Vision GNN: An Image is Worth Graph of Nodes

Kai Han[1,2*] **Yunhe Wang**[2*] **Jianyuan Guo**[2] **Yehui Tang**[2,3] **Enhua Wu**[1,4]
[1]State Key Lab of Computer Science, ISCAS & UCAS
[2]Huawei Noah's Ark Lab
[3]Peking University [4]University of Macau
{kai.han,yunhe.wang}@huawei.com, weh@ios.ac.cn

Abstract

Network architecture plays a key role in the deep learning-based computer vision system. The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects. In this paper, we propose to represent the image as a graph structure and introduce a new *Vision GNN* (ViG) architecture to extract graph-level feature for visual tasks. We first split the image to a number of patches which are viewed as nodes, and construct a graph by connecting the nearest neighbors. Based on the graph representation of images, we build our ViG model to transform and exchange information among all the nodes. ViG consists of two basic modules: Grapher module with graph convolution for aggregating and updating graph information, and FFN module with two linear layers for node feature transformation. Both isotropic and pyramid architectures of ViG are built with different model sizes. Extensive experiments on image recognition and object detection tasks demonstrate the superiority of our ViG architecture. We hope this pioneering study of GNN on general visual tasks will provide useful inspiration and experience for future research.

The PyTorch code is available at `https://github.com/huawei-noah/Efficient-AI-Backbones` and the MindSpore code is available at `https://gitee.com/mindspore/models`.

1 Introduction

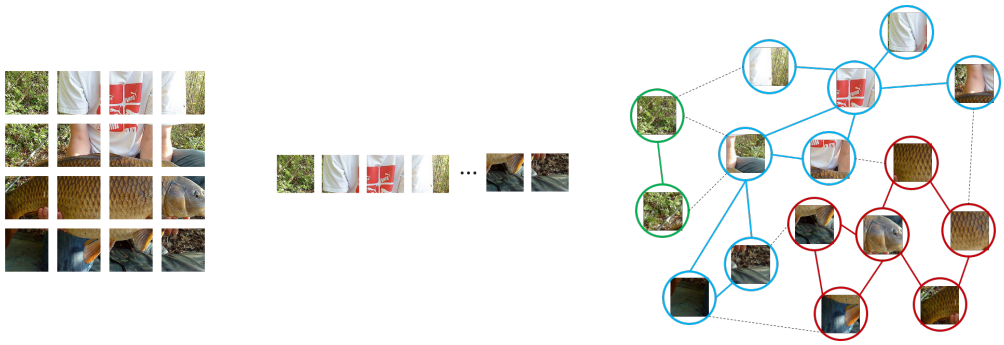
In the modern computer vision system, convolutional neural networks (CNNs) used to be the defacto standard network architecture [29, 27, 17]. Recently, transformer with attention mechanism was introduced for visual tasks [9, 3] and attained competitive performance. MLP-based (multi-layer perceptron) vision models [49, 50] can also work well without using convolutions or self-attention. These progresses are pushing the vision models towards an unprecedented height.

Different networks treat the input image in different ways. As shown in Figure 1, the image data is usually represented as a regular grid of pixels in the Euclidean space. CNNs [29] apply sliding window on the image and introduce the shift-invariance and locality. The recent vision transformer [9] or MLP [49] treats the image as a sequence of patches. For example, ViT [9] divides a 224×224 image into a number of 16×16 patches and forms a sequence with length of 196 as input.

Instead of the regular grid or sequence representation, we process the image in a more flexible way. One basic task of computer vision is to recognize the objects in an image. Since the objects are usually not quadrate whose shape is irregular, the commonly-used grid or sequence structures in previous networks like ResNet and ViT are redundant and inflexible to process them. An object can be viewed as a composition of parts, *e.g.*, a human can be roughly divided into head, upper body,

*Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) Grid structure. (b) Sequence structure. (b) Graph structure.

Figure 1: Illustration of the grid, sequence and graph representation of the image. In the grid structure, the pixels or patches are ordered only by the spatial position. In the sequence structure, the 2D image is transformed to a sequence of patches. In the graph structure, the nodes are linked by its content and are not constrained by the local position.

arms and legs. These parts linked by joints naturally form a graph structure. By analyzing the graph, we are able to recognize the human. Moreover, graph is a generalized data structure that grid and sequence can be viewed as a special case of graph. Viewing an image as a graph is more flexible and effective for visual perception.

Based on the graph representation of images, we build the vision graph neural network (ViG for short) for visual tasks. Instead of treating each pixel as a node which will result in too many nodes (>10K), we divide the input image to a number of patches and view each patch as a node. After constructing the graph of image patches, we use our ViG model to transform and exchange information among all the nodes. The basic cells of ViG include two parts: Grapher and FFN (feed-forward network) modules. Grapher module is constructed based on graph convolution for graph information processing. To alleviate over-smoothing phenomenon of conventional GNN, a FFN module is utilized for node feature transformation and encouraging node diversity. With Grapher and FFN modules, we build our ViG models in both isotropic and pyramid manners. In the experiments, we demonstrate the effectiveness of ViG model on visual tasks like image classification and object detection. For instance, our Pyramid ViG-S achieves 82.1% top-1 accuracy on ImageNet classification task, which outperforms the representative CNN (ResNet [17]), MLP (CycleMLP [5]) and transformer (Swin-T [35]) with similar FLOPs (about 4.5G). To the best of our knowledge, our work is the first to successfully apply graph neural network on large-scale visual tasks. We hope our work will inspire the community to further explore more powerful network architectures.

2 Related Work

In this section, we first revisit the backbone networks in computer vision. Then we review the development of graph neural network, especially GCN and its applications on visual tasks.

2.1 CNN, Transformer and MLP for Vision

The mainstream network architecture in computer vision used to be convolutional network [29, 27, 17]. Starting from LeNet [29], CNNs have been successfully used in various visual tasks, *e.g.*, image classification [27], object detection [42] and semantic segmentation [36]. The CNN architecture is evolving rapidly in the last ten years. The representative works include ResNet [17], MobileNet [21] and NAS [75, 70]. Vision transformer was introduced for visual tasks from 2020 [14, 9, 3, 4]. From then on, a number of variants of ViT [9] were proposed to improve the performance on visual tasks. The main improvements include pyramid architecture [57, 35], local attention [15, 35] and position encoding [61]. Inspired by vision transformer, MLP is also explored in computer vision [49, 50]. With specially designed modules [5, 32, 12, 48], MLP can achieve competitive performance and work on general visual tasks like object detection and segmentation.

2.2 Graph Neural Network

The earliest graph neural network was initially outlined in [11, 44]. Micheli [38] proposed the early form of spatial-based graph convolutional network by architecturally composite nonrecursive layers. In recent several years, the variants of spatial-based GCNs have been introduced, such as [39, 1, 10]. Spectral-based GCN was first presented by Bruna *et al.* [2] that introduced graph convolution based on the spectral graph theory. Since this time, a number of works to improve and extend spectral-based GCN have been proposed [18, 7, 26]. The GCNs are usually applied on graph data, such as social networks [13], citation networks [45] and biochemical graphs [55].

The applications of GCN in the field of computer vision [63, 28, 56, 25] mainly include point clouds classification, scene graph generation, and action recognition. A point cloud is a set of 3D points in space which is usually collected by LiDAR scans. GCN has been explored for classifying and segmenting points clouds [28, 58, 69]. Scene graph generation aims to parse the input image intro a graph with the objects and their relationship, which is usually solved by combining object detector and GCN [63, 68]. By processing the naturally formed graph of linked human joints, GCN was utilized on human action recognition task [24, 67]. GCN can only tackle specific visual tasks with naturally constructed graph. For general applications in computer vision, we need a GCN-based backbone network that directly processes the image data.

3 Approach

In this section, we describe how to transform an image to a graph and introduce vision GNN architectures to learn visual representation.

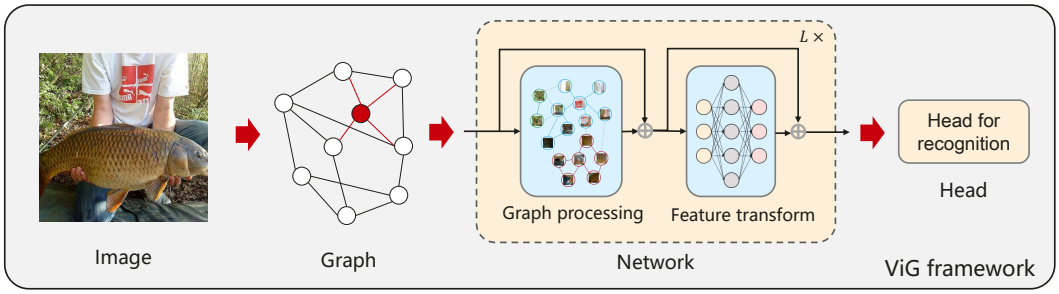


Figure 2: The framework of the proposed ViG model.

3.1 ViG Block

Graph Structure of Image. For an image with size of $H \times W \times 3$, we divided it into N patches. By transforming each patch into a feature vector $\mathbf{x}_i \in \mathbb{R}^D$, we have $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]$ where D is the feature dimension and $i = 1, 2, \cdots, N$. These features can be viewed as a set of unordered nodes which are denoted as $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$. For each node v_i, we find its K nearest neighbors $\mathcal{N}(v_i)$ and add an edge e_{ji} directed from v_j to v_i for all $v_j \in \mathcal{N}(v_i)$. Then we obtain a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{E}$ denote all the edges. We denote the graph construction process as $\mathcal{G} = G(X)$ in the following. By viewing the image as a graph data, we explore how to utilize GNN to extract its representation.

The advantages of graph representation of the image include: 1) graph is a generalized data structure that grid and sequence can be viewed as a special case of graph; 2) graph is more flexible than grid or sequence to model the complex object as an object in the image is usually not quadrate whose shape is irregular; 3) an object can be viewed as a composition of parts (*e.g.*, a human can be roughly divided into head, upper body, arms and legs), and graph structure can construct the connections among those parts; 4) the advanced research on GNN can be transferred to address visual tasks.

Graph-level processing. To be general, we start from the features $X \in \mathbb{R}^{N \times D}$. We first construct a graph based on the features: $\mathcal{G} = G(X)$. A graph convolutional layer can exchange information between nodes by aggregating features from its neighbor nodes. Specifically, graph convolution

operates as follows:

$$\begin{aligned}\mathcal{G}' &= F(\mathcal{G}, \mathcal{W}) \\ &= Update(Aggregate(\mathcal{G}, W_{agg}), W_{update}),\end{aligned} \tag{1}$$

where W_{agg} and W_{update} are the learnable weights of the aggregation and update operations, respectively. More concretely, the aggregation operation computes the representation of a node by aggregating features of neighbor nodes, and the update operation further merge the aggregated feature:

$$\mathbf{x}_i' = h(\mathbf{x}_i, g(\mathbf{x}_i, \mathcal{N}(\mathbf{x}_i), W_{agg}), W_{update}), \tag{2}$$

where $\mathcal{N}(\mathbf{x}_i^l)$ is the set of neighbor nodes of $\mathbf{x}_i^l$. Here we adopt max-relative graph convolution [30] for its simplicity and efficiency:

$$g(\cdot) = \mathbf{x}_i'' = [\mathbf{x}_i, \max(\{\mathbf{x}_j - \mathbf{x}_i | j \in \mathcal{N}(\mathbf{x}_i)\}], \tag{3}$$

$$h(\cdot) = \mathbf{x}_i' = \mathbf{x}_i'' W_{update}, \tag{4}$$

where the bias term is omitted. The above graph-level processing can be denoted as $X' = \text{GraphConv}(X)$.

We further introduce multi-head update operation of graph convolution. The aggregated feature $\mathbf{x}_i''$ is first split into h heads, *i.e.*, $\mathit{head}^1, \mathit{head}^2, \cdots, \mathit{head}^h$ and then these heads are updated with different weights respectively. All the heads can be updated in parallel and are concatenated as the final values:

$$\mathbf{x}_i' = [\mathit{head}^1 W_{update}^1, \mathit{head}^2 W_{update}^2, \cdots, \mathit{head}^h W_{update}^h]. \tag{5}$$

Multi-head update operation allows the model to update information in multiple representation subspaces, which is beneficial to the feature diversity.

ViG block. The previous GCNs usually repeatedly use several graph convolution layers to extract aggregated feature of the graph data. The over-smoothing phenomenon in deep GCNs [31, 40] will decrease the distinctiveness of node features and lead to performance degradation for visual recognition, as shown in Figure 3 where diversity is measured as $\|X - \mathbf{1}\tilde{\mathbf{x}}^T\|$ with $\tilde{\mathbf{x}} = \arg\min_{\tilde{\mathbf{x}}} \|X - \mathbf{1}\tilde{\mathbf{x}}^T\|$ [8]. To alleviate this issue, we introduce more feature transformations and nonlinear activations in our ViG block.

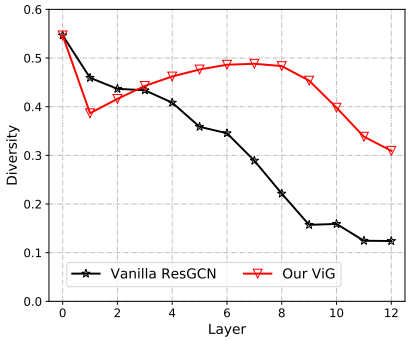


Figure 3: Feature diversity of nodes as layer changes.

We apply a linear layer before and after the graph convolution to project the node features into the same domain and increase the feature diversity. A nonlinear activation function is inserted after graph convolution to avoid layer collapse. We call the upgraded module as Grapher module. In practice, given the input feature $X \in \mathbb{R}^{N \times D}$, the Grapher module can be expressed as

$$Y = \sigma(\text{GraphConv}(XW_{in}))W_{out} + X, \tag{6}$$

where $Y \in \mathbb{R}^{N \times D}$, W_{in} and W_{out} are the weights of fully-connected layers, σ is the activation function, *e.g.*, ReLU and GeLU [19], and the bias term is omitted.

To further encourage the feature transformation capacity and relief the over-smoothing phenomenon, we utilize feed-forward network (FFN) on each node. The FFN module is a simple multi-layer perceptron with two fully-connected layers:

$$Z = \sigma(YW_1)W_2 + Y, \tag{7}$$

where $Z \in \mathbb{R}^{N \times D}$, W_1 and W_2 are the weights of fully-connected layers, and the bias term is omitted. The hidden dimension of FFN is usually larger than D. In both Grapher and FFN modules, batch normalization is applied after every fully-connected layer or graph convolution layer, which is omitted in Eq. 6 and 7 for concision. A stack of Grapher module and FFN module constitutes the ViG block which serves as the basic building unit for constructing a network. Based on the graph representation of images and the proposed ViG block, we can build the ViG network for visual tasks as shown in Figure 2. Compared to vanilla ResGCN [30], our ViG can maintain the feature diversity (Figure 3) as the layer goes deeper so as to learn discriminative representations.

3.2 Network Architecture

In the field of computer vision, the commonly-used transformer usually has an isotropic architecture (*e.g.*, ViT [9]), while CNNs prefer to use pyramid architecture (*i.e.*, ResNet [17]). To have a extensive comparison with other types of neural networks, we build two kinds of network architectures for ViG, *i.e.*, isotropic architecture and pyramid architecture.

Isotropic architecture. Isotropic architecture means the main body has features with equal size and shape throughout the network, such as ViT [9] and ResMLP [50]. We build three versions of isotropic ViG architecture with different models sizes, *i.e.*, ViG-Ti, S and B. The number of nodes is set as $N = 196$. To enlarge the receptive field gradually, the number of neighbor nodes K increases from 9 to 18 linearly as the layer goes deep in these three models. The number of heads is set as $h = 4$ by default. The details are listed in Table 1.

Table 1: Variants of our isotropic ViG architecture. The FLOPs are calculated for the image with 224×224 resolution. 'Ti' denotes tiny, 'S' denotes small, and 'B' denotes base.

Model	Depth	Dimension D	Params (M)	FLOPs (B)
ViG-Ti	12	192	7.1	1.3
ViG-S	16	320	22.7	4.5
ViG-B	16	640	86.8	17.7

Pyramid architecture. Pyramid architecture considers the multi-scale property of images by extracting features with gradually smaller spatial size as the layer goes deeper, such as ResNet [17] and PVT [57]. Empirical evidences show that pyramid architecture is effective for visual tasks [57]. Thus, we utilize the advanced design and build four versions of pyramid ViG models. The details are shown in Table 2. Note that we utilize the spatial reduction [57] in the first two stages to handle large number of nodes.

Table 2: Detailed settings of Pyramid ViG series. D: feature dimension, E: hidden dimension ratio in FFN, K: number of neighbors in GCN, $H \times W$: input image size. 'Ti' denotes tiny, 'S' denotes small, 'M' denotes medium, and 'B' denotes base.

Stage	Output size	PyramidViG-Ti	PyramidViG-S	PyramidViG-M	PyramidViG-B
Stem	$\frac{H}{4} \times \frac{W}{4}$	Conv×3	Conv×3	Conv×3	Conv×3
Stage 1	$\frac{H}{4} \times \frac{W}{4}$	$\begin{bmatrix} D=48 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=80 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=96 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=128 \\ E=4 \\ K=9 \end{bmatrix} \times 2$
Downsample	$\frac{H}{8} \times \frac{W}{8}$	Conv	Conv	Conv	Conv
Stage 2	$\frac{H}{8} \times \frac{W}{8}$	$\begin{bmatrix} D=96 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=160 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=192 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=256 \\ E=4 \\ K=9 \end{bmatrix} \times 2$
Downsample	$\frac{H}{16} \times \frac{W}{16}$	Conv	Conv	Conv	Conv
Stage 3	$\frac{H}{16} \times \frac{W}{16}$	$\begin{bmatrix} D=240 \\ E=4 \\ K=9 \end{bmatrix} \times 6$	$\begin{bmatrix} D=400 \\ E=4 \\ K=9 \end{bmatrix} \times 6$	$\begin{bmatrix} D=384 \\ E=4 \\ K=9 \end{bmatrix} \times 16$	$\begin{bmatrix} D=512 \\ E=4 \\ K=9 \end{bmatrix} \times 18$
Downsample	$\frac{H}{32} \times \frac{W}{32}$	Conv	Conv	Conv	Conv
Stage 4	$\frac{H}{32} \times \frac{W}{32}$	$\begin{bmatrix} D=384 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=640 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=768 \\ E=4 \\ K=9 \end{bmatrix} \times 2$	$\begin{bmatrix} D=1024 \\ E=4 \\ K=9 \end{bmatrix} \times 2$
Head	1×1	Pooling & MLP	Pooling & MLP	Pooling & MLP	Pooling & MLP
Parameters (M)		10.7	27.3	51.7	92.6
FLOPs (B)		1.7	4.6	8.9	16.8

Positional encoding. In order to represent the position information of the nodes, we add a positional encoding vector to each node feature:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{e}_i, \tag{8}$$

where $\mathbf{e}_i \in \mathbb{R}^D$. The absolute positional encoding as described in Eq. 8 is applied in both iostropic and pyramid architectures. For pyramid ViG, we further include relative positional encoding by

following the advanced designs like Swin Transformer [35]. For node i and j, the relative positional distance between them is $\mathbf{e}_i^T \mathbf{e}_j$, which will be added into the feature distance for constructing the graph.

4 Experiments

In this section, we conduct experiments to demonstrate the effectiveness of ViG models on visual tasks including image recognition and object detection.

4.1 Datasets and Experimental Settings

Datasets. In image classification task, the widely-used benchmark ImageNet ILSVRC 2012 [43] is used in the following experiments. ImageNet has 1.2M training images and 50K validation images, which belong to 1000 categories. For the license of ImageNet dataset, please refer to `http://www.image-net.org/download`. For object detection, we use COCO 2017 [34] dataset with 80 object categories. COCO 2017 contains 118K training images and 5K validation images. For the licenses of these datasets, please refer to `https://cocodataset.org/#home`.

Experimental Settings. For all the ViG models, we utilize dilated aggregation [30] in Grapher module and set the dilated rate as $\lceil l/4 \rceil$ for the l-th layer. GELU [19] is used as the nonlinear activation function in Eq. 6 and 7. For ImageNet classification, we use the commonly-used training strategy proposed in DeiT [51] for fair comparison. The data augmentation includes RandAugment [6], Mixup [73], Cutmix [72], random erasing [74] and repeated augment [20]. The details are shown in Table 3. For COCO detection task, we take RetinaNet [33] and Mask R-CNN [16] as the detection frameworks and use our Pyramid ViG as backbone. All the models are trained on COCO 2017 training set in "1×" schedule and evaluated on validation set. We implement the networks using PyTroch [41] and MindSpore [23] and train all our models on 8 NVIDIA V100 GPUs.

Table 3: Training hyper-parameters for ImageNet.

(Pyramid) ViG	Ti	S	M	B
Epochs	300			
Optimizer	AdamW [37]			
Batch size	1024			
Start learning rate (LR)	2e-3			
Learning rate schedule	Cosine			
Warmup epochs	20			
Weight decay	0.05			
Label smoothing [47]	0.1			
Stochastic path [22]	0.1	0.1	0.1	0.3
Repeated augment [20]	✓			
RandAugment [6]	✓			
Mixup prob. [73]	0.8			
Cutmix prob. [72]	1.0			
Random erasing prob. [74]	0.25			
Exponential moving average	0.99996			

Table 4: Results of ViG and other isotropic networks on ImageNet. ♠ CNN, ■ MLP, ♦ Transformer, ★ GNN.

Model	Resolution	Params (M)	FLOPs (B)	Top-1	Top-5
♠ ResMLP-S12 conv3x3 [50]	224×224	16.7	3.2	77.0	-
♠ ConvMixer-768/32 [52]	224×224	21.1	20.9	80.2	-
♠ ConvMixer-1536/20 [52]	224×224	51.6	51.4	81.4	-
♦ ViT-B/16 [9]	384×384	86.4	55.5	77.9	-
♦ DeiT-Ti [51]	224×224	5.7	1.3	72.2	91.1
♦ DeiT-S [51]	224×224	22.1	4.6	79.8	95.0
♦ DeiT-B [51]	224×224	86.4	17.6	81.8	95.7
■ ResMLP-S24 [50]	224×224	30	6.0	79.4	94.5
■ ResMLP-B24 [50]	224×224	116	23.0	81.0	95.0
■ Mixer-B/16 [49]	224×224	59	11.7	76.4	-
★ ViG-Ti (ours)	224×224	7.1	1.3	**73.9**	**92.0**
★ ViG-S (ours)	224×224	22.7	4.5	**80.4**	**95.2**
★ ViG-B (ours)	224×224	86.8	17.7	**82.3**	**95.9**

4.2 Main Results on ImageNet

Isotropic ViG The neural network with iostropic architecture keeps the feature size unchanged in its main computational body, which is easy to scale and is friendly for hardware acceleration. This scheme is widely used in transformer models for natural language processing [53]. The recent neural networks in vision also explore it such as ConvMixer [49], ViT [9] and ResMLP [50]. We compare our isotropic ViG with the existing iostropic CNNs [50, 49], transformers [9, 51] and MLPs [50, 49] in Table 4. From the results, ViG performs better than other types of networks. For example, our ViG-Ti achieves 73.9% top-1 accuracy which is 1.7% higher than DeiT-Ti model with similar computational cost.

Pyramid ViG The pyramid architecture gradually shrinks the spatial size of feature maps as the network deepens, which can leverage the scale-invariant property of images and produce multi-scale features. The advanced networks usually adopt the pyramid architecture, such as ResNet [17], Swin Transformer [35] and CycleMLP [5]. We compare our Pyramid ViG with those representative pyramid networks in Table 5. Our Pyramid ViG series can outperform or be comparable to the state-of-the-art pyramid networks including CNN, MLP and transformer. This indicates that graph neural network can work well on visual tasks and has the potential to be a basic component in computer vision system.

Table 5: Results of Pyramid ViG and other pyramid networks on ImageNet. ♠ CNN, ■ MLP, ♦ Transformer, ★ GNN.

Model	Resolution	Params (M)	FLOPs (B)	Top-1	Top-5
♠ ResNet-18 [17, 59]	224×224	12	1.8	70.6	89.7
♠ ResNet-50 [17, 59]	224×224	25.6	4.1	79.8	95.0
♠ ResNet-152 [17, 59]	224×224	60.2	11.5	81.8	95.9
♠ BoTNet-T3 [46]	224×224	33.5	7.3	81.7	-
♠ BoTNet-T3 [46]	224×224	54.7	10.9	82.8	-
♠ BoTNet-T3 [46]	256×256	75.1	19.3	83.5	-
♦ PVT-Tiny [57]	224×224	13.2	1.9	75.1	-
♦ PVT-Small [57]	224×224	24.5	3.8	79.8	-
♦ PVT-Medium [57]	224×224	44.2	6.7	81.2	-
♦ PVT-Large [57]	224×224	61.4	9.8	81.7	-
♦ CvT-13 [60]	224×224	20	4.5	81.6	-
♦ CvT-21 [60]	224×224	32	7.1	82.5	-
♦ CvT-21 [60]	384×384	32	24.9	83.3	-
♦ Swin-T [35]	224×224	29	4.5	81.3	95.5
♦ Swin-S [35]	224×224	50	8.7	83.0	96.2
♦ Swin-B [35]	224×224	88	15.4	83.5	96.5
■ CycleMLP-B2 [5]	224×224	27	3.9	81.6	-
■ CycleMLP-B3 [5]	224×224	38	6.9	82.4	-
■ CycleMLP-B4 [5]	224×224	52	10.1	83.0	-
■ Poolformer-S12 [71]	224×224	12	2.0	77.2	93.5
■ Poolformer-S36 [71]	224×224	31	5.2	81.4	95.5
■ Poolformer-M48 [71]	224×224	73	11.9	82.5	96.0
★ Pyramid ViG-Ti (ours)	224×224	10.7	1.7	**78.2**	**94.2**
★ Pyramid ViG-S (ours)	224×224	27.3	4.6	**82.1**	**96.0**
★ Pyramid ViG-M (ours)	224×224	51.7	8.9	**83.1**	**96.4**
★ Pyramid ViG-B (ours)	224×224	92.6	16.8	**83.7**	**96.5**

4.3 Ablation Study

We conduct ablation study of the proposed method on ImageNet classification task and use the isotropic ViG-Ti as the base architecture.

Type of graph convolution. We test the representative variants of graph convolution, including EdgeConv [58], GIN [64], GraphSAGE [13] and Max-Relative GraphConv [30]. From table 6, we can see that the top-1 accuracies of different graph convolutions are better than that of DeiT-Ti,

Table 6: ImageNet results of different types of graph convolution. The basic architecture is ViG-Ti.

GraphConv	Params (M)	FLOPs (B)	Top-1
EdgeConv [58]	7.2	2.4	74.3
GIN [64]	7.0	1.3	72.8
GraphSAGE [13]	7.3	1.6	74.0
Max-Relative GraphConv [30]	7.1	1.3	73.9

indicating the flexibility of ViG architecture. Among them, Max-Relative achieves the best trade-off between FLOPs and accuracy. In rest of the experiments, we use Max-Relative GraphConv by default unless specially stated.

The effects of modules in ViG. To make graph neural network adaptive to visual task, we introduce FC layers in Grapher module and utilize FFN block for feature transformation. We evaluate the effects of these modules by ablation study. We change the feature dimension of the compared models to make their FLOPs similar, so as to have a fair comparison. From Table 7, we can see that directly utilizing graph convolution for image classification performs poorly. Adding more feature transformation by introducing FC and FFN consistently increase the accuracy.

Table 7: The effects of modules in ViG on ImageNet.

GraphConv	FC in Grapher module	FFN module	Params (M)	FLOPs (B)	Top-1
✓	✗	✗	5.8	1.4	67.0
✓	✓	✗	4.4	1.4	73.4
✓	✗	✓	7.7	1.3	73.6
✓	✓	✓	7.1	1.3	73.9

The number of neighbors. In the process of constructing graph, the number of neighbor nodes K is a hyperparameter controlling the aggregated range. Too few neighbors will degrade information exchange, while too many neighbors will lead to over-smoothing. We tune K from 3 to 20 and show the results in Table 8. We can see that the number of neighbor nodes in the range from 9 to 15 can perform well on ImageNet classification task.

Table 8: Top-1 accuracy *vs.* K on ImageNet.

K	3	6	9	12	15	20	9 to 18
Top-1	72.2	73.4	73.6	73.6	73.5	73.3	73.9

The number of heads. Multi-head update operation allows Grapher module to process node features in different subspaces. The number of heads h in Eq. 5 controls the transformation diversity in subspaces and the FLOPs. We tune h from 1 to 8 and show the results in Table 9. The FLOPs and top-1 accuracy on ImageNet changes slightly for different h. We select $h = 4$ as default value for the optimal trade-off between FLOPs and accuracy.

Table 9: Top-1 accuracy *vs.* h on ImageNet.

h	1	2	4	6	8
FLOPs / Top-1	1.6B / 74.2	1.4B / 74.0	1.3B / 73.9	1.2B / 73.7	1.2B / 73.7

4.4 Object Detection

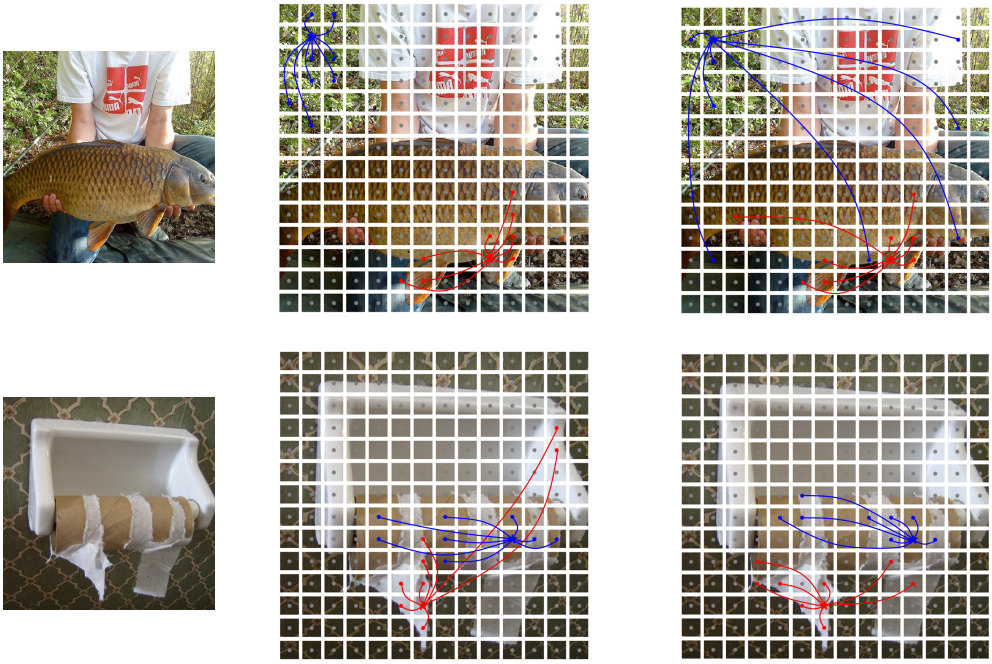
We apply our ViG model on object detection task to evaluate its generalization. To have a fair comparison, we utilize the ImageNet pretrained Pyramid ViG-S as the backbone of RetinaNet [33] and Mask R-CNN [16] detection frameworks. The models are trained in the commonly-used "1x" schedule and FLOPs is calculated with 1280×800 input size. From the results in Table 10, we can see that our Pyramid ViG-S performs better than the representative backbones of different types, including ResNet [17], CycleMLP [5] and Swin Transformer [35] on both RetinaNet and Mask R-CNN. The superior results demonstrate the generalization ability of ViG architecture.

Table 10: Object detection and instance segmentation results on COCO val2017. Our Pyramid ViG is compared with other backbones on RetinaNet and Mask R-CNN frameworks.

Backbone	RetinaNet 1×							
	Param	FLOPs	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet50 [17]	37.7M	239.3B	36.3	55.3	38.6	19.3	40.0	48.8
ResNeXt-101-32x4d [62]	56.4M	319B	39.9	59.6	42.7	22.3	44.2	52.5
PVT-Small [57]	34.2M	226.5B	40.4	61.3	44.2	25.0	42.9	55.7
CycleMLP-B2 [5]	36.5M	230.9B	40.6	61.4	43.2	22.9	44.4	54.5
Swin-T [35]	38.5M	244.8B	41.5	62.1	44.2	25.1	44.9	**55.5**
Pyramid ViG-S (ours)	36.2M	240.0B	**41.8**	**63.1**	**44.7**	**28.5**	**45.4**	53.4
Backbone	**Mask R-CNN 1×**							
	Param	FLOPs	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet50 [17]	44.2M	260.1B	38.0	58.6	41.4	34.4	55.1	36.7
PVT-Small [57]	44.1M	245.1B	40.4	62.9	43.8	37.8	60.1	40.3
CycleMLP-B2 [5]	46.5M	249.5B	42.1	64.0	45.7	38.9	61.2	41.8
PoolFormer-S24 [71]	41.0M	-	40.1	62.2	43.4	37.0	59.1	39.6
Swin-T [35]	47.8M	264.0B	42.2	64.6	**46.2**	39.1	61.6	**42.0**
Pyramid ViG-S (ours)	45.8M	258.8B	**42.6**	**65.2**	46.0	**39.4**	**62.4**	41.6

4.5 Visualization

To better understand how our ViG model works, we visualize the constructed graph structure in ViG-S. In Figure 4, we show the graphs of two samples in different depths (the 1st and the 12th blocks). The pentagram is the center node, and the nodes with the same color are its neighbors. Two center nodes are visualized as drawing all the edges will be messy. We can observe that our model can select the content-related nodes as the first order neighbors. In the shallow layer, the neighbor nodes tend to be selected based on low-level and local features, such as color and texture. In the deep layer, the neighbors of the center nodes are more semantic and belong to the same category. Our ViG network can gradually link the nodes by its content and semantic representation and help to better recognize the objects.



(a) Input image. (b) Graph connection in the 1st block. (c) Graph connection in the 12th block.

Figure 4: Visualization of the constructed graph structure. The pentagram is the center node, and the nodes with the same color are its neighbors in the graph.

5 Conclusion

In this work, we pioneer to study representing the image as graph data and leverage graph neural network for visual tasks. We divide the image into a number of patches and view them as nodes. Constructing graph based on these nodes can better represent the irregular and complex objects in the wild. Directly using graph convolution on the image graph structure has over-smoothing problem and performs poorly. We introduce more feature transformation inside each node to encourage the information diversity. Based on the graph representation of images and improved graph block, we build our vision GNN (ViG) networks with both isotropic and pyramid architectures. Extensive experiments on image recognition and object detection demonstrate the superiority of the proposed ViG architecture. We hope this pioneering work on vision GNN can serve as a basic architecture for general visual tasks.

Acknowledgement

This research is supported by NSFC (62072449, 61872345), National Key R&D Program of China (2021YFB1715800), and Macau Science &Tech. Fund (0018/2019/AKP. We gratefully acknowledge the support of MindSpore, CANN (Compute Architecture for Neural Networks) and Ascend AI Processor used for this research.

References

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pages 2001–2009, 2016.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, 2020.

[4] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *CVPR*, 2021.

[5] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. In *ICLR*, 2022.

[6] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, volume 29, 2016.

[8] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *ICML*, pages 2793–2803. PMLR, 2021.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272. PMLR, 2017.

[11] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, volume 2, pages 729–734, 2005.

[12] Jianyuan Guo, Yehui Tang, Kai Han, Xinghao Chen, Han Wu, Chao Xu, Chang Xu, and Yunhe Wang. Hire-mlp: Vision mlp via hierarchical rearrangement. In *CVPR*, 2022.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1025–1035, 2017.

[14] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[15] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. In *NeurIPS*, 2021.

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, pages 2961–2969, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[18] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[19] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[20] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *CVPR*, 2020.

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[22] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661. Springer, 2016.

[23] Huawei. Mindspore. `https://www.mindspore.cn/`, 2020.

[24] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *CVPR*, pages 5308–5317, 2016.

[25] Yongcheng Jing, Yining Mao, Yiding Yang, Yibing Zhan, Mingli Song, Xinchao Wang, and Dacheng Tao. Learning graph neural networks for image style transfer. In *ECCV*, 2022.

[26] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[28] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, pages 4558–4567, 2018.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, pages 9267–9276, 2019.

[31] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545, 2018.

[32] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. As-mlp: An axial shifted mlp architecture for vision. In *ICLR*, 2022.

[33] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014.

[35] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 10012–10022, 2021.

[36] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[38] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[39] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023. PMLR, 2016.

[40] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[44] 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, 2008.

[45] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[46] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *CVPR*, pages 16519–16529, 2021.

[47] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[48] Yehui Tang, Kai Han, Jianyuan Guo, Chang Xu, Yanxi Li, Chao Xu, and Yunhe Wang. An image patch is a wave: Phase-aware vision mlp. In *CVPR*, 2022.

[49] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, volume 34, 2021.

[50] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[51] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[52] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[54] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *NeurIPS*, pages 3839–3848, 2018.

[55] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[56] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *ICCV*, pages 3056–3065, 2019.

[57] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021.

[58] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[59] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

[60] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *ICCV*, pages 22–31, 2021.

[61] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *ICCV*, pages 10033–10041, 2021.

[62] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 1492–1500, 2017.

[63] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *CVPR*, pages 5410–5419, 2017.

[64] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2018.

[65] Yixing Xu, Kai Han, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Learning frequency domain approximation for binary neural networks. In *NeurIPS*, volume 34, pages 25553–25565, 2021.

[66] Yixing Xu, Yunhe Wang, Hanting Chen, Kai Han, Chunjing Xu, Dacheng Tao, and Chang Xu. Positive-unlabeled compression on the cloud. In *NeurIPS*, volume 32, 2019.

[67] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

[68] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *ECCV*, pages 670–685, 2018.

[69] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *CVPR*, pages 7074–7083, 2020.

[70] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020.

[71] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *CVPR*, 2022.

[72] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

[73] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[74] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, volume 34, pages 13001–13008, 2020.

[75] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 1.
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] No potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4.1.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] The common settings on ImageNet and COCO datasets.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.1.
   (b) Did you mention the license of the assets? [Yes] See Section 4.1.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Theoretical Analysis

In our ViG block, we propose to increase feature diversity in nodes by utilizing more feature transformation such as FFN module. We show the empirical comparison between vanilla ResGCN and our ViG model in our paper. Here we make a simple theoretical analysis of the benefit of FFN module in ViG on increasing the feature diversity. Given the output features of graph convolution $X \in \mathbb{R}^{N \times D}$, the feature diversity [8] is measured as

$$\gamma(X) = \|X - \mathbf{1}\mathbf{x}^T\|, \quad \text{where} \quad \mathbf{x} = \arg\min_{\mathbf{x}} \|X - \mathbf{1}\mathbf{x}^T\|, \tag{9}$$

where $\|\cdot\|$ is the $\ell_{1,\infty}$ norm of a matrix. By applying FFN module on the features, we have the following theorem.

Theorem 1. *Given a FFN module, the diversity $\gamma(FFN(X))$ of its output features satisfies*

$$\gamma(FFN(X)) \leq \lambda\gamma(X), \tag{10}$$

where λ is the Lipschitz constant of FFN with respect to p-norm for $p \in [1, \infty]$.

Proof. The FFN includes weight matrix multiplication, bias addition and elementwise nonlinear function, which all preserve the constancy-across-rows property of $\text{FFN}(\mathbf{1}\mathbf{x}^T)$. Therefore, we have

$$\begin{aligned}
\gamma(\text{FFN}(X)) &= \|\text{FFN}(X) - \mathbf{1}\mathbf{x}'^T\|_p \\
&\leq \|\text{FFN}(X) - \text{FFN}(\mathbf{1}\mathbf{x}^T)\|_p && \triangleright \text{ FFN preserves constancy-across-rows.} \\
&\leq \lambda\|X - \mathbf{1}\mathbf{x}^T\|_p && \triangleright \text{ Definition of Lipschitz constant.} \\
&= \lambda\gamma(X),
\end{aligned}$$

□

The Lipschitz constant of FFN is related to the norm of weight matrices and is usually much larger than 1 [54]. Thus, the Theorem 1 shows that introducing $\gamma(\text{FFN}(X))$ in our ViG block tends to improve the feature diversity in graph neural network.

A.2 Pseudocode

The proposed Vision GNN framework is easy to be implemented based on the commonly-used layers without introducing complex operations. The pseudocode of the core part, *i.e.*, ViG block is shown in Algorithm 1.

Algorithm 1 PyTorch-like Code of ViG Block

```
import torch.nn as nn
from gcn_lib.dense.torch_vertex import DynConv2d
# gcn_lib is downloaded from https://github.com/lightaime/deep_gcns_torch

class GrapherModule(nn.Module):
  """Grapher module with graph conv and FC layers
  """
  def __init__(self, in_channels, hidden_channels, k=9, dilation=1, drop_path=0.0):
    super(GrapherModule, self).__init__()
    self.fc1 = nn.Sequential(
      nn.Conv2d(in_channels, in_channels, 1, stride=1, padding=0),
      nn.BatchNorm2d(in_channels),
    )
    self.graph_conv = nn.Sequential(
      DynConv2d(in_channels, hidden_channels, k, dilation, act=None),
      nn.BatchNorm2d(hidden_channels),
      nn.GELU(),
    )
    self.fc2 = nn.Sequential(
      nn.Conv2d(hidden_channels, in_channels, 1, stride=1, padding=0),
      nn.BatchNorm2d(in_channels),
    )
    self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()
```

```
  def forward(self, x):
    B, C, H, W = x.shape
    x = x.reshape(B, C, -1, 1).contiguous()
    shortcut = x
    x = self.fc1(x)
    x = self.graph_conv(x)
    x = self.fc2(x)
    x = self.drop_path(x) + shortcut
    return x.reshape(B, C, H, W)

class FFNModule(nn.Module):
  """Feed-forward Network
  """
  def __init__(self, in_channels, hidden_channels, drop_path=0.0):
    super(FFNModule, self).__init__()
    self.fc1 = nn.Sequential(
      nn.Conv2d(in_channels, hidden_channels, 1, stride=1, padding=0),
      nn.BatchNorm2d(hidden_channels),
      nn.GELU()
    )
    self.fc2 = nn.Sequential(
      nn.Conv2d(hidden_channels, in_channels, 1, stride=1, padding=0),
      nn.BatchNorm2d(in_channels),
    )
    self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()

  def forward(self, x):
    shortcut = x
    x = self.fc1(x)
    x = self.fc2(x)
    x = self.drop_path(x) + shortcut
    return x

class ViGBlock(nn.Module):
  """ViG block with Grapher and FFN modules
  """
  def __init__(self, channels, k, dilation, drop_path=0.0):
    super(ViGBlock, self).__init__()
    self.grapher = GrapherModule(channels, channels * 2, k, dilation, drop_path)
    self.ffn = FFNModule(channels, channels * 4, drop_path)

  def forward(self, x):
    x = self.grapher(x)
    x = self.ffn(x)
    return x
```