unlabeled scene graph dataset (Visual Genome), and then directly use the pretrained graph compression model to compress the graphs in our contributed Img2SceneGraph-ACSG (where we use 1000 graphs for training and 1000 for test). Table. 5 shows classification accuracy for the downstream GCNs. (More training details in Appendix). Our method outperforms all baselines.

Table 4: Few shot graph classification performance on FRANKENSTEIN dataset. "No compression" classifier cannot use unlabeled data U, hence marked N/A.

Compression method	GCN classifier parameters	Unlabeled pretrain dataset size	Training set	Test set	Accuracy (2 classes)
No compression	52.1k	N/A	100	100	62 %
Graph Unet	2.1k	4000	100	100	53 %
gPool GAE	2.1k	4000	100	100	57 %
SAGPool GAE	2.1k	4000	100	100	51 %
MIAGAE (Ours)	2.1k	4000	100	100	**65 %**

Table 5: Few shot graph classification performance on AI scene graph dataset with transfer learning

Compression method	GCN classifier parameters	Unlabeled pretrain dataset size	Training set	Test set	Accuracy (80 classes)
No compression	56.9k	N/A	1000	1000	11 %
Graph Unet	6.3k	5000	1000	1000	5 %
gPool GAE	6.3k	5000	1000	1000	16 %
SAGPool GAE	6.3k	5000	1000	1000	19 %
MIAGAE (Ours)	6.3k	5000	1000	1000	**25 %**

4 CONCLUSION

We proposed a novel Graph Autoencoder structure, Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE), which achieves state-of-the-art performance on graph compression and representation learning tasks. The proposed pooling method, SimAGPool, uses nodes similarity and topology to select the most representative nodes with no extra trainable parameters. Multi-kernel Inductive-Conv helps to learn general node representation in an autoencoder structure. We show that MIAGAE can be a general graph representation learning framework for downstream tasks. We hope that researchers find our Graph autoencoder useful and extend it for their applications.

ACKNOWLEDGMENTS

This work was supported by C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), the Army Research Office (W911NF2020053), and the Intel and CISCO Corporations. The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

REFERENCES

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11): 1944–1957, 2007.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

KOUASSI Konan Jean-Claude. Ai-challenger-scene-classification dataset. `https://www.kaggle.com/kjeanclaude/ai-challenger-scene-classification`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Boris Knyazev, Graham W Taylor, and Mohamed R Amer. Understanding attention and generalization in graph neural networks. *arXiv preprint arXiv:1905.02850*, 2019.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pp. 3734–3743. PMLR, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 419–432, 2008.

Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence*, volume 2015, pp. 3756–3762. IJCAI-INT JOINT CONF ARTIF INTELL, 2015.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Kaihua Tang. A scene graph generation codebase in pytorch, 2020. `https://github.com/KaihuaTang/Scene-Graph-Benchmark.pytorch`.

Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. Learning to compose dynamic tree structures for visual contexts. In *Conference on Computer Vision and Pattern Recognition*, 2019.

Kaihua Tang, Yulei Niu, Jianqiang Huang, Jiaxin Shi, and Hanwang Zhang. Unbiased scene graph generation from biased training. In *Conference on Computer Vision and Pattern Recognition*, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling, 2019.

Fang Zhou. Graph compression. *Department of Computer Science and Helsinki Institute for Information Technology HIIT*, pp. 1–12, 2015.

APPENDIX

A DATASET

A.1 IMAGE TO SCENE GRAPH PIPELINE

Img2SceneGraph provides a pipeline that transfers images to scene graphs with node attributes. Here is a typical workflow:

1. For each image, we used the pre-trained model from Scene-Graph-Benchmark (Tang, 2020; Tang et al., 2019; 2020) to synthesis the following outputs: 79 bounding boxes (b-boxes) labeled by a single word and over 6,000 relationship pairs (rel-pairs) between b-boxes, both of them are sorted by their corresponding confidence scores.
2. To form a scene graph, we provide multiple methods to select edges and nodes.
   Select edges first
   (a) Select the top $n\%$ rel-pairs as edges and corresponding b-boxes as nodes.
   (b) Select rel-pairs with confidence score higher than k as edges and corresponding b-boxes as nodes.
   (c) Select top m rel-pairs as the edges and corresponding b-boxes as nodes.
   Select nodes first
   (d) Select the top $n\%$ b-boxes as nodes and corresponding rel-pairs as edges.
   (e) Select the b-boxes with confidence score higher than k as nodes and corresponding rel-pairs as edges.
   (f) Select the top m b-boxes as nodes and corresponding rel-pairs as edges.
3. For each node, we generate a d-dimension word embedding using (Mikolov et al., 2013) from the label of b-box, which is considered as our initial node feature. For each edge, we can use a one-hot vector or word embedding to obtain the edge label/feature. If the image has a label, it will be the graph label as well.

Our primary motive for creating Img2SceneGraph and corresponding datasets (described below) is that it allows rapid idea prototyping for graph compression and representation learning.

A.2 IMG2SCENEGRAPH-ACSG

Img2SceneGraph-ACSG is a scene graph dataset with graph labels. We created it by using our Img2SceneGraph pipeline on AI Challenger Scene Graph dataset (Jean-Claude). It is a labeled image dataset that contains 7,120 images with 80 classes. We used method (a) described above with n = 10. The word embedding dimension d is 500.

A.3 VISUAL GENOME

Visual Genome (kri) is an image dataset that contains over 108K images without labels. We chose a small split of it (9987) and use Img2Scenegraph with the same setting (described in Sec. A.2) to generate the scene graph dataset (Img2Scenegraph-VisualGenome). It is used for both graph compression and compression models pretrained to evaluate the graph representation learning performance across datasets.

You can download the source code of Img2SceneGraph pipeline and synthesized dataset (Image2Scenegraph-ACSG and Image2Scenegraph-VisualGenome) from: `http://ilab.usc.edu/datasets/i2sg` , which we plan to keep up-to-date with contributions from ourselves and the community.

B NETWORK ARCHITECTURE

The network architectures of our Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE) is shown in Table. 6, which is used in the experiments of two main tasks: graph compression (main paper Sec. 3.1) and few shot graph classification (main paper Sec. 3.2).

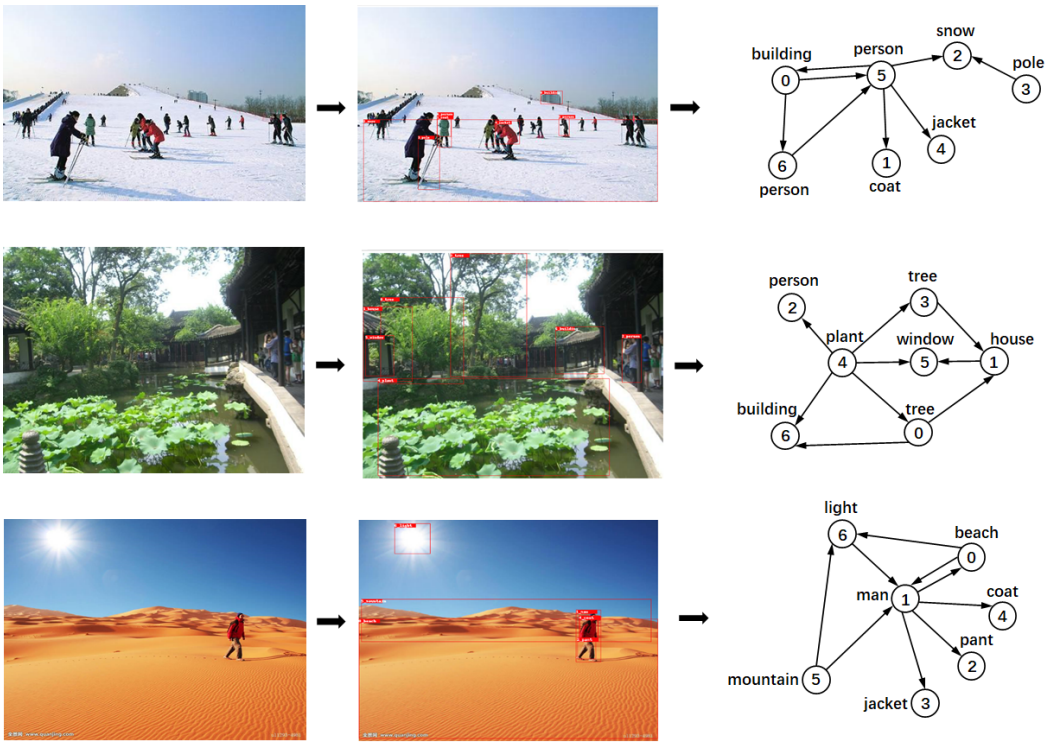

Figure 3: Illustrations on scene graph synthesis with Img2SceneGraph from scratch. All three samples are from Img2SceneGraph-ACSG. Here we use method (f) and n = 7

For compression task, we chose Graph Unet (Gao & Ji, 2019) as the baseline model. Besides, we create 4 other baseline models serving for our ablation study. Here are the settings:

- gPool-GAE: we replaced our pooling method with gPool (Gao & Ji, 2019);
- SAGPool-GAE: we replaced our pooling method with SAG Pool (Lee et al., 2019);
- GCN-GAE: we used GCN (Kipf & Welling, 2016a) to substitute our MI-Conv;
- GAT-GAE: we used GAT (Veličković et al., 2017) to substitute our MI-Conv;

GCN classification models used in the main paper Sec. 3.2 is composed of two graph convolutional layers with node feature dimension 64, followed by l layers MLP to output the final classification result. Specifically, the GCN classifiers used in the FRANKENSTEIN dataset use l=2 without batch normalization. The GCN classifiers used in the Img2SceneGraph-ACSG dataset use l=4 with batch normalization.

C TRAINING DETAILS

We mentioned the high-level training instructions in the main paper Sec. 3. Here are the details.

C.1 GRAPH COMPRESSION

We trained MIAGAE and all the baselines on all the datasets using Adam (Kingma & Ba, 2017) with β_1=0.9 and β_2=0.999, learning rate=0.001, for 500 epochs. For COLORS-3, we chose the data labeled from 0 to 7 as the training set, which contains 7,996 samples. For FRANKENSTEIN, the size of the training set is 4,000. For Img2SceneGraph-ACSG and Visual Genome, the size of the training set is 2,000 and 5,000, respectively.

Table 6: Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE) with 3 pairs of Multi-kernel Inductive Graph convolution layer (MI-Conv) and Similarity Attention Graph Pooling layer (SimAGPool) in Encoder. Decoder is formed by 3 Inductive Un-pooling layers. Some notation: d_0: number of dimensions of input node feature; n_0: number of nodes of input data; n_q: number of nodes after q^{th} pooling; k: number of kernels in MI-Conv; p: pooling ratio $p \in (0; 1]$.

Layer	Input $\rightarrow$ Output Shape	Layer Information
Encoder	$(n_0, d_0) \rightarrow (n_0, 64)$	MI-Conv (*in*=d_0, *out=64, k*), Leaky ReLU
	$(n_0, 64) \rightarrow (n_1, 64)$	SimAGpool (*in=64, p=0.8*)
	$(n_1, 64) \rightarrow (n_1, 64)$	MI-Conv (*in=64, out=64, k*), Leaky ReLU
	$(n_1, 64) \rightarrow (n_2, 64)$	SimAGpool (*in=64, p=0.8*)
	$(n_2, 64) \rightarrow (n_2, 64)$	MI-Conv (*in=64, out=64, k*), Leaky ReLU
	$(n_2, 64) \rightarrow (n_3, 64)$	SimAGpool (*in=64, p=0.8*)
Decoder	$(n_3, 64) \rightarrow (n_2, 64)$	Inductive Un-Pool (*in=64, out=64*), ReLU
	$(n_2, 64) \rightarrow (n_1, 64)$	Inductive Un-Pool (*in=64, out=64*), ReLU
	$(n_1, 64) \rightarrow (n_0, d_0)$	Inductive Un-Pool (*in=64, out=*d_0), ReLU

C.2 Few shot graph classification

C.2.1 FRANKENSTEIN

We chose 100 samples and compressed them with the pre-trained compression model to train the classifier using Adam β_1=0.9 and β_2=0.999, batch size 200, learning rate 0.01 for 1,000 epochs. For the non-compression method, the input is the uncompressed graphs. Then we evaluated the performance on a test set with another 100 samples.

C.2.2 Visual Genome & Img2SceneGraph-ACSG

For the compression part, we first trained MIAGAE and all the baselines on the Visual Genome dataset using Adam β_1=0.9 and β_2=0.999, batch size 500, learning rate 0.001 for 100 epochs. For the classification part, we chose 1,000 samples from the Img2SceneGraph-ACSG dataset and compressed them using the pre-trained model. The compressed graphs will train the classifier using bath size 200, learning rate 0.01, weight decay 0.01 for 1,000 epochs. We evaluated the performance on a test set with another 1,000 samples from Img2SceneGraph-ACSG.