WiGNet: Windowed Vision Graph Neural Network

Gabriele Spadaro[1,2] Marco Grangetto[1] Attilio Fiandrotti[1,2]
Enzo Tartaglione[2] Jhony H. Giraldo[2]

[1]University of Turin, Italy
[2] LTCI, Télécom Paris, Institut Polytechnique de Paris, France
gabriele.spadaro@unito.it

Abstract

*In recent years, Graph Neural Networks (GNNs) have demonstrated strong adaptability to various real-world challenges, with architectures such as Vision GNN (ViG) achieving state-of-the-art performance in several computer vision tasks. However, their practical applicability is hindered by the computational complexity of constructing the graph, which scales quadratically with the image size. In this paper, we introduce a novel **Wi**ndowed vision **G**raph neural **Net**work (WiGNet) model for efficient image processing. WiGNet explores a different strategy from previous works by partitioning the image into windows and constructing a graph within each window. Therefore, our model uses graph convolutions instead of the typical 2D convolution or self-attention mechanism. WiGNet effectively manages computational and memory complexity for large image sizes. We evaluate our method in the ImageNet-1k benchmark dataset and test the adaptability of WiGNet using the CelebA-HQ dataset as a downstream task with higher-resolution images. In both of these scenarios, our method achieves competitive results compared to previous vision GNNs while keeping memory and computational complexity at bay. WiGNet offers a promising solution toward the deployment of vision GNNs in real-world applications. We publicly released the code and pre-trained models at https://github.com/EIDOSLAB/WiGNet.*

1. Introduction

In the last decade, the field of computer vision has progressed significantly, largely due to the success of deep neural networks [20]. These models are now established as state-of-the-art in several tasks such as image classification, object detection, semantic segmentation, etc [1, 3, 14, 30–32]. In particular, Convolutional Neural Networks (CNNs) [21] exploit the locality of natural images to extract features while Vision Transformers (ViTs) [6,24,36] implement the attention operator to exploit long-range dependencies of the input image. Recently, vision-based Graph Neural Networks (GNNs) [11, 27, 28], have been proposed with promising results for vision tasks. Vision GNNs first build a graph over the features extracted from the image, and then apply graph convolutions instead of regular 2D convolutions (CNNs) or the self-attention mechanism (ViTs). As a result, vision GNNs have benefited from the rich literature of GNNs [43], adding a new dimension to the landscape of deep learning for image analysis.

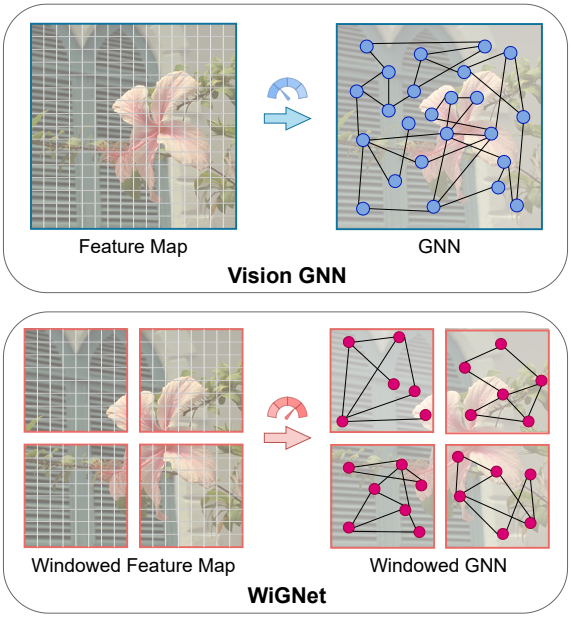


Figure 1. Implementation of Vision GNN (ViG) [11] and the proposed WiGNet. Our model first divides images into local windows where graphs are built. This fundamental change dramatically increases computational and memory efficiency in vision tasks.

Despite the promising results achieved by vision GNNs,

there still remain some open challenges. More precisely, the Vision GNN (ViG) model [11] relies on the k-Nearest Neighbors (k-NN) method to construct the graph. Therefore, the computational complexity of ViG increases quadratically with the number of nodes (patches) extracted from the image and hence with the image size (please refer to Fig. 5 for further details). This hinders their applicability to real-world large-scale datasets and high-resolution images, limiting their practical use. Overcoming the scalability issue in vision GNNs is crucial for their wider adoption and deployment in real-world applications.

To address the scalability challenges, we propose a novel **Wi**ndowed vision **G**raph neural **Net**work (WiGNet) model. Similar to previous vision GNNs, WiGNet treats an image as a graph, yet in a fundamentally different manner. Namely, the image is first partitioned into non-overlapping windows and only then a separate graph is built for each window as shown in Fig. 1. The complexity of building the graphs in our approach grows only linearly with the number of windows, while maintaining competitive results in image classification tasks. By focusing on localized regions within the image through windowed processing, WiGNet efficiently captures relevant features while alleviating the scalability issues encountered by previous vision GNN approaches.

Our work makes the following significant contributions:

- To the best of our knowledge, we are the first to introduce the concept of windowed processing in the context of vision GNNs.
- We show that the computational and memory complexity of WiGNet only grows linearly with the image size, paving the way for broader applications of graph-based models in computer vision.
- We thoroughly validate WiGNet in the ImageNet-1k benchmark dataset and test its adaptability as a feature extractor on CelebA-HQ as a downstream task with higher resolution images. In both of these scenarios, WiGNet outperforms or obtains competitive performance regarding previous deep learning models like CNNs, ViTs, and ViGs.

Our results in ImageNet-1k suggest that WiGNet successfully exploits vision GNNs for image classification tasks. In addition, classification results of higher-resolution images show that WiGNet is able to achieve state-of-the-art results while keeping complexity under control.

2. Related Work

In this section, we first give an overview of deep learning networks typically adopted in computer vision like CNNs and ViTs. Then we review GNNs, analyzing their applications to visual tasks and their limitations.

2.1. CNNs and ViTs.

Convolutional Neural Networks (CNNs) started dominating the computer vision field since the seminal AlexNet [19] paper. CNNs exploit the locality of pixels to extract features from the input image useful to the task on which they are trained. CNNs represented the de-facto standard to solve different tasks from image classification to object detection [30, 31], semantic segmentation [13, 32], image compression [1, 26], and many others. The rapid development that these architectures have experienced over the past decade has led to the development of models such as ResNet [14] and MobileNet [16], among others [17, 34, 35].

More recently, researchers in computer vision have focused on Visual Transformers (ViTs), with the self-attention mechanism at its core. ViTs build upon the attention mechanism proposed by the Transformer architecture [37] for Natural Language Processing (NLP) tasks in origin, and later on applied with success to different computer vision tasks [3, 6, 24, 40]. Attention enables capturing long-range dependencies between pixels, achieving state-of-the-art results in several computer vision tasks. The Swin Transformer [24], in particular, proposed a hierarchical Transformer-based architecture to extract tokens at different scales and work with high-resolution images. The multi-head self-attention operator in Swin Transformer is computed in non-overlapped windows. To introduce cross-window connections, the authors proposed a shifted window partitioning approach that alternates with the regular window partitioning in consecutive blocks of Swin Transformer. This method allows for connections between neighboring windows in the previous layer, leading to improvements in image classification, object detection, and semantic segmentation [24].

The core function in Swin Transformers can be thought of as an attention operator applied in fully connected graphs from windows in images. WiGNet is, in essence, different from Swin Transformers since (i) we operate in k-NN graphs instead of fully connected graphs, and (ii) we use a GNN function instead of the self-attention mechanism. These two changes achieve competitive results regarding the Swin Transformer, other ViTs, and CNN models.

2.2. Graphs in Computer Vision.

GNNs emerged as an extension of the convolution operation of CNNs for regular-structured data such as images to the graph domain. GNNs are typically used for learning graph-structured data representations. Bruna *et al*. [2] proposed the first modern GNN by extending the convolutional operator of CNNs to graphs. Incorporating concepts of signal processing on graphs, Defferrard *et al*. [5] introduced localized spectral filtering for graphs. Later, Kipf and Welling [18] approximated the spectral filtering operation to obtain efficient Graph Convolutional Networks (GCNs). Inspired by these works, Veličković *et al*. [38]

presented the attention mechanism on GNNs, resulting in a graph where the connection weights are unique and learned for each edge. This allows GATs to effectively model complex relationships between nodes, although with increased computational complexity.

Even though GNNs have generally been adopted for graph-based data [18], they have recently demonstrated remarkable success when applied to tasks such as image classification [11, 27, 28] and segmentation [8]. The Vision GNN (ViG) model [11], in particular, drew inspiration from the partition concept introduced in ViTs [6], dividing the input image into smaller patches, and considering each of these patches as a node in the graph of the image. To establish connections between these nodes, the k-NN algorithm is adopted by considering the similarity of nodes in the feature space. These features are updated using graph convolution operators, considering the features of the node itself and those of its neighbors [7]. Following a similar paradigm as Transformers, these features contribute to the classification of the entire graph, thereby classifying the entire image.

Using a graph can be beneficial in image processing tasks as it allows for the exploitation of non-local dependencies without the need for multiple convolutional layers and to model complex objects having irregular shapes. Furthermore, graphs are a more general data structure with respect to a grid of pixels (as modeled by CNNs) or a fully connected graph of patches (as modeled by ViTs). These advantages have led graph-based models to reach state-of-the-art not only in image classification but also in object detection and instance segmentation [11, 28]. However, vision GNNs still have a very high computational complexity, especially when working with high-resolution images. For this reason, in the first layers of ViG, the graph is created in a bipartite way, connecting patches of the original feature maps with a subsample version of it, obtained using a non-learnable subsampling filter. Although this approach decreases the complexity of ViG, it still has a quadratic time complexity, making it slow for processing large images.

More recently, new techniques have emerged to reduce the complexity of ViG by focusing on the graph construction phase. MobileViG [27], for instance, proposed a Sparse Vision Graph Attention (SVGA) module, in which the graph is statically constructed, thus without adopting k-NN. Here a patch of the image is connected to patches at a certain hop distance on the same row and column. In this way the number of connections depends on the size of the image, rapidly increasing the memory required to perform graph convolutions. Thus, to obtain a mobile-friendly model, MobileViG only adopts the SVGA module in the last stage of the architecture (where the input tensor is smaller) while the previous stages are implemented using classical depth-wise 2D convolutions. This results in a hybrid model in which the CNNs and GNNs are both adopted. GreedyViG [28] proposed a dynamic version of SVDA named Dynamic Axial Graph Construction (DAGC). This module adopts the same fixed graph of SVDA and dynamically masks some connections. To create this mask, GreedyViG estimates the mean and standard deviation of the Euclidean distances between the patches in the original image and a diagonally flipped version. Moreover, this DAGC module is implemented in each stage of the architecture, making this CNN-GNN model highly memory-intensive.

Unlike these previous methods, WiGNet always works on the original feature maps and not on an undersampled version (like ViG). Our method partitions these feature maps into windows of fixed size in which the graph can be constructed. Moreover, the k-NN operator is not replaced with a fixed graph structure, but we exploit the locality of pixels to reduce the complexity of this operation, making it linear with respect to the size of the input image.

3. Windowed Vision Graph Neural Network

This section describes in detail the proposed WiGNet architecture. Firstly, we provide some background on graphs and GNNs. Secondly, we describe the architectural design motivating our choices. Then we discuss the implication of computational complexity, comparing WiGNet with the ViG model. Finally, we propose three different WiGNet versions that we use for our experiments in Section 4.

3.1. Preliminaries

Graph. A graph is a mathematical entity that can be represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ is the set of nodes, and $\mathcal{E} \subseteq \{(i, j) \mid i, j \in \mathcal{V} \text{ and } i \neq j\}$ is the set of edges between nodes i and j. We can associate F-dimensional feature vectors to every i-th node in G such that $\mathbf{x}_i \in \mathbb{R}^F$. Therefore, we represent the whole set of features in G with the matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times F}$.

Message passing function. In GNNs, the message-passing function is the standard paradigm for computing graph convolutions [7]. Let $\mathbf{x}'_i$ be the output of a generic graph convolution, we can thus define the message-passing function as follows:

$$\mathbf{x}'_i = \mathrm{UPDATE}\left(\mathbf{x}_i, \mathrm{AGG}(\{\mathbf{x}_j \; \forall \, j \in \mathcal{N}_i\})\right), \qquad (1)$$

where $\mathcal{N}_i$ is the set of neighbors of i, $\mathrm{AGG}(\cdot)$ is a generic function used to aggregate neighbor information, and $\mathrm{UPDATE}(\cdot)$ updates the representation of the node itself. A graph convolutional layer can be thought of as an implementation of this message-passing operator by concretely defining the update and aggregation functions.

3.2. WiGNet Architecture

Architecture overview. Fig. 2a shows a bird's eye view of the WiGNet architecture, implementing a four-stage pyra-

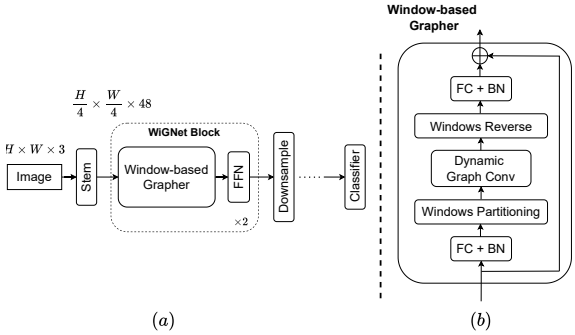


Figure 2. (a): WiGNet architecture exemplified for the *Tiny* version (see Table 1 for details), equipped with a linear classifier. (b): A graphical illustration of the Window-based Grapher module.

midal feature extractor, where at each stage features of increasingly smaller sizes are extracted. A WiGNet is composed of three basic building blocks: (i) the *Stem*, (ii) the *WiGNet block*, and (iii) the *downsampling* module.

The Stem block is a simple feature extractor composed of three convolutional layers that receive as input an image of size $H \times W \times 3$, divides the image into N patches and transforms them into a feature vector $\mathbf{x}_i \in \mathbb{R}^F$ for each patch, obtaining $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]^\top$.

The WiGNet block is composed of a *Window-based Grapher* module and a *Feed Forward Network* (FFN). The Grapher module partitions the image into non-overlapping windows, builds a graph for each window, and then local GNN updates are applied to each window. This is a fundamentally different approach than ViG [11] where a large graph is built on top of the entire image, with the complexity implications discussed above. The FFN module further encourages feature diversity. The downsampling block reduces the feature dimension by merging node representations. Each of the downsampling modules reduces the number of nodes by a factor of 2 while increasing the size of the feature vectors associated with the remaining nodes.

The complete network architecture is composed of a stack of the Stem function and four WiGNet-plus-Downsampling blocks. Fig. 2a shows a final fully connected layer for producing class scores. We propose three versions of WiGNet in Table 1: (i) tiny (WiGNet-Ti), (ii) small (WiGNet-S), and (iii) medium (WiGNet-M). In the following, we describe in detail the Grapher module, at the core of WiGNet.

The Grapher module. The Window-based Grapher module illustrated in Fig. 2b is at the core of WiGNet. Preliminary, the feature vector generated from the Stem module (or the previous Grapher module) is processed by a fully connected layer with batch normalization. Firstly, the *Windows Partitioning* component splits the input tensor into non-overlapping windows having a fixed size of $M \times M$. Secondly, the *Dynamic Graph Convolution* component builds a graph and performs graph convolution independently for each window. This component at the core of the Grapher and is described in detail in the following section. Next, the *Windows Reverse* component reshapes the output of the Dynamic Graph Convolution into the original feature vector as generated by the Windows Partitioning component. The feature vector is then passed as input to a fully connected layer with batch normalization. Finally, the Window-based Grapher block is completed with a skip connection.

Stage	Output size	WiGNet-Ti	WiGNet-S	WiGNet-M
Stem	$\frac{H}{4} \times \frac{W}{4}$	Conv×3	Conv×3	Conv×3
Stage 1	$\frac{H}{4} \times \frac{W}{4}$	$\begin{bmatrix} D=48 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=80 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=96 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$
Downsample	$\frac{H}{8} \times \frac{W}{8}$	Conv	Conv	Conv
Stage 2	$\frac{H}{8} \times \frac{W}{8}$	$\begin{bmatrix} D=96 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=160 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=192 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$
Downsample	$\frac{H}{16} \times \frac{W}{16}$	Conv	Conv	Conv
Stage 3	$\frac{H}{16} \times \frac{W}{16}$	$\begin{bmatrix} D=240 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 6$	$\begin{bmatrix} D=400 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 6$	$\begin{bmatrix} D=384 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 16$
Downsample	$\frac{H}{32} \times \frac{W}{32}$	Conv	Conv	Conv
Stage 4	$\frac{H}{32} \times \frac{W}{32}$	$\begin{bmatrix} D=384 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=640 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$	$\begin{bmatrix} D=768 \\ E=4 \\ k=9 \\ W=8 \end{bmatrix} \times 2$
Head	1×1	Pooling & MLP	Pooling & MLP	Pooling & MLP
Parameters (M)		10.8	27.4	49.7
MACs (B)		2.1	5.7	11.2

Table 1. Detailed settings of WiGNet series. D: feature dimension, E: hidden dimension ratio in FFN, k: number of neighbors in GCN, W: window size, $H \times W$: input image size. 'Ti' denotes tiny, 'S' denotes small, and 'M' denotes medium.

Dynamic Graph Convolution component. For each w-th window, the dynamic graph convolution component implements the k-NN algorithm to produce a graph $G^w = (\mathcal{V}^w, \mathcal{E}^w)$, where $\mathcal{E}^w \subseteq \{(i,j) \mid i,j \in \mathcal{V}^w\}$ is the set of edges and $\mathcal{V}^w$ is the set of nodes in G^w. In particular, two nodes (i,j) are connected if $j \in \mathcal{N}_i^w$, where $\mathcal{N}_i^w$ is the set of k nearest neighbors for the node i belonging to the same window w. Therefore, similarly to ViG [11], we apply the Max-Relative graph convolution proposed by Li *et al*. [23] to update the representation of the i-th node in the w-th window as follows:

$$\mathbf{x}'^w_i = \mathbf{W}_{\text{update}} \left(\mathbf{x}^w_i \parallel \max(\{\mathbf{x}^w_j - \mathbf{x}^w_i \; \forall \, j \in \mathcal{N}^w_i\}) \right), \tag{2}$$

where $\parallel$ is the concatenation function, and $\mathbf{W}_{\text{update}}$ is a matrix of learnable parameters.

Fig. 4 shows an example of the computation of $\mathbf{x}'^w_i$ in

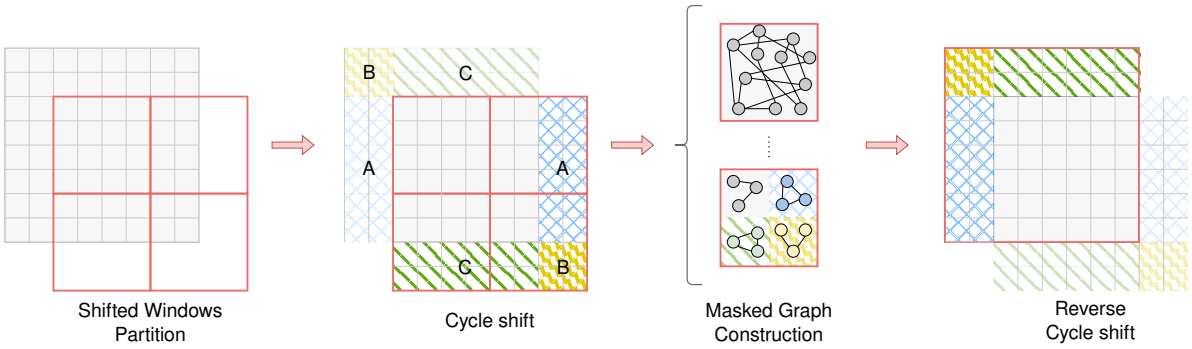


Figure 3. Overview of the cycling operation used to obtain shifted windows. The top-left part of the feature maps is copied on the bottom-right part, then the masking mechanism is used to avoid connection between non-adjacent nodes in the original feature maps.

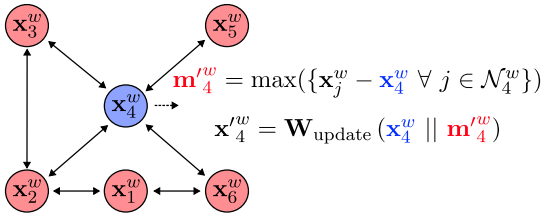


Figure 4. Illustrative example of the dynamic graph convolution of WiGNet.

(2) for the 4-th node, where $\mathcal{N}_4^w = \{2, 3, 5, 6\}$, we represent the aggregation step in (1) as $\mathbf{m'}_4^w = \max(\{\mathbf{x}_j^w - \mathbf{x}_4^w \; \forall \; j \in \mathcal{N}_4^w\})$, and the update step as $\mathbf{x'}_4^w = \mathbf{W}_{\text{update}} \left(\mathbf{x}_4^w \; || \; \mathbf{m'}_4^w\right)$. For simplicity, we omit the window notation and refer to the graph convolution in (2) as $\mathbf{X}' = \text{GraphConv}(\mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{N \times F} = [\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N]^\top$ is the set of node features.

GNNs typically include few graph convolutional layers due to the over-smoothing problem [9], where features tend to be more and more similar and thus less discriminative with the network depth. For this reason, we employ an FFN to perform feature transformations and non-linear activations after the Window-based Grapher module. Our Grapher also has a fully connected layer before and after the dynamic graph convolution layer as in [11]. Besides, the graph representations are dynamically updated with k-NN in every new layer as in [23]. Therefore, given an input feature $\mathbf{X} \in \mathbb{R}^{N \times F}$, the overall Window-based Grapher module transfer function can be expressed as:

$$\mathbf{Y} = \sigma\left(\text{GraphConv}(\mathbf{X}\mathbf{W}_{\text{in}})\right)\mathbf{W}_{\text{out}} + \mathbf{X}, \tag{3}$$

where $\mathbf{W}_{\text{in}} \in \mathbb{R}^{F \times 2F}$ and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{2F \times F}$ are learnable parameters of fully-connected layers that respectively increase and reduce the input feature dimension F, and $\sigma(\cdot)$ is a non-linear activation function. We omit the bias terms and batch normalization in (3) for the sake of simplicity. The operation in (3) results in a new feature embedding $\mathbf{Y} \in \mathbb{R}^{N \times F}$.

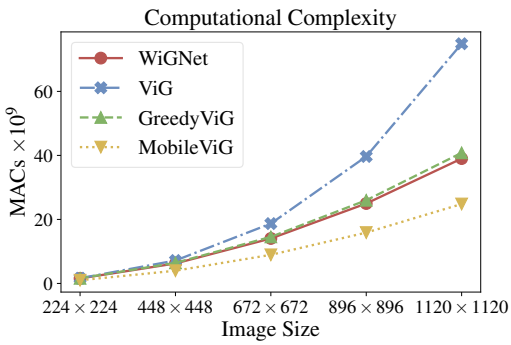
The FNN module. To encourage feature diversity, the output $\mathbf{Y}$ of the Grapher module is processed by the FFN module as follows. The FFN module is implemented as a multi-layer perceptron with two fully connected layers with residual connection to $\mathbf{Y}$ given by:

$$\mathbf{Z} = \sigma(\mathbf{Y}\mathbf{W}_1)\mathbf{W}_2 + \mathbf{Y}, \tag{4}$$

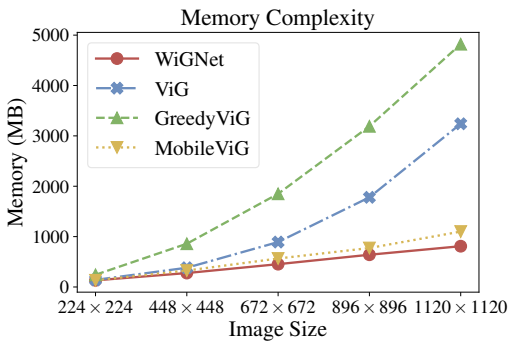
where $\mathbf{W}_1 \in \mathbb{R}^{F \times p}$ projects the input features into an p-dimensional space, with $p = F \times E$ and E the hidden dimension ratio, and $\mathbf{W}_2 \in \mathbb{R}^{p \times F}$ re-projects the features into the original F-dimensional space. The FFN module in (4) also contains: (i) batch normalization layer after each linear projection $\mathbf{W}_1$ and $\mathbf{W}_2$, and (ii) bias terms that we omit for the sake of simplicity.

3.3. Shifted Windows.

To introduce cross-window connections while maintaining the efficient computation described above, we include in WiGNet a shifting operator similar to the one adopted in Swin Transfomer [24]. More precisely, we implement a Shifted Window-based Grapher module, where the graph construction and convolution are performed on shifted windows as illustrated in Fig. 3. To do this, we adopt a cycling operation to partition the feature map, and we use a masking mechanism to allow connection only between nodes adjacent in the feature map. In other words, multiple subgraphs may arise in the same window as shown in Fig. 3, where different colors and texture backgrounds are used to identify the masking mechanism (*i.e.*, connections are allowed only between nodes that fall in the same color area). This phenomenon results in a heterogeneous construction of the graphs, implying a considerable drop in the number of neighboring nodes in certain regions. For instance, a node belonging to the top-left window of Fig. 3 will be connected to k other nodes in that window out of $M \times M$ possible nodes, where M is the window size. Instead, a node in the section *B* of the bottom-right window will still be connected to k other nodes but out of $S \times S$ possible nodes, where S is the shift-size typically set as $S = \lfloor \frac{M}{2} \rfloor$. To attempt to solve



(a) Computational complexity in MACs.



(b) Memory complexity in MegaBytes (MB).

Figure 5. Computational complexity and GPU memory footprint of several vision GNN architectures and WiGNet in terms of MACs and MB on NVIDIA GeForce RTX 3090 GPU.

this issue, we linearly adjust the number of neighbors of each node by considering the maximum number of possible neighbors that the masking mechanism allows it to have. In particular, given k, the window-size $M \times M$, and the number of possible neighbors for the node i (P_i), we can use $k_i = k \times \frac{P_i}{M^2}$ as the number of neighbors for that node.

3.4. Complexity Considerations.

Although both our method and ViG use k-NN to create the graph, one of the major advantages of WiGNet is the reduction in computational complexity as the image size increases. ViG's k-NN complexity, indeed, grows with the square of the number hw of nodes (patches) of the whole feature map and is given by:

$$\Omega\left(\text{ViG, } k\text{-NN}\right) = (hw)^2. \tag{5}$$

In contrast, the windowed approach of WiGNet results in a complexity that grows linearly with the number of patches as follows:

$$\Omega(\text{WiGNet, } k\text{-NN}) = \left(\frac{hw}{|\mathcal{V}^w|}\right)|\mathcal{V}^w|^2 = hw|\mathcal{V}^w|, \tag{6}$$

where $|\mathcal{V}^w|$ is the number of nodes on each window w. We compare Multiply–Accumulate (MACs) operations and memory footprint of ViG, WiGNet and two other graph-based models in Fig. 5.

4. Experiments

In this section, we first experiment with WiGNet over the ImageNet-1K [33] dataset comparing against ResNet [14], Pyramid Vision Tranformer [39], Swin Transformers [24], Poolformer [44], ViG [11], MobileViG [11], and GreedyViG [12]. For the sake of comparability, we consider $k = 9$ neighbors for graph construction as in ViG [11]. Then, once we train our model on ImageNet, we evaluate its adaptability to a new classification task with higher-resolution images. To do this, we use our *tiny* model as a pre-trained frozen backbone for facial identification on the CelebA-HQ [22] dataset. Finally, we perform two ablation studies on key design choices: whether or not to use shifting windows and which graph convolutional layer to adopt.

4.1. Experimental Setup

Datasets. In image classification, the benchmark dataset ImageNet ILSVRC 2012 [33] is commonly used as a standard evaluation metric. ImageNet contains approximately 1.2 million in training images and $50,000$ in validation images, spanning across $1,000$ categories[1].

The CelebA-HQ [22] dataset is instead used to test the adaptability of our model in a downstream task with high-resolution images. Indeed, this dataset is a high-quality version of CelebA [25] that consists of $30,000$ images. We use this dataset to perform a facial identification of 307 classes by rescaling the images to a resolution of 512×512. This rescaling is performed to be able to train the most memory-intensive models like ViG and GreedyViG.

Implementation details. For training all WiGNet models on ImageNet we keep similar hyperparameters as ViG [11]. We adopt the commonly-used training strategy proposed in DeiT [36] for fair comparison. The data augmentation includes RandAugment [4], Mixup [46], Cutmix [45], random erasing [47]. Additionally, for WiGNet-M we adopt the repeated augmentation [15] and an Exponential Moving Average (EMA) scheme. We implement our models using PyTroch [29] and train all of them on 8 GPUs NVIDIA GeForce RTX 3090.

Then, once we obtain our pre-trained models, we perform a transfer-learning experiment on higher-resolution images. In this context, all models are finetuned using Adam as an optimizer having a constant learning rate of 0.001 for 30 epochs using a Cross-Entropy loss function and

[1]For information on the licensing of the ImageNet dataset, please refer to the website http://www.image-net.org/download.

Model	Resolution	Params (M)	MACs (G)	Top-1	Top-5
♠ ResNet-18 [14, 42]	224×224	12	1.8	70.6	89.7
♠ ResNet-50 [14, 42]	224×224	25.6	4.1	79.8	95.0
♠ ResNet-152 [14, 42]	224×224	60.2	11.5	81.8	95.9
♦ PVT-Tiny [39]	224×224	13.2	1.9	75.1	-
♦ PVT-Small [39]	224×224	24.5	3.8	79.8	-
♦ PVT-Medium [39]	224×224	44.2	6.7	81.2	-
♦ PVT-Large [39]	224×224	61.4	9.8	81.7	-
♦ Swin-T [24]	224×224	29	4.5	81.3	95.5
♦ Swin-S [24]	224×224	50	8.7	83.0	96.2
■ Poolformer-S12 [44]	224×224	12	2.0	77.2	93.5
■ Poolformer-S36 [44]	224×224	31	5.2	81.4	95.5
■ Poolformer-M48 [44]	224×224	73	11.9	82.5	96.0
♣ ViG-Ti [11]	224×224	10.7	1.7	78.2	94.2
♣ ViG-S [11]	224×224	27.3	4.6	82.1	96.0
♣ ViG-M [11]	224×224	51.7	8.9	83.1	96.4
✠ MobileViG-Ti [27]	224×224	5.2	0.7	75.7	-
✠ MobileViG-S [27]	224×224	7.2	1.0	78.2	-
✠ MobileViG-M [27]	224×224	14.0	1.5	80.6	-
▼ GreedyViG-S [28]	224×224	12.0	1.6	81.1	-
▼ GreedyViG-M [28]	224×224	5.2	3.2	82.9	-
★ WiGNet-Ti (ours)	224×224	10.7	1.6	78.4	94.3
★ WiGNet-Ti (ours)	256×256	10.8	2.1	78.8	94.6
★ WiGNet-S (ours)	256×256	27.4	5.7	82.0	95.9
★ WiGNet-M (ours)	256×256	49.7	11.2	83.0	96.3

Table 2. Results of WiGNet and other deep learning methods on ImageNet. ♠ CNN, ■ MLP, ♦ Transformers, ♣ ViG, ✠ MobileViG, ▼ GreedyViG and ★ WiGNet (ours).

a batch size of 64, except for GreedyViG and ViG where we used a batch-size of 16 for memory reasons.

4.2. Main Results

First, we provide the classification results on ImageNet. Then, we show that our pre-trained backbone achieves a better trade-off between accuracy and complexity than other models using CelebA-HQ as a downstream task.

ImageNet. Table 2 shows the comparison between WiGNet and previous state-of-the-art (SOTA) methods. WiGNet outperforms or achieves competitive results against previous SOTA models for similar complexity. For instance, comparing WiGNet with non-graph-based models, our tiny model with 78.8 of accuracy outperforms all previous methods for low MACs (around 2G), and a small number of parameters (around 10M). Similarly, the WiGNet-S achieves better results than the Swin-T model with comparable MACs and than ResNet-152 with almost half the parameters. In addition, WiGNet shows competitive results against the previous graph-based method under similar conditions. Moreover WiGNet-Ti trained with slightly larger images (256×256) and using a window size of 8×8, works better than the same model trained on (224×224) images since in this case the window-size is smaller (7×7) and thus the number of possible neighbors.

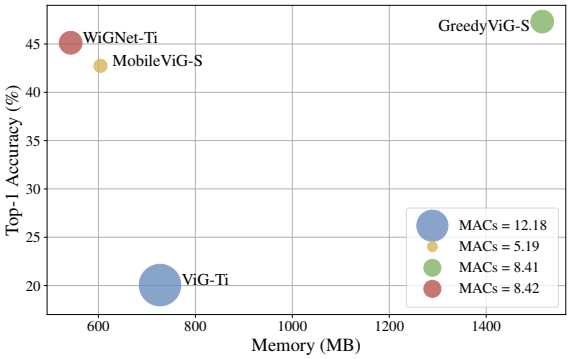


Figure 6. Comparison of Graph-based models on 512×512 resolution images. The size of the dots represents the used MACs.

CelebA-HQ. To show the adaptability of WiGNet to new classification tasks having higher resolution images, we conduct experiments using our pre-trained model on ImageNet as a frozen backbone on the CelebA-HQ dataset [22] as a downstream facial identity classification task. Particularly, a new classification layer was trained on these features keeping the rest of the architecture frozen. Fig. 6 shows the results obtained by our backbone compared to other graph-based models in terms of accuracy, memory usage, and MACs using 512×512 resolution images. Here we notice that ViG struggles to converge, while our backbone achieves the second-best result, only outperformed by GreedyViG. However, by comparing the memory footprint we observe that WiGNet needs only 0.5 GB, while for GreedyViG the occupancy is $\sim 3\times$ more. MobileViG, instead, is the model with lowest MACs. Nevertheless, it occupies more memory than WiGNet achieving worst results.

It is clear from Fig. 6 that WiGNet is the closest model to the optimal point (*i.e.* top-left corner of the plot), achieving similar Top-1 accuracy results to GreedyViG but using significantly less memory, even compared to MobileViG. In Fig. 5 we also analyzed the complexity of these models in terms of MACs and memory as the resolution of the input image increases. From these results we observe that WiGNet computation and memory requirements scale only linearly with the image size. By comparison, the memory requirements for GreedyViG (and for ViG also the complexity in terms of MACs) scales quadratically with the image size. These results show that WiGNet can operate with images of high resolution using less memory than MobileViG while maintaining the complexity under control.

4.3. Ablation Studies

Shifted windows. Table 3 shows the results when the WiGNet uses the Shifted Window-based Grapher module explained in Sec. 3.3. We observe that contrary to the Swin Transformer, the shifting strategy does not bring any advantage to WiGNet in this context, despite the results

Model	Resolution	Shifting	Top-1
WiGNet-Ti	256×256	✗	78.9
WiGNet-Ti	256×256	✓	78.8
WiGNet-S	256×256	✗	82.0
WiGNet-S	256×256	✓	82.0
WiGNet-M	256×256	✗	82.9
WiGNet-M	256×256	✓	83.0

Table 3. Ablation study in the impact of the shifting operation and the adaptive k-NN strategy for WiGNet on ImageNet.

Model	Resolution	Shifting	Adaptive-k-NN	Top-1
WiGNet-Ti	512×512	✗	-	39.48 ($\pm$ 6.02)
WiGNet-Ti	512×512	✓	✗	41.15 ($\pm$ 0.65)
WiGNet-Ti	512×512	✓	✓	45.13 ($\pm$ 1.73)

Table 4. Ablation study in the impact of the shifting operation for higher resolution images from the CelebA-HQ dataset.

seeming to improve slightly by increasing the model size. We hypothesize that, because of the low resolution of the images in ImageNet, is possible to independently analyze the windows and still obtain good results. Therefore, we conduct the same transfer learning experiment described in Sec. 4.2 to monitor the behavior of our backbone without shifting and with higher-resolution images, ablating also on the Adaptive k-NN strategy. In Tab. 4 we observe that for larger images the shifting operator is crucial, allowing for a gain of almost 2% points on average, and a significantly lower standard deviation. Moreover, this gain increases to 6% when the Adaptive k-NN strategy is implemented.

Graph Convolutional Operator. Finally, we conduct an ablation study with some well-known graph convolutional functions in the Grapher module, including Max-Relative GraphConv [23], GraphSAGE [10] and EdgeConv [41]. Table 5 shows the results of this experiment when the WiGNet-Ti model is trained on ImageNet without the shifting operator, as it seems to work slightly better for the tiny size model. We observe that the Max-Relative graph convolution achieves competitive results with less complexity.

4.4. Limitations

The main limitation of our method compared to other Graph-based approaches is the lack of global information during the feature update process. This problem is partially mitigated by the hierarchical structure of the architecture and the shifting operation, which allows to capture less local (but not global) information through cross-windows connections. However, this operation in WiGNet is not as straightforward as in Swin Transformers: we should dynamically adapt the k value for the k-NN in the borders of the image to be consistent with the rest of the regions as shown in Fig. 3. By adopting this strategy, we successfully classify high-resolution images. Nevertheless, we believe that more global information might be useful in tasks where we need to capture long-range dependencies like, for example, image segmentation. One possible solution to this limitation is to promote connections among graphs in the same layer. However, this solution poses practical and theoretical challenges that deserve exploration in future works.

Model	Graph-Conv	MACs (G)	Top-1	Top-5
WiGNet-Ti	Max-Relative	2.1	78.9	94.6
WiGNet-Ti	GraphSAGE	2.4	78.4	94.3
WiGNet-Ti	EdgeConv	3.3	78.7	94.5

Table 5. ImageNet results using different Graph Convolutional layers. Comparison performed on the *tiny* model size without the shifting operator.

5. Conclusions

This work introduced a new Windowed vision GNN (WiGNet) for image analysis tasks. Our model partitions the input images into windows, and therefore graphs are constructed in these local windows. Thus, we use the Max-Relative graph convolution operation on each window for feature updating. We show in theory and practice that the computational and memory complexity of WiGNet scales linearly with the image size. At the same time, for previous vision GNNs such as ViG [11], the complexity grows quadratically. This has profound implications for GNN-based vision models' applicability in tasks requiring high-resolution images. We conducted experiments in the ImageNet-1k benchmark dataset and then we show that WiGNet can be successfully adopted as a pre-trained backbone for high-resolution image classification on the CelebA-HQ dataset, achieving a better trade-off between accuracy and complexity with respect to other graph-based models. Thus, WiGNet offers a strong and scalable alternative to previous models for computer vision tasks, proving to be suitable for working with high-resolution images.

Acknowledgements

This work has been carried out at the Energy4Climate Interdisciplinary Center (E4C) of IP Paris and Ecole Nationale des Ponts et Chaussées, which is in part supported by 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02], and by the Foundation of Ecole Polytechnique with the private sponsor Fonds Ifker pour le Climat, financed by Stéphane y Agnès Ifker. This research was also partially funded by Hi! PARIS Center on Data Analytics and Artificial Intelligence, and was provided with computer and storage resources by GENCI at IDRIS thanks to the grant 2024-AD011015338 on the supercomputer Jean Zay.

References

[1] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *ICLR*, 2017. 1, 2

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014. 2

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1, 2

[4] 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. 6

[5] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016. 2

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 1, 2, 3

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017. 3

[8] Jhony H Giraldo et al. Hypergraph convolutional networks for weakly-supervised semantic segmentation. In *ICIP*, 2022. 3

[9] Jhony H Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D Malliaros. On the trade-off between over-smoothing and over-squashing in deep graph neural networks. In *CIKM*, 2023. 5

[10] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017. 8

[11] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision GNN: An image is worth graph of nodes. In *NeurIPS*, 2022. 1, 2, 3, 4, 5, 6, 7, 8

[12] Yan Han, Peihao Wang, Souvik Kundu, Ying Ding, and Zhangyang Wang. Vision HGNN: An image is more than a graph of nodes. In *ICCV*, 2023. 6

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 6, 7

[15] 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. 6

[16] 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*, 2017. 2

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2, 3

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 2

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[21] 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. 1

[22] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *CVPR*, 2020. 6, 7

[23] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *ICCV*, 2019. 4, 5, 8

[24] 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*, 2021. 1, 2, 5, 6, 7

[25] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 6

[26] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *NeurIPS*, 2018. 2

[27] Mustafa Munir, William Avery, and Radu Marculescu. Mobilevig: Graph-based sparse attention for mobile vision applications. In *CVPR Workshop*, 2023. 1, 3, 7

[28] Mustafa Munir, William Avery, Md Mostafijur Rahman, and Radu Marculescu. Greedyvig: Dynamic axial graph construction for efficient vision gnns. In *CVPR*, 2024. 1, 3, 7

[29] 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. In *NeurIPS*, 2019. 6

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1, 2

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 1, 2

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 1, 2

[33] 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. *IJCV*, 115(3):211–252, 2015. 6

[34] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2

[35] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2

[36] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 1, 6

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018. 2

[39] 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. 6, 7

[40] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 2

[41] 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 TOG*, 38(5):1–12, 2019. 8

[42] Ross Wightman, Hugo Touvron, and Herve Jegou. ResNet strikes back: An improved training procedure in timm. In *NeurIPS*, 2021. 7

[43] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE T-NNLS*, 32(1):4–24, 2020. 1

[44] 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. 6, 7

[45] 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. 6

[46] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 6

[47] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020. 6