Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

SViG: A Similarity-thresholded Approach for Vision Graph Neural Networks

ISMAEL ELSHARKAWI, HOSSAM SHARARA, AND AHMED RAFEA
Department of Computer Science and Engineering, The American University in Cairo, New Cairo 11835 Egypt
Corresponding author: Ismael Elsharkawi (ismaelelsharkawi@aucegypt.edu).

ABSTRACT Image representation in computer vision is a long-standing problem that has a great effect on the performance of a model. The traditional methods rely on treating the image as a grid, then using Convolutional Neural Networks (CNNs) to generate an image representation. Afterwards, Vision Transformers and MLP-Mixers were introduced to represent an image as a sequence of features similar to other sources of data. More recently, Vision Graph Neural Network (ViG) have proposed the treatment of an image as a graph of nodes. Nevertheless, the graph construction in ViG depends on k-nearest neighbours (k-nn). This leads to a number of challenges; first, determining the right value for the k in every layer, and more importantly, that this value is the same for all nodes, which leads to limiting the graph expressiveness in capturing the image details, as some parts of the image can be similar to a larger number of other parts, while for other parts, we might be forcing completely irrelevant neighbours to satisfy the k-nn condition. In this work, we propose a new approach that relies on the similarity score thresholding to create the graph edges and, subsequently, the neighboring nodes. Rather than the number of neighbours, we allow the specification of the normalized similarity threshold as an input parameter for each layer, which is more intuitive. We also propose a decreasing threshold framework for selecting the input threshold for all layers. We show that our proposed method is able to achieve higher performance than the ViG model for image classification on the benchmark ImageNet-1K dataset, without increasing the complexity of the model.

INDEX TERMS Graph Neural Networks, Vision Graph Neural Networks, Image Classification

I. INTRODUCTION

In computer vision, a number of different types of backbone models have been introduced in the literature for large datasets. The purpose of these backbones is to have a pre-trained model that could be fine-tuned for different downstream tasks. The most famous task used during the pre-training of a model is image classification on ImageNet-1K [1]. The challenge is to represent an image in the best possible method. The most prominent 3 methods are Convolution Neural Networks (CNN) [2], Vision Transformers (ViT) [3], and Multi-Layer Perceptron Mixers (MLP-Mixers) [4]. Recently, Vision Graph Neural Networks (ViG) were introduced as well to tackle this task. Figure 1 shows a comparison between the different ways of representing an image.

CNNs apply moving learnable filters to images with an increasing depth. By treating an image as a grid, CNNs provide spatial locality and shift in-variance. AlexNet [5] was among the first attempts for using CNNs for the classification of ImageNet [1] images.

More recently, the emergence of Vision Transformers [3] and MLP-Mixers [4] proposed the treatment of images as *patches*, which means dividing an image into a grid of equal-sized squares, then treating them as a sequence. Similar to the core idea of NLP Transformers [6], Vision Transformers treat an image as a sequence of patches and allow the patches to *attend* to one another, however, that comes at the cost of a large number of parameters and Floating Point Operations (FLOPs). MLP-Mixers rather treat an image as a *table* of "*patches* x *channels*", then use channel-mixing MLP layers in addition to patch-mixing MLP layer. Each of those "mixing" layers is a transpose operation, followed by an MLP layer for each dimension.

In order to have a more generic relation between patches, Vision Graph Neural Networks (ViG) [7] proposes treating an image as a *graph* of patches. Representing the image as a graph does not limit the receptive field of a node in the graph. In addition, it is a more flexible and effective method to represent the relationships between patches. However, a graph requires a different type of ML algorithms than structured data, such as images and text. Graph ML is traditionally used for data that is naturally represented as a graph. Examples of these include social networks [8], citation networks [9] and biochemical graphs [10].

When it comes to computer vision, the use of Graph ML has been proposed for 3D computer vision tasks, such as scene graph generation, point cloud classification and segmentation. A recent example for the point cloud classification and segmentation is DGCNN [11], which used k-nearest neighbours (k-nn) to construct the graph of points in the point cloud in addition to proposing EdgeConv, which is a type of graph convolution (GCN) [12] layer tailored for point cloud data. Following from that, DeepGCNs [13] proposed the use of linear layers before and after a GCN in order to fight oversmoothing. Oversmoothing is the phenomenon where the representation of nodes in a graph become similar to each other after multiple graph convolutions. DeepGCNs also proposed a different type of graph convolution operation, which is Max-Relative Graph Convolution. In addition, they proposed the use of dilated k-nn convolution, where for each node, the set of $k * d$ nearest neighbors (or most similar nodes) is picked. Then k neighbors are chosen in a stochastic manner from that set. ViG [7] adopted a very similar methodology to the one proposed by DeepGCNs, however, they applied it to 2D images to be able to have a generic graph-based framework for 2D computer vision tasks. Rather than a graph of point clouds, an image is treated as a graph of patches.

The backbone of a ViG is composed of a cascade of a type of GCN [12] layers, called Grapher modules, where the graph is constructed based on the similarity matrix of the patches (or nodes). The receptive field of a node is dependant on feature extractor, called the Stem module, and the input image resolution. The graph is reconstructed in each GCN layer, where the neighborhood of each node is the k-closest nodes to it in the latent space of that layer.

It is important to note that the number of patches *might* be different in each layer, depending on the architecture of the network. That means that the receptive field of each node (or a patch) might be different in each layer. That difference is only there in pyramid architectures, but not the traditional isotropic architectures. In isotropic architectures, the number of nodes is kept constant across all layers, whereas in pyramid architectures, the number of nodes is lowered as the network goes deeper.

The methodology in ViG [7] has a major drawback: each node might pick irrelevant nodes as neighbors because the k is fixed by definition and *popular* nodes might have a fewer number of neighbors than they should, which leads to missing important information in the aggregation operation. To address this drawback, we propose a new similarity-thresholding approach which focus on picking the neighboring nodes based on a similarity threshold, which then allows each node to have different numbers of neighbours in the graph based on its properties. In this paper, we will focus on testing our approach on the tiny version of the isotropic ViG. While trying different model sizes, different pyramid architectures might yield better performance, we leave the Neural Architectural Search (NAS) task for future work. **Our contribution can be summarized as follows:**

- We propose a more robust and flexible approach for graph construction without introducing a computational overhead or making assumptions about the topology of the constructed graph. Unlike the k-nn graph construction in ViG, we simply pick edges for the whole graph based on their corresponding normalized similarity score and a threshold. Moreover, our approach fights oversmoothing by introducing a decrement in the thresholds across layers.
- We achieve a Top-1 validation accuracy of 74.6% on ImageNet-1K, a 0.7% improvement compared to the ViG-Ti baseline [7].
- Our implementation is generic and does not force picking the exact same number of neighbors for each node in a graph in the latent space. It can also operate on any input adjacency matrix without introducing a memory or a computational overhead.

The structure of this paper is as follows: we first carry out a literature review in section II, then we explain our proposed methodology in section III and how to construct a graph using similarity-thresholding. Section IV discusses the experimental setup and the experimental results, mainly Top-1 and Top-5 accuracies of the GNN models on ImageNet [1]. In section V, we summarize our contributions again and highlight future directions of work.

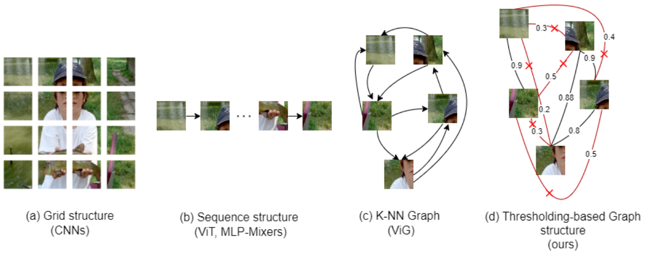
0.3
0.4
0.9
0.9
0.5
0.88
0.2
0.3
0.8
0.5
(a) Grid structure (CNNs)
(b) Sequence structure (ViT, MLP-Mixers)
(c) K-NN Graph (ViG)
(d) Thresholding-based Graph structure (ours)

FIGURE 1. Different image representations (a) CNN Grid Structure. (b) Vision Transformers Sequeunce Structure. (c) k-nn Vision GNN (ViG) Graph Structure, k = 2. (d) Our proposed Graph Structure (SViG): similarity-thresholded graph construction for a threshold = 0.8. (The normalized similarity scores on edges are hypothetical and self-edges are ignored).

II. RELATED WORK

Since the introduction of Vision Graph Neural Networks [7], there have been two types of improvement proposed to ViG: proposing changes in the graph construction methodology and proposing new hybrid CNN-GNN model architecture to have the best of both worlds. This section is going to focus on the graph construction methodology. We leave the Neural Architectural Search (NAS) task for future work.

To solve the graph construction problem, ViGHNN [14] proposed the use of a *hyper-graph* rather than the use of a graph. A hyper-graph means that having clusters of nodes in the graph, such that: for each cluster, message passing is done from all nodes in the cluster to a virtual *super-node* to represent each cluster. This is followed by another message passing from that super-node to the nodes in the cluster. They use Fuzzy C-Means to create clusters of patches, where the clusters can have overlapping nodes. The drawback of that approach is that the clustering of nodes is computationally expensive.

Another method proposed for graph construction in MobileViG [15] was to create an edge between each node and the nodes on the *dilated axes* of that node. A dilated axis is the set of every other node on the row or the column to which that node belongs. Nevertheless, the approach of MobileViG, while efficient, aggregates noisy information from irrelevant neighbors to each node. In order to limit the number of edges picked for each node, GreedyViG [16] proposed using dynamic axial graph construction, which means that each node will pick neighbors on its dilated axes only if they have a euclidean distance smaller than $\mu - \sigma$, where both μ and σ are estimated values for each dilated axis. While this method is more efficient than the one used in MobileViG, important neighbors might still be dropped simply because they are not on the dilated axes of a node. In addition, this method of graph construction makes a strong assumption that important neighbors strictly have a distance that is smaller than the $\mu - \sigma$ for all Grapher modules, which is not always the case. Likewise, MobileViGv2 [17] proposed picking a pre-defined number of neighbors from the nodes on each dilated axis, where that number would be a hyper-parameter. Nonetheless, the graph construction method proposed by MobileViGv2 is exactly like the k-nn graph construction, except for the fact that neighbors have to be on the dilated axes.

All of the previously mentioned models were trained on the ImageNet-1K image classification task. Another Computer Vision task that could make use of variants of ViG is Super Resolution (SR), which is the task of creating a high resolution image given a low resolution one. Image Processing GNNs (IPG) [18] used a variant of ViG for SR. For each patch, IPG performs a *local* search neighborhood (the 8 surrounding patches) and a *global* search (a grid that skips every other column and row of the patch grid). For each patch, IPG determines the *importance* of each patch depending on the detail retained in the image when down sampling the patch, then upsampling it again. This *importance* of a patch is used to determine the degree of the node. The method IPG uses has 3 drawbacks. First, it is specifically tailored for the SR task, because the node degree is determined based on the effort needed to reconstruct that node when downsampled. Second, determining the degree-node for each node, then picking the top-k neighbors for each node is computationally expensive. Third, the search space of the neighbors of any node should include all the patches in an image, rather than a subset, to avoid limiting the receptive field of a node.

In order to address the aforementioned shortcomings, we propose a flexible, yet simple, method of graph construction, where the neighborhood of a node could include any other node in the graph. In addition, our proposed method does not require heavy computation for clustering nodes and it is well-suited to a wide range of tasks, and is not tailored for a specific task.

III. METHODOLOGY

This section describes the notations used, the architecture of our proposed Similarity-thresholded Vision Graph Neural Network (SViG), then the method of the graph construction in each layer, and finally the flexible aggregation framework we use. In our work, we only focus on the tiny isotropic ViG architecture.

A. NOTATIONS

An input image $I \in \mathbb{R}^{H\times W\times 3}$ is divided into N nodes using the stem module. The input data matrix to a Grapher module l is $X^l = [x_1^l, x_2^l, x_3^l, ..., x_N^l]$, where $1 \leq l \leq L_G$ and L_G is the number of Grapher modules in the backbone. Each node feature vector $x_i^l \in \mathbb{R}^D$, where D is the dimension of the embedding of each node. A graph $G = (V, E)$ is composed of a set of nodes V and the edges E that connect those nodes. Each node v_i has an associated feature vector x_i^l for each Grapher layer l. The edge between two nodes u and v is denoted by $e_{u,v}$. The distance between the representations of two nodes u and v is denoted as $d(u, v)$ and the similarity is denoted as $s(u, v)$.

B. OVERVIEW: SVIG-TI

The architecture of the SViG-Ti is adopted from ViG-Ti [7]. A ViG consists of a stem module, followed by the backbone, then an average 2D pooling layer, then a prediction head. The backbone is a sequence of Grapher modules followed by Feed-Forward Networks (FFN). This architecture is shown in figure 2.

A stem module is a Convolution Neural Network (CNN) module that is used to extract features from an image. The output of the Stem module is $\frac{H}{16} \times \frac{W}{16} \times D$, where D is the number of filters of the last convolution operation. This output can be regarded as a set of N nodes ($N = \frac{H}{16} \times \frac{W}{16}$) of the graph G in the first Grapher layer. Thus, each node feature vector has dimensions $1 \times 1 \times D$ and the number of nodes in the graph is $\frac{H}{16} \times \frac{W}{16}$. Following the stem module we add learnable positional embeddings.

A Grapher module is composed of a fully connected layer, followed by a GCN layer, followed by a concatenation with the input feature vector, then a fully connected layer and finally, another fully connected layer. MaxRelative convolution [13] is the GCN operation. A Grapher module is described by the following set of equations:

$$\begin{aligned}
{h_i}^1 &= {x_i}^l W_{in} \\
{h_i}^2 &= \max(h_u - {h_i}^1 | u \in Neighborhood(v_i)) \\
{h_i}^3 &= \sigma([{h_i}^1, {h_i}^2] W_{update}) W_{out} \\
{x_i}^{l+1} &= {h_n}^3 + x_i^l
\end{aligned} \tag{1}$$

where W_{update}, W_{in} and W_{out} are learnable weights and ${h_i}^k$ is the intermediate representation of node v_i in that layer. The second linear layer in the Grapher module (W_{update}) is a multi-head update operation. The activation function used is GELU [19]. Figure 3 shows a Grapher module architecture.

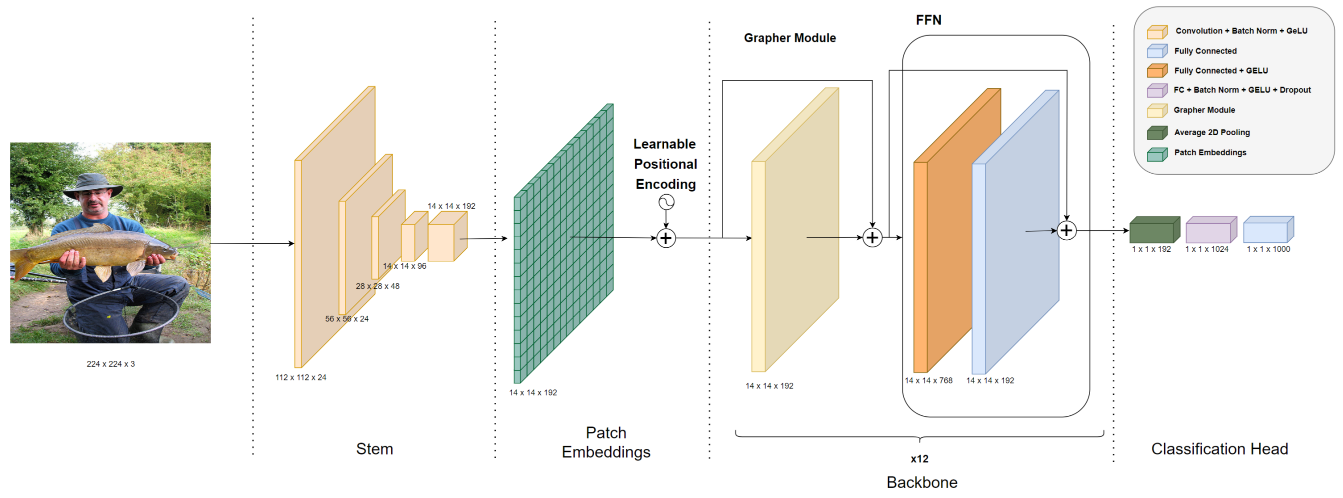

FIGURE 2. ***SViG-Ti Model Architecture*: This figure shows the architecture of SViG-Ti, which is composed of a stem module to construct the embedding of the nodes. For a 224 $\times$ 224 $\times$ 3 image, each node in the graph represents a 19x19 patch in the image. The backbone is a cascade of Grapher and FFN modules. The final part of the model is an average 2D pooling layer and a classification head that outputs a classification score for each of the 1000 classes in ImageNet.**

A Feed-Forward Network (FFN) is intended to increase the feature diversity and fight oversmoothing. An FFN is composed of 2 linear layers with a GELU activation function in between, in addition to a residual connection. An FFN is described by the following equation:

$$Z = \sigma(YW_1)W_2 + Y \tag{2}$$

where $Y, Z \in \mathbb{R}^{N\times D}$, Y is the output of the Grapher module and Z is the input to the next Grapher module.

C. SVIG-TI ARCHITECTURE

We focus on the isotropic architecture rather than the pyramid architecture in our work. To be able to check the effect of the graph construction independently, we adopt the smallest isotropic architecture suggested by [7], which is ViG-Ti. ViG-Ti has 12 Grapher modules. The Stem module has 5 convolution layers. The number of multi-heads in the second linear layer of the Grapher module (W_{update}) is taken as 4. In this work, the downstream task used to evaluate the performance of the model is image classification. The classification head is composed of 2 fully connected layers.

In our work, we pick $D = 192$. We resize all input images to $224 \times 224 \times 3$. The size of the patch embeddings (after the Stem module) is $14 \times 14 \times 192$. This means that the receptive field of each node, or the patch size, is 19×19 in the image. Thus, a graph would have 196 nodes. A node feature vector $x_i^l \in \mathbb{R}^{192}$ is the feature vector of node v_i in Grapher layer l.

D. GRAPH CONSTRUCTION

In ViG [7], the graph was reconstructed dynamically in every Grapher module. This construction was based on the stochastic dilated k-nn selection of the neighborhood of each node in the latent space. However, this methodology makes a strong assumption about the graph topology; it forces each node to select the exact same number of neighbors, which might lead to noise in the aggregation operation in addition to missing out on important neighbors in for other nodes. To alleviate this issue, we use a similarity-thresholded approach to allow each node to pick a variable number of neighbors based on how similar they are to it. This allows a more realistic graph construction process.

First, we compute the pairwise distances between every pair of nodes in an image as follows:

$$d(v, u) = v \cdot v^T - 2v \cdot u + u \cdot u^T \tag{3}$$

The similarity score of an edge $e_{v,u}$ is $s(v, u) = -d(v, u)$. Assuming that the distribution of the similarity scores of edges in a fully connected graph follows a normal distribution, the similarity scores are normalized as follows:

$$s_{norm}(v, u) = \frac{s(v, u) - \mu_S}{\sigma_S} \tag{4}$$

where μ_S and σ_S are calculated for all edges in an image, and not per node. They are calculated dynamically during runtime, using the feature vector X^l for Grapher layer l independent from other Grapher layers.

A new hyper-parameter t_i is introduced. t_i represents the threshold at Grapher module i. The selection is made as follows:

$$\begin{aligned} &if \quad s_{norm}(v, u) > CDF^{-1}(t_i): \quad e_{v,u} = 1 \\ &else: \quad e_{v,u} = 0 \end{aligned} \tag{5}$$

where $CDF^{-1}(t_i)$ is the inverse of the CDF (cumulative distribution function) of the standard normal distribution at t_i. This means that an edge is picked if its normalized similarity score exceeds the threshold of that layer, where the threshold represents the CDF of the normal distribution (i.e. for a threshold = 0.9, top 10 percentile of all edges are picked,

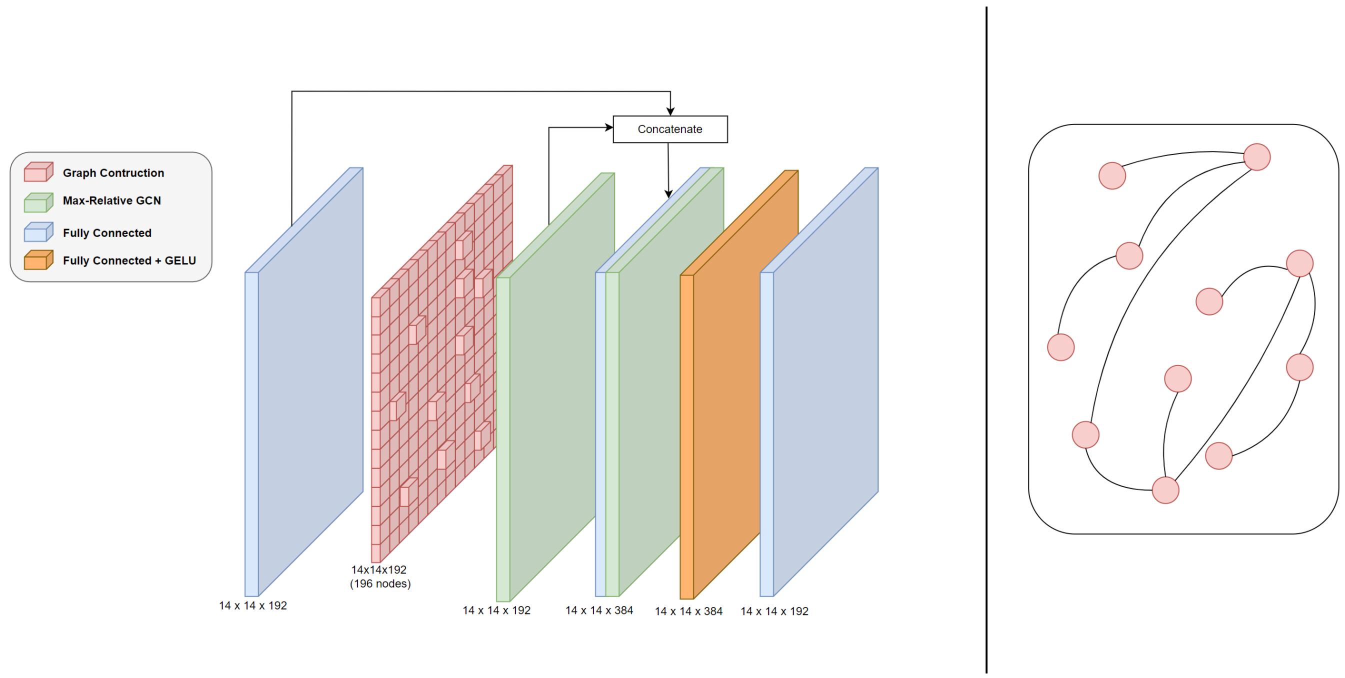

FIGURE 3. ***Left:*** **A Grapher Module is composed of a linear layer, followed by a graph construction layer, then a Graph Convolution layer, and finally 2 linear layers. For the graph construction layer, the graph topology is determined based on the normalized similarity score of the edge between every two nodes. Each square in the Graph Construction layer is the representation of one node ($1 \times 1 \times 192$) in the graph. Some nodes in the graph construction layer (in red) are chosen to show the graph construction process to the right.** ***Right:*** **An edge is constructed between two nodes if the normalized similarity score between those two nodes exceeds a predefined threshold t_i. Non-selected edges are not shown in this graph.**

assuming the distribution of the similarity scores follows a normal distribution).

This simple *thresholding* method ensures that an edge is chosen in both directions since $s(v, u) = s(u, v)$. It also makes sure that "popular" nodes have more incoming edges, which leads to a better representation after the aggregation. In addition, this *thresholding* method picks edges rather than picking neighbors for each node.

Due to the over-smoothing [20], node representations become similar in later layers. To overcome over-smoothing, the threshold t_i needs to be lower in later layers to have a larger neighborhood for each node and thus aggregate from nodes that are less similar. We choose to have a constant decrement for t_i, which we call d_t, where $t_i - t_{i+1} = d_t$. We show experimentally in section IV the importance of having a decrement d_t.

E. FLEXIBLE AGGREGATION

To implement a graph convolution layer, the tensor of the features of *source nodes* and *target nodes* are needed to perform the aggregation in equation 1. The representation of the tensor of the features of source or target nodes puts a constraint on the graph construction and aggregation, especially in the second step in equation 1.

When representing the neighbors of each node, ViG [7] had a fixed k neighbors for all nodes in an image, and all images in a batch. ViG creates a tensor of *self* nodes, and a tensor of *neighbor* nodes, to be able to do the subtraction operation in equation 1, rather than tensors of source and destination nodes. This means that in ViG [7], the tensors of the *self* nodes and *neighbor* nodes have dimensions $[B, N, D, k]$, where B is the batch size and k is the pre-defined number of neighbors for each node, used in the k-nn graph construction. ViG [7] performs a subtraction of the self and neighbor tensors, then a *max* operation on the last dimension representing the k neighbors, as a part of the MaxRelative Convolution1. This essentially restricts each node, in every image in the batch, to have exactly k neighbors.

To solve this issue in SViG, without adding an extra memory consumption overhead, we use one *global* graph. The construction of the global graph starts with the graph construction for each image individually. Afterwards, a global graph containing all graphs in a batch is created, where each graph, or image, is an isolated graph in that large graph. Figure 4 shows how the global graph construction is performed. This is inspired by the method of batch implementation in PyTorch Geometric (PyG) [21].

To implement the flexible graph construction, we reshape the input tensor, from $[B, N, D]$ to $[B * N, D]$. An edge index would have the dimensions $[2, E_{all}]$, with each pair determining the source and destination node indices. It is worth mentioning that the indices in this context are global graph indices (i.e. they range from 1 to $B * N$). Also, E_{all} is the number of all edges in the whole batch. The tensor of source nodes and destination nodes have dimensions $[E_{all}, D]$. We perform the subtraction using the source and destination indexed feature tensors. Then we perform the maxpooling operation in the aggregation using a *scatter* operation, that is available in PyG [21], and the target node indices. After the aggregation, the target tensor, with dimensions $[B*N, D]$, represents the features of all the nodes in all images in a batch. Then, it is reshaped to restore the batch dimensions (the dimensions are $[B, N, D]$). This tensor corresponds to h_i^2 in equation 1 for the whole batch. PyTorch-like code is shown in algorithm 1 for the flexible aggregation. With this flexible implementation, there is no constraint on the number of edges or the number of neighbors that each node can have. The aggregation operation can be performed using any input adjacency matrix, without having to mask a part of the data, and thus without introducing a memory overhead. In addition, as shown in section IV, our flexible aggregation does not introduce a computational overhead in terms of the number of FLOPs.

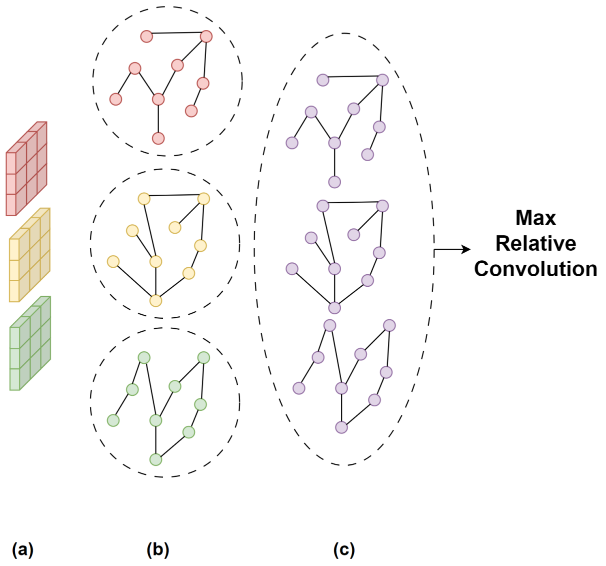

FIGURE 4. This figure shows the different stages for the flexible graph aggregation. (a) This is a batch of 3 images, where each image is represented by a 3 × 3 feature map. The total number of nodes per image is 9. (b) A graph is constructed for each image on its own, based on the similarity-threshold-ed graph construction. (c) The 3 graphs for the images are merged together into the 3 isolated sub-graphs within a larger graph.

IV. EXPERIMENTS

In this section, We show our results on the ImageNet image classification task and compare them with other models in the literature.

A. DATASET DESCRIPTION

In our experimental setup, we use ImageNet ILSVRC 2012 dataset [1]. The dataset license is available at https://www.image-net.org/download. This dataset has 1000 classes with 1.2 million training images and 50k testing images.

B. EXPERIMENTAL SETUP

We implement our model using PyTorch [22], Timm [23] and PyTorch Geometric (PyG) [21]. We use 8 NVIDIA V100 GPUs, with 32GB of GPU memory each, in our training. For a fair comparison, we use the same hyper-parameters and training strategy used in ViG [7]. The Table 1 has the hyper parameters used in the training setup. The effective batch size is 1024. For each reported result, the model is trained from scratch on ImageNet [1] for 300 epochs. The size of all input images is set to 224x224. We use AdamW [24] as an optimizer with a base learning rate of 2e-3. In addition, we use cosine annealing schedule with 20 warm-up epochs. The data augmentation strategy used include RandAugment [25], MixUp [26], CutMix [27], random erasing [28] and repeated augment [29]. Unless stated otherwise, t_1 is set to 0.89 and d_t is set to 0.03.

TABLE 1. Hyper-parameters used in the training of SViG-Ti

Hyper-parameter	Value set for SViG-Ti
Epochs	300
Optimizer	AdamW [24]
Batch size	1024
Start learning rate (LR)	2e-3
Learning rate Schedule	Cosine
Warmup epochs	20
Weight decay	0.05
Label smoothing [30]	0.1
Stochastic path [31]	0.1
Repeated augment [29]	✓
RandAugment [25]	✓
Mixup prob. [26]	0.8
Cutmix prob. [27]	1.0
Random erasing prob. [28]	0.25
Exponential moving average	0.99996

C. EXPERIMENTAL RESULTS

We focus on the comparison between SViG-Ti and other tiny (with the similar number of parameters) isotropic architectures that use GNNs exclusively in their backbones to have a fair comparison. We illustrate the performance of our proposed SViG-Ti in comparison with ViG-Ti [7] and ViGHNN-Ti [14] GNN in Table 2.

TABLE 2. Comparing SViG-Ti to isotropic GNN architectures of similar sizes on ImageNet.

Model	Params (M)	FLOPs (B)	Top-1	Top-5
ViG-Ti [7]	7.2	1.3	74.2	92.0
ViHGNN-Ti [14]	8.2	1.8	74.3	92.5
SViG-Ti	**7.2**	**1.3**	**74.6**	**92.2**

Our proposed SViG-Ti model outperforms ViG-Ti, that has a comparable number of parameters and FLOPs, with an improvement of 0.7% on Top-1 and 0.2% on Top-5. SViG-Ti also outperforms ViHGNN-Ti [14] with 0.3% on Top-1, while using much lower number of parameters and FLOPs.

Algorithm 1 PyTorch-like Code of Max-Relative Convolution with Flexible Aggregation

```
class MaxRelativeConvlution(nn.Module):
   """
   B: Batch size
   N: Number of nodes in a graph
   D: Number of Channels or features in the vector representation of a node
   E(all): Number of edges in the whole Batch

   x: [B, N, D]
   edge_index: [2,E(all)] where index 0 is the source and index 1 is the target
   """
   ...
   ...
   def forward(self, x, edge_index):
      B, N, D = x.shape
      x_all_batches = x.reshape(B*N,D)
      x_i = x_all_batches[edge_index[0]] #Destination nodes
      x_j = x_all_batches[edge_index[1]] #Source nodes (neighborhood)
      x_res = x_j - x_i # [E(all), D]
      x_out = torch_scatter.scatter(src=x_res,
                        index=edge_index[0], dim=0,
                        reduce='max') # [B*N, D]
      x_out = x_out.reshape(B, N, D)
      ...
```

It is worth noting that a specific configuration of ViGHNN-Ti achieves a Top-1 accuracy of 75.0%, however, this comes at the cost of 9.7M parameters and 3.8B FLOPs. Thus, with a comparable number of parameters and FLOPs, SViG-Ti is still outperforming it.

D. COMPUTATIONAL COMPLEXITY OF SIMILARITY THRESHOLDED GRAPH CONSTRUCTION

The computational complexity of the similarity matrix calculation is $O(N^2 * D)$, since the similarity score is calculated for every pair of nodes (thus N^2) and dot product computational complexity is equal to the number of features per feature vector (hence D). It is worth mentioning that the similarity matrix calculation is performed for the k-nn graph construction in ViG [7]. The computational complexity of the threshold-based graph construction for each layer is $O(N^2)$, where each similarity score, corresponding to a possible edge is normalized and threshold-ed once, thus there is no need for sorting (opposed to k-nn graph construction). Assuming that $N \approx D$, the complexity of the similarity matrix calculation would be $O(N^3)$. Thus, the complexity of the graph construction is dominated by the similarity matrix calculation, which is done in both k-nn and similarity thresholded graph construction. As a result, our similarity thresholded graph construction does not introduce a computational overhead. This complexity is validated by the comparable number of FLOPs of our SViG-Ti model and the baseline ViG-Ti model as seen in table 2.

E. ABLATION STUDIES

Since there are 2 newly introduced hyper-parameters, we first show the effect of varying t_1 while keeping d_t constant, then we show the effect of varying d_t while keeping t_1 constant.

1) Effect of varying the starting threshold t_1

We show the effect of varying t_1 in Table 3. Each data point reported in table 3 represents a training experiment from scratch on ImageNet with $d_t = 0.03$ and t_1 values in table 3. The reported results represent the validation accuracies. The general trend seems to be that decreasing the thresholds gets better results until it saturates at $t_1 = 0.86$. The best Top-1 validation accuracy was 74.6% at $t_1 = 0.86, 0.89$. Changing the starting threshold changes all the thresholds in all Grapher modules by the same amount, which is why the Top-1 validation accuracy does not drop when decreasing t_1.

TABLE 3. Training SViG-Ti from scratch on ImageNet while varying t_1, with $d_t = 0.03$. This table shows the Top-1 and Top-5 validation accuracy.

t_1	0.86	0.89	0.91	0.93	0.96
Top-1 Accuracy	74.6	74.6	74.5	74.4	74.2
Top-5 Accuracy	92.1	92.2	92.2	92.1	92.2

2) Effect of varying the decrement d_t

In this study, we show how varying d_t affects the Top-1 validation accuracy in Table 4. The experimental results in table 4 are reported for models that are trained from scratch on ImageNet using $t_1 = 0.89$ and d_t values in table 4. Having no decrement gives a relatively low Top-1 validation accuracy. This shows the importance of having a decrement and not using the same threshold for all layers. It also shows that having a decrement helps fight oversmoothing. When increasing the decrement d_t, the Top-1 validation accuracy seems to improve. The highest Top-1 validation accuracy is 74.6% at $d_t = 0.03$. It is interesting that increasing the decrement from 0.03 to 0.04 decreases the Top-1 validation accuracy by 0.6%. This shows that a high decrement leads to aggregation from irrelevant neighbors in later layers, which hurts the Top-1 validation accuracy. This is due to the fact that changing the decrement d_t does not change the thresholds of the Grapher modules *uniformly*, like changing t_1. This is the reason the model performance is more sensitive to changes in d_t than changes in t_1.

TABLE 4. Training SViG-Ti from scratch on ImageNet while varying d_t, with $t_1 = 0.89$. This table shows the Top-1 and Top-5 validation accuracy.

d_t	No Decrement	0.02	0.03	0.04
Top-1 Accuracy	74.1	74.2	74.6	74.0
Top-5 Accuracy	92.0	92.1	92.2	91.9

V. CONCLUSION

In our work, we proposed a powerful alternative to k-nn through a decreasing similarity-thresholding technique, which overcomes the bias that k-nn causes without increasing the computation overhead. Our method for graph construction does not impose any graph topology over the network. Moreover, our framework proposed only two hyper-parameters (t_1 and d_t) instead of picking a k for each layer. As a result of our graph construction, our model SViG-Ti achieves a competitive performance of 74.6% Top-1 validation accuracy on a ImageNet, which is a robust benchmark. We propose a more robust implementation for the aggregation to avoid the constraint of having a fixed number of neighbors for each node. We do that without increasing memory or computation overhead. For future work, we plan on exploring the same technique with pyramid architectures, and larger isotropic models. We also hope that this work inspires more robust graph construction methods that do not enforce assumptions about the underlying graph topology.

REFERENCES

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. 1, 2, 6

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 1

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021. 1

[4] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "Mlp-mixer: An all-mlp architecture for vision," *arXiv preprint arXiv:2105.01601*, 2021. 1

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf 1

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762 1

[7] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," in *NeurIPS*, 2022. 1, 2, 3, 4, 5, 6, 7

[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1025–1035. 1

[9] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008. 1

[10] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008. 2

[11] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *CoRR*, vol. abs/1801.07829, 2018. [Online]. Available: http://arxiv.org/abs/1801.07829 2

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017. 2

[13] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "Can gcns go as deep as cnns?" *CoRR*, vol. abs/1904.03751, 2019. [Online]. Available: http://arxiv.org/abs/1904.03751 2, 3

[14] Y. Han, P. Wang, S. Kundu, Y. Ding, and Z. Wang, "Vision HGNN: an image is more than a graph of nodes," in *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 19 821–19 831. [Online]. Available: https://doi.org/10.1109/ICCV51070.2023.01820 2, 6

[15] M. Munir, W. Avery, and R. Marculescu, "Mobilevig: Graph-based sparse attention for mobile vision applications," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023 - Workshops, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 2211–2219. [Online]. Available: https://doi.org/10.1109/CVPRW59228.2023.00215 3

[16] M. Munir, W. Avery, M. M. Rahman, and R. Marculescu, "Greedyvig: Dynamic axial graph construction for efficient vision gnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3

[17] W. Avery, M. Munir, and R. Marculescu, "Scaling graph convolutions for mobile vision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2024, pp. 5857–5865. 3

[18] Y. Tian, H. Chen, C. Xu, and Y. Wang, "Image processing gnn: Breaking rigidity in super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 24 108–24 117. 3

[19] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023. [Online]. Available: https://arxiv.org/abs/1606.08415 3

[20] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018, pp. 3538–3545. 5

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 5, 6

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019. 6

[23] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019. 6

[24] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017. 6

[25] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *CVPR Workshops*, 2020. 6

[26] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *ICLR*, 2018. 6

[27] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *ICCV*, 2019. 6

[28] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *AAAI*, vol. 34, no. 07, 2020, pp. 13 001–13 008. 6

[29] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry, "Augment your batch: Improving generalization through instance repetition," in *CVPR*, 2020. 6

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826. 6

[31] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*. Springer, 2016, pp. 646–661. 6

ISMAEL ELSHARKAWI received his BSc from the American University in Cairo (AUC) in 2022. He is also currently pursuing a MSc in Computer Science at the American University in Cairo (AUC). Ismael received the Alfi Scholarship for both his Bachelor and Master's degree at AUC. His research interests are Computer Vision and Graph Machine Learning. He has been working as a Software Engineer in Siemens EDA since June 2022.

HOSSAM SHARARA joined the Department of Computer Science and Engineering at The American University in Cairo (AUC) as an Assistant Professor in July 2018. He got his BSc and MSc degrees in computer science and engineering from Alexandria University, Egypt, in 2004 and 2007, respectively. He then joined the Department of Computer Science at the University of Maryland, College Park, USA as a research assistant in the year 2007, from where he obtained another MSc degree in 2010 and received his PhD degree in relational machine learning and data ,ining in 2012 under the supervision of Prof. Lise Getoor. His PhD research was partially supported by the Dean's fellowship award, which he was granted from the University of Maryland, College Park, in 2010. From 2012 to 2018, Sharara worked as a senior engineer/research scientist at Google (2012 - 2016), and Facebook (2016 - 2018). He returned to Egypt in early 2018, joining Uber Egypt as a Senior Manager of Business and Data Analytics.

AHMED RAFEA served as the Chair of the Computer Science Department and the Vice Dean with the Faculty of Computers and Information, Cairo University. He also served as a Visiting Professor with San Diego State University and National University, USA. He was the Principal Investigator of several projects for developing Intelligent Systems and Machine Translation in collaboration with European and American Universities. He is currently a Computer Science Professor and the Ex-Chair of the Computer Science and Engineering Department, The American University in Cairo. He has led many projects aiming at using Artificial Intelligence and Expert Systems Technologies for the development of the Agriculture sector in Egypt. He has authored over 200 scientific articles in International and National Journals, Conference Proceedings, and Book chapters. His research interests are data, text and web mining, natural language processing and machine translation, knowledge engineering, and knowledge-based system development. Dr. Rafea was a member with the Center of Excellence on Data Mining and Computer Modeling, sponsored by the Ministry of Communication and Information Technology.

• • •