Vision Graph Neural Networks: A Replication Study

Morteza Mirzaei
McMaster University
Hamilton, Ontario, Canada
mirzam48@mcmaster.ca

ABSTRACT

This paper aims to implement and replicate the findings of the Vision Graph Neural Networks (ViG) paper. ViG adopts a novel approach by treating images as graphs and utilizing graph neural networks (GNNs) algorithms for feature extraction and downstream tasks. We will reproduce experiments and conduct ablation studies to evaluate ViG's performance in tasks such as image classification, thus contributing to the assessment of its reproducibility and potential in visual perception tasks. Additionally, we will discuss challenges encountered during implementation, such as missing details or implementation complexities. The code for our implementation can be accessed at https://github.com/mirzaim/VisionGNN.

1 INTRODUCTION

Traditional convolutional neural networks (CNNs) have historically dominated the landscape of computer vision. However, recent advancements in transformer architectures [4, 14], coupled with attention mechanisms, have propelled Vision Graph Neural Networks (ViG) [5] into the spotlight. By treating images as graphs and harnessing the power of graph neural networks (GNNs), ViG offers more adaptability and efficacy in visual perception tasks, transcending the limitations of conventional grid or sequence representations. The seminal Vision GNN paper [5] introduces an innovative architecture that partitions input images into patches, each treated as a node within a graph. Through iterative information exchange among these nodes via graph convolutional and feed-forward network modules, ViG achieves remarkable performance across diverse tasks such as image classification and object detection.

In this paper, we aim to replicate the results of the Vision GNN paper [5]. Our methodology involves a comprehensive discussion of the datasets and experimental setups employed for our investigations. We replicate the experiments conducted in the original paper, including the main results and ablation studies on architecture. Furthermore, we delve into the challenges encountered during the implementation process and elucidate the strategies employed to overcome them. Through our endeavors, we seek to contribute to the examination of the reproducibility of the ViG model.

2 EXPERIMENTS

In this section, we conduct experiments to demonstrate the effectiveness of ViG models with image classification tasks. We begin by outlining the datasets used for training and evaluation, followed by details of the experimental setup, including model configurations and training procedures.

Figure 1: Sample images from the Imagenette dataset.

2.1 Datasets

We utilized two smaller datasets compared to ImageNet [3] in our experiments: the Imagenette dataset and the Imagewoof dataset [7], due to limited resources.

The Imagenette dataset is a subset of 10 classified classes from ImageNet, featuring diverse categories such as tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, and parachute (see Figure 1). The Imagenette dataset is publicly available on GitHub at the following link: https://github.com/fastai/imagenette. It comprises 13,394 images, with a 70/30 train/validation split.

Similarly, we employed the Imagewoof dataset, a subset of 10 dog breeds from ImageNet that present more challenging classification tasks. The included breeds are Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever, and Old English sheepdog. The Imagewoof dataset contains 12,954 images with 70/30 train/validation split. It is accessible through the same GitHub repository as Imagenette.

For consistency and computational efficiency, we utilized the '320 px' version of both datasets, where images have their shortest side resized to 320 pixels while maintaining aspect ratio. This version ensures standardized resolution across the dataset, facilitating uniformity in image dimensions for both training and evaluation purposes.

Table 1: Results of VisionGNN models on Imagenette and Imagewoof datasets.

Model	Depth	Dimension	Params (M)	Top-1 Accuracy	
				Imagenette	Imagewoof
VisionGNN-Tiny	12	192	9.8	69.04	42.61
VisionGNN-Small	16	320	36.2	72.36	45.00
VisionGNN-Big	16	640	144.5	74.39	45.97

To facilitate reproducibility and ease of access, we provide a script named `get_dataset.sh` within our code repository. This script automates the process of downloading and extracting both the Imagenette and Imagewoof datasets to the appropriate location, ensuring consistency across different environments and simplifying the setup process for reproducing our experiments.

2.2 Experimental Settings

We employed the GELU activation function [6] within our neural network architecture, resizing each image to dimensions of 224×224. Data augmentation techniques included Random Augmentation [2], Random Erasing [20], and Horizontal Flip. Prior to training, images were normalized using the mean and standard deviation from the ImageNet dataset [3]. Although initial experiments with Mixup [19] and Cutmix [18] augmentation did not yield significant improvements, subsequent research suggested their efficacy in higher epochs, typically beyond 80 [10, 16]. Because of limited resources and time, we trained the model for 80 epochs; therefore, Mixup and Cutmix were not utilized in the final training. A batch size of 64 images was utilized for training, with a classifier comprising two linear layers with dimensions of input features $\rightarrow$ 1024 $\rightarrow$ number of classes. Loss optimization was performed using the Cross-Entropy loss function, coupled with the Adam optimizer [8] with learning rate, β_1, and β_2 values set to 10^{-5}, 0.9, and 0.999 respectively. Implementation of the model was conducted using the PyTorch framework [11]. Notably, computational resources utilized for training comprised an Intel Core i9-13900KF CPU, 32 GB RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GB dedicated memory.

2.3 Main Results

In this section, we present the performance analysis of VisionGNN models on the Imagenette and Imagewoof datasets. The main paper discusses two implementations of VisionGNN: Isotropic and Pyramid. For this study, we focused on the Isotropic architecture, which maintains the feature size unchanged in its main computational body, facilitating scalability and hardware acceleration. This architectural scheme has gained popularity in transformer models for natural language processing [14], and recent advancements in vision-based neural networks have also explored its efficacy, such as MLP-Mixer [12], ViT [4], and ResMLP [13].

Table 1 provides an overview of the results obtained for VisionGNN models of varying sizes. All models were trained for 80 epochs, although it should be noted that the largest network could potentially benefit from additional training epochs as it was still converging. However, due to constraints in time and resources, training was terminated at 80 epochs. The table illustrates the top-1 accuracy achieved by VisionGNN models on both Imagenette and Imagewoof datasets, demonstrating the performance of the Isotropic architecture across different network sizes. As you can see, by increasing the model size, the accuracy generally increases.

Table 2: Ablation study of different modules in VisionGNN.

GraphConv	FC in Grapher	FFN	Params (M)	Top-1 Acc
✓	✗	✗	3.2	71.80
✓	✓	✗	9.8	73.10
✓	✗	✓	29.6	74.83
✓	✓	✓	36.2	72.36

2.4 Ablation Study

To further investigate the impact of different components within the VisionGNN architecture, we conducted an ablation study to assess the performance of the model under various conditions. We conduct ablation study of the proposed method on Imagenette classification task and use the isotropic VisionGNN-Small as the base architecture.

2.4.1 The effect of different modules in VisionGNN. To adapt the graph neural network to visual tasks, fully connected (FC) layers were introduced in the Grapher module, and feedforward neural network (FFN) blocks were utilized for feature transformation. The effects of these modules were evaluated through an ablation study.

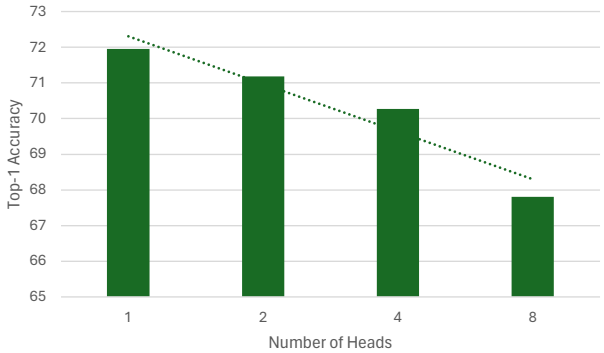

Figure 2: Effect of the number of heads on the top-1 accuracy of the VisionGNN model. The x-axis represents the number of heads, while the y-axis represents the top-1 accuracy achieved on the Imagenette dataset.

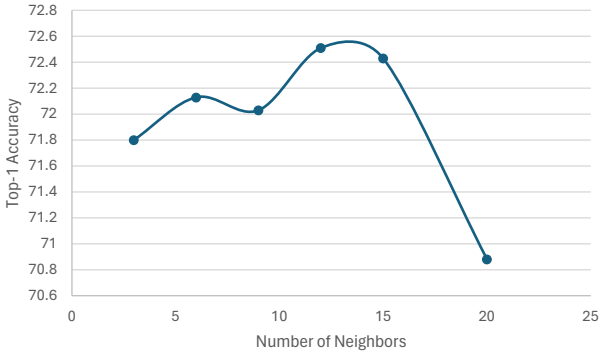

Figure 3: Performance comparison of different numbers of neighbors on the Imagenette dataset. The chart illustrates the impact of varying the parameter K on the classification task accuracy.

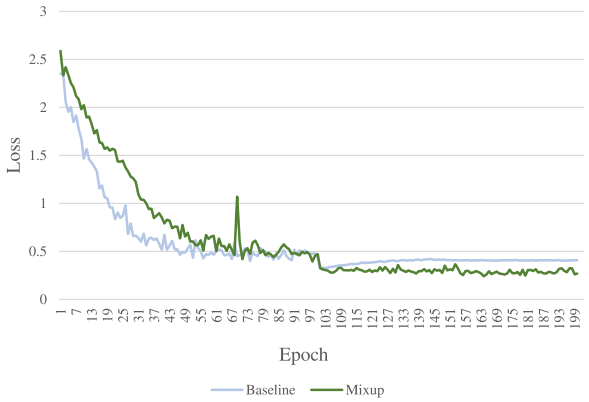

Figure 4: Training loss comparison with and without Mixup augmentation. The figure is adapted from [17].

It's worth noting that all models were trained for 80 epochs, which may not have been sufficient for the model with all components to fully converge and optimize the FC layers. As a result, a performance degradation was observed in the last row of Table 2. However, if training were to continue, it is expected that the accuracy of the model with all components would surpass that of the previous configurations.

2.4.2 The number of heads. The multi-head update operation enables the Grapher module to process node features in different subspaces. The number of heads controls the transformation diversity in these subspaces. It's assumed that different subspaces are independent from each other. Therefore, increasing the number of heads results in fewer parameters overall, but it also diminishes the connections between different parts of the vector. As observed in Figure 2, the accuracy generally decreases with an increase in the number of heads. For the main model, we chose the number of heads equal to one as the default value.

2.4.3 The number of neighbors. The number of neighboring nodes during graph construction is a crucial parameter affecting the range of information aggregation. Striking a balance is essential, as too few neighbors hinder effective information exchange, while an excessive number may cause over-smoothing. We conducted experiments by varying the parameter K from 3 to 20 and present the results in Figure 3. Our analysis reveals that optimal performance on the Imagenette classification task is observed when K is within the range of 12 to 15.

3 REIMPLEMENTATION CHALLENGES

In this section, we discuss the challenges encountered during the reimplementation of the paper. Despite diligent efforts to reproduce the reported results, several obstacles were encountered along the way. We provide insights into these challenges and detail the strategies employed to overcome them, offering valuable lessons for future endeavors in replicating research findings.

3.1 Graph Structure Extraction

At the outset of the reimplementation process, a challenge arose from a misconception regarding the extraction of the graph structure from the image. Initially, it was inferred from the paper that a single graph structure should be extracted from the image at the beginning and maintained throughout the entirety of the model's architecture and higher layers. However, it became evident that, in reality, recalculating the graph structure was necessary at each layer of the model.

In different layers of the model, the level of detail or abstraction from the input data varies. Some layers focus on basic features like edges or shapes, while others focus on more complex patterns or concepts. When we try to preserve the same graph structure derived from the simpler features in the lower layers and use it in the higher layers, it doesn't work well. This is because, in the higher layers where the model deals with more abstract concepts, it needs to make connections between distant features or elements in the input data. So, the graph structure needs to adapt and change at each layer to accommodate these long-distance connections. The architecture is more like a transformer, and each layer needs to find different graph connections.

3.2 Effectiveness of Mixup and Cutmix

Another challenge emerged when attempting to utilize advanced data augmentation techniques such as Mixup [19] and Cutmix [18]. Despite experimenting with various hyperparameters, these augmentation methods resulted in decreased model accuracy. Upon investigation, it was discovered that these techniques require longer training durations to become effective. Specifically, it was observed that Mixup, a commonly used augmentation method, necessitates more epochs to converge due to its exploration of additional regions in the data space. For instance, while a standard training routine of 90 epochs may suffice, training ResNet-50 on ImageNet with Mixup typically requires up to 200 epochs to achieve convergence. This phenomenon extends beyond Mixup and is also observed in other augmentation methods, which significantly enhance the complexity of training data [1, 18].

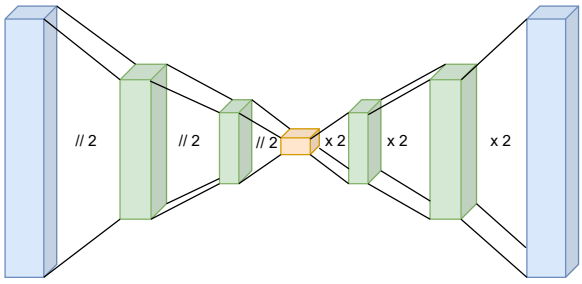

Figure 5: rchitecture of the feature extractor used in our reimplementation.

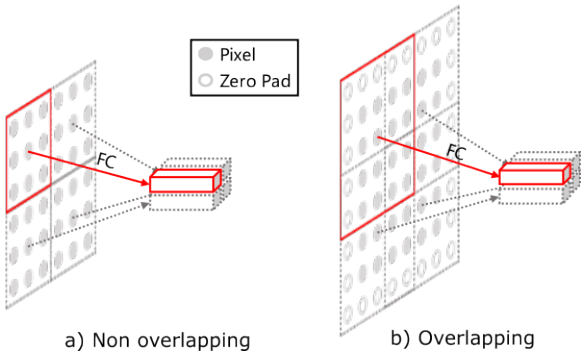

Figure 6: Comparison between overlapping and non-overlapping feature extraction methods. This figure is adapted from [15].

An analysis of the training loss for ResNet-18 with and without Mixup augmentation revealed an intriguing pattern visualized in figure 4. Initially, the loss was higher with Mixup, particularly in the early epochs. However, beyond 100 epochs, the loss consistently decreased with Mixup, lowering the training loss without augmentation [17].

Although the benefits of these augmentation techniques become apparent after a certain number of epochs, our constrained resources and time limitations prevented us from fully exploiting their potential. Consequently, Mixup and Cutmix augmentations were not incorporated into the final training process.

3.3 Feature Extraction from Patches

The original paper does not provide details on how to extract features from patches. In our reimplementation, we utilized a method involving six linear layers to extract features, where each patch is flattened and fed into the network. Each layer consists of a fully connected neural network, batch normalization layer, and GELU activation function. This architecture resembles an autoencoder. A diagram of the network can be seen in Figure 5.

Upon further investigation and examination of other implementations of the paper, we discovered that the original approach utilizes a more advanced method known as overlapping feature extraction. This method involves an overlapping sliding window into one token progressively (see Figure 6) and employing a deep convolutional neural network to extract features [15]. Even though the overlapping feature extraction method is more advanced and computationally expensive, our method is simpler and able to achieve similar results.

3.4 Distance Measure in KNN Calculation

The original paper does not specify how they measure the distance between feature vectors when calculating K-nearest neighbors (KNN). In our reimplementation, for simplicity, we opted to use cosine similarity, which involves taking the inner product of two vectors and selecting the k most similar ones as the output of the KNN algorithm.

Upon reviewing the original paper's code, it appears that they employ a more advanced KNN algorithm known as "Dilated KNN." This method utilizes the Dilated k-NN to find dilated neighbors after every Graph Convolutional Network (GCN) layer and construct a Dilated Graph. Specifically, for an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with Dilated k-NN and d as the dilation rate, the Dilated k-NN returns the k nearest neighbors within the $k \times d$ neighborhood region by skipping every d neighbors. The nearest neighbors are determined based on a pre-defined distance metric, commonly the ℓ_2 distance in the feature space of the current layer [9].

Despite the advanced nature of Dilated KNN, our approach using cosine similarity proved to be more computationally efficient while achieving comparable accuracy.

3.5 Multi-Head layer implementation

As we explained in the 2.4.2, we devide the feature space to subspaces as number of heads and apply the different transformation to each subspace and then concatenate the results. In the first, implementation we used number of heads time the linear layer to apply the transformation to each subspace and concatenate the results. However, as the number of heads increases, this implementation becomes inefficient and computationally expensive because it uses the for loop to apply the transformation to each subspace as you can see in Algorithm 1.

But we find out we could use 1d convolutional layer with kernel size of one and group size of number of heads. This layer is equivalent to the linear layer for each subspace and concatenation of the results. This implementation fully utilizes the parallel processing capabilities of the GPU and more efficient in maner of time and memory.

Algorithm 1 Multi-Head Layer Implementation with Linear Layers

Input: Input vectors V, number of heads h
Output: Transformed vectors $\tilde{V}$
Divide input vectors into h heads: $v_1, \ldots, v_h := V$
for $i = 1$ **to** h **do**
    $\tilde{v}_i = W_i \cdot v_i + b_i$
end for
$\tilde{V} = \text{concatenate}([\tilde{v}_1, \ldots, \tilde{v}_h])$

REFERENCES

[1] Pengguang Chen, Shu Liu, Hengshuang Zhao, and Jiaya Jia. 2020. Gridmask data augmentation. *arXiv preprint arXiv:2001.04086* (2020).

[2] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. 2020. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 702–703.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[5] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. 2022. Vision gnn: An image is worth graph of nodes. *Advances in neural information processing systems* 35 (2022), 8291–8303.

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

[7] Jeremy Howard. [n.d.]. imagenette. https://github.com/fastai/imagenette/

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9267–9276.

[10] Zixuan Liu, Ziqiao Wang, Hongyu Guo, and Yongyi Mao. 2023. Over-training with mixup may hurt generalization. *arXiv preprint arXiv:2303.01475* (2023).

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[12] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems* 34 (2021), 24261–24272.

[13] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. 2022. Resmlp: Feedforward networks for image classification with data-efficient training. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2022), 5314–5321.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[15] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. 2022. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media* 8, 3 (2022), 415–424.

[16] Hao Yu, Huanyu Wang, and Jianxin Wu. 2021. Mixup without hesitation. In *Image and Graphics: 11th International Conference, ICIG 2021, Haikou, China, August 6–8, 2021, Proceedings, Part II 11*. Springer, 143–154.

[17] Hao Yu, Huanyu Wang, and Jianxin Wu. 2021. Mixup without hesitation. In *Image and Graphics: 11th International Conference, ICIG 2021, Haikou, China, August 6–8, 2021, Proceedings, Part II 11*. Springer, 143–154.

[18] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. 2019. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6023–6032.

[19] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[20] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13001–13008.