Images as Graphs: Training Deep Graph Neural Networks for Computer Vision

Alex Biehl, Oscar Shi, Nishant Aswani

[1]New York University
Tandon School of Engineering
Brooklyn, New York 11221 USA
ab7289@nyu.edu, zs1616@nyu.edu, nsa325@nyu.edu

Abstract

Computer vision tasks have typically been approached using grid-convolutions. However, recent work in the graph domain has raised interest in applying graph neural networks (GNNs) to computer vision problems. Specifically, graph convolution are a powerful tool that allow the model to capture neighborhood information in a graph dataset, acting as a generalization to grid-like convolution but for unstructure data. In this project, we compared the performance of different graph convolution layers on the MNIST and CIFAR-10 benchmarks. In our experiments, we used the GeneralConv, the Graph Attention Network (GAT), and the GENConv layers. As a check, we verified that all of our models achieved test accuracies of $\geq 95\%$ on the MNIST dataset. We found that our best performing models were the 16-layer GeneralConv models that achieved test accuracies of 56% on the CIFAR-10 dataset. This was followed by the GENConv models and the GATConv model, respectively. However, as the models got deeper, GENConv models were not able to maintain performance, while GATConv did not suffer significantly. Our results demonstrate that GCN models do suffer from severe performance degradation as they are stacked deeper, despite attempts by researchers to reformulate layers, such as the GENConv layer, for deeper networks.

Source Code

Our project's source code including notebooks can be found at https://github.com/ab7289-tandon-nyu/GraphVision

Introduction

Problem Description

Deep learning solutions for computer vision typically represent images in a pixel-grid format and apply successive convolution operations, which has the limitation that it treats all pixels equally and requires the training dataset to be of uniform size (Vasudevan et al. 2022). In a relatively recent set of approaches, images are represented as graphs to train graph neural networks on computer vision tasks, allowing the inclusion of non-traditional and non-rectangular images. Graph convolution networks (GCNs) are a subset of GNNs which capture neighborhood information, providing a graph-based analogue to grid convolutions.

However, GCNs suffer from performance degradation as models get deeper. Recent work on GCNs has sought to improve deep GCN performance on large-scale graphs (Li et al. 2020). This work runs experiments on GNN architectures with computer vision benchmarks to compare and contrast a few recent proposals in the GNN domain.

Literature Review

Tasks Based On Graph Data

The output from a network's final GCN layer can be used to compute the results for a variety of tasks (Dwivedi et al. 2020):

- graph classification
- graph regression
- node classification
- edge classification

Among those tasks, our project focuses on graph classification.

Graph Data As Input

Input Representation For a GCN model, input features are typically represented in the following manner, according to (Dwivedi et al. 2020):

- each node contains a list of node features
- each edge optionally contains a number of edge features
- both node features and edge features are embedded to the same-dimensioned hidden features

For image input data in particular, there are a variety of ways to convert pixels into nodes. The most obvious method is to use a grid, either one pixel per node or a fixed number of pixels per node (Monti et al. 2016).

Alternatively, pixels can be grouped into nodes by in a less rigid manner. (Lin, Zhong, and Lu 2021) summarizes that there are so far three ways to flexibly group pixels: superpixel segmentation, semantic segmentation, and deep clustering. Given the time constraint of the project, we used only graph image data based on superpixels.

Superpixels One such data set was based on (Dwivedi et al. 2020), which used the SLIC algorithm to create CIFAR10 graphs with 85-150 nodes. The authors also created MNIST graphs with 40-75 nodes.

Another data set for MNIST was created based on (Monti et al. 2016), which in its paper described three versions of MNIST graphs: one version where all images have 300 superpixels, the second 150 superpixels, and the third 75 superpixels. Though the number of superpixels are uniform across the images, their graph differ because edges are connected differently: the smaller graphs (75 superpixels) vary more among the images compared to the largest graphs (300 superpixels). This variance from one image to another can cause performance troubles for those GNN models that rely overly on inputs to be uniform and domain-dependent (Monti et al. 2016).

Graphs of Superpixels Once superpixels are created, the next design choice would be how to connect them. In other words, how to define edges between any two nodes. (Monti et al. 2016) simply define edges based on adjacency: if two superpixels touch each other, they will share an edge. In contrast, (Dwivedi et al. 2020) builds an adjacency matrix with a k-nearest neighbor algorithm, and uses $k = 8$ for both their MNIST and CIFAR10 benchmark datasets.

Data Sets We wanted to select data sets consisted of real-world images. Based on the thorough review done by (Dwivedi et al. 2020), our selection narrowed down to

- MNIST
- CIFAR10

Among the two, we expected our effort to focus more on CIFAR10 and less on MNIST. That is because classifying the handwritten digits in the MNIST dataset is almost trivial by today's standards (Monti et al. 2016), with which most modern models can be expected to reach at least 90% test accuracy. (Dwivedi et al. 2020) suggests that MNIST dataset is most useful as a first-step sanity check on our designs, which is how we treat the dataset in our work.

Meanwhile, to make the MNIST dataset more challenging and therefore interesting to test our GNNs on, we also used the version of MNIST superpixels based on (Monti et al. 2016).

Graph Convolution Networks (GCN)

Graph Convolution Networks (GCN) are a strain of GNNs that may be understood as a generalization of convolutional neural networks for unstructured data, such as graphs. The input to a GCN layer is a set of node features $\mathbf{h} = \{h_1, h_2, ..., h_n\}$ and the layer outputs a new set of node features $\mathbf{h}' = \{h'_1, h'_2, ..., h'_n\}$.

At the beginning of each GCN layer, the features of each node are averaged with their local neighborhood by using a special matrix (comprising of the graph's adjacency matrix). This results in producing coefficients c_{ij} which are normalization constants dependent on the graph structure. Next, to compute the output features, the normalization constant linearly combined with neighborhood input features and passed through an activation function.

$$\text{h'}_i = \sigma(\textstyle\sum_j \frac{1}{c_{ij}} \mathbf{W} h_j)$$

Most importantly, the value c_{ij} isn't a learnable representation; rather, it is a value derived based on the structure of the graph.

GCNs have shown success with graph data; however, contrary to CNNs and transformers, they tend to degrade when stacked deeper. Thus, some works, such as (Li et al. 2020), have proposed strategies to train deeper GCNs without performance degradation. Another work (Han et al. 2022) attempts to mitigate the performance degradation of deep GCNs by introducing more "feature transformations and nonlinear activations". Thus, there seems to be a variety of approaches to make GCNs more capable of handling larger graphs.

Graph Attention Networks (GATs)

Inspired by the popularity of self-attention in deep learning, the authors of graph attention networks (GATs) sought to introduce an attention-based architecture to graph-based node classification. GATs are meant to compute the "hidden representations of each node [...] by attending over its neighbors" (Veličković et al. 2017). GAT models can be built solely by stacking graph attention layers.

Within a GAT layer, each node is first multiplied by a weight matrix $\mathbf{W}$, before it attends to each of its first-order neighbors (e.g. $e_{ij} = a(\mathbf{W}h_i, \mathbf{W}h_j)$) to produce an attention coefficient. The authors implement this attention mechanism as a concatenation ($||$), a single feedforward layer, followed by a LeakyReLU operation. Symbolically, this is represented as (Veličković et al. 2017):

$$\text{e}_{ij} = LeakyReLU(a \cdot \mathbf{W}h_i || \mathbf{W}h_j)$$

Each attention coefficients passes through a softmax layer, resulting in a normalized attention coefficient α_{ij}. Finally, to obtain the output feature set $\mathbf{h}'$, each normalized attention coefficient is linearly combined with corresponding input features and passed through an activation function (Veličković et al. 2017).

$$\text{h'}_i = \sigma(\textstyle\sum_j softmax(e_{ij}))$$

GAT models can simply be viewed as a variation of GCNs, where the coefficient multiplied with $\mathbf{W}h_j$) can be learned through an attention mechanism. On the other hand, as described earlier, GCNs use a 'static' coefficient which depends on the graph structure (Javaloy et al. 2022).

Deeper GCNs (GENConv)

GENConv was proposed as a layer for training deep GCNs. The layer differs from typical GCN layers by introducing a generalized message aggregator, a novel type of residual connection, and an additional "message normalization" layer (Li et al. 2020).

To obtain the output feature set $\mathbf{h}'$, a GENConv layer uses the function:

$$\text{h'}_i = MLP(h_i + s \cdot \|h_i\| \cdot \tfrac{e_i}{\|e_i\|})$$

where s is a learnable scalar factor and e_i is a coefficient, like in previous layers (Li et al. 2020).

Unlike the previously described layer types, it is important to note that GENConv has an additional MLP as a final step in the layer. Moreover, the non-linear activation function is actually hidden within the aggregation step (Li et al. 2020).

$$\mathrm{e}_{ij} = \sigma(\psi_{ij})$$

We don't expand upon the aggregation step here; in short, it includes a residual connection as well as an understanding of the graph structure. The authors claim that the residual connections significantly improve the performance over typical GCNs.

Technical Details

Data Sets

Table 1 summarizes the three data sets that we used.

	CIFAR10(1)	MNIST(1)	MNIST(2)
#graphs	45,000	55,000	70,000
#nodes/graph	117.6	70.6	75
#edges/graph	941.2	564.5	1,393.0
#features	5	3	1
#classes	10	10	10

Table 1: Statistics of Data Sets Used: (1) superpixel graphs generated according to (Dwivedi et al. 2020) and (2) superpixel graphs generated by (Monti et al. 2016)

The nodes in (Dwivedi et al. 2020) include not only the color channels but also two additional features representing the location of each node within the original image.

Models

Our objective led us to only GCN architectures that are deep, one prominent example of which is the DeeperGCN design proposed by (Li et al. 2020). Within that architecture, data repeatedly flows through the following layers:

1. **Normalization Layer**: we mostly used batch normalization as described in (Ioffe and Szegedy 2015), and also experiment with layer normalization over minibatches as described in (Ba, Kiros, and Hinton 2016)
2. **Activation Layer**: we utilized ReLU, and also experimented with GELU (Gaussian Error Linear Units) and ELU (Exponential Linear Unit)
3. **Dropout layer**: we experiment dropout probabilities of 0.20, 0.25, and 0.50
4. **Graph Convolution Layer (GCN Operator)**: We experimented with three main types of Graph Convolution Layers:
   (a) The GENeralized Graph Convolution (**GENConv**) from (Li et al. 2020)
   (b) The general GNN layer (**GeneralConv**) from (You, Ying, and Leskovec 2020)
   (c) The graph attentional operator (**GATConv**) from (Veličković et al. 2017)
5. **Skip Connection**: we used the default residual connection in (Li et al. 2020) known as "res+"

GENConv Models Our GENConv models are based on the GENeralized Graph Convolution (GENConv) proposed by (Li et al. 2020), which constructs message as $\mathbf{x}_i' =$

$$\mathrm{MLP}\left(\mathbf{x}_i + \mathrm{AGG}\left(\left\{\mathrm{ReLU}\left(\mathbf{x}_j + \mathbf{e_{ji}}\right) + \epsilon : j \in \mathcal{N}(i)\right\}\right)\right),$$

where AGG in our model was implemented as softmax.

Coimpared to our other graph convolution types, the GENConv contains significantly more trainable parameter, as illustrated by Table 2. That is because GENConv's message normalization involves a multi-layer perceptron (MLP) that is configurable (Li et al. 2020). All our GENConv models used the default of 2 layer hidden MLP.

	GENConv 4L 128H	GeneralConv 4L 128H	GATConv 4L 128H
#parameters	268,046	70,410	71,434

Table 2: Number of parameters of our three main types of graph convolution layers, illustrated by one set of models using 4 graph convolutional layers (L) and a hidden dimension of 128 (H)

This increased number of parameters, however, did not necessarily result in prolonged training time. In fact, among our three types of graph convolutions, our GENConv models were the fastest to train, all else being equal.

GeneralConv Models Our GeneralConv Models are based on the general GNN layer (GeneralConv) proposed by (You, Ying, and Leskovec 2020), in which the k-th GNN layer is defined as

$$\mathbf{h}_v^{(k+1)} = \mathrm{AGG}(\{\mathrm{ACT}(\mathrm{DROPOUT}(\mathrm{BN}(\mathbf{W}^{(h)}\mathbf{h}_u^{(k)} + \mathbf{b}^{(k)}))), u \in \mathcal{N}(v)\}),$$

where $\mathbf{h}_u^{(k)}$ is the k-th layer embedding of node v, $\mathbf{W}^{(h)}$, $\mathbf{b}^{(k)}$ are trainable weights, and $\mathcal{N}(v)$ is the local neighborhood of v.

GATConv Models Our GATConv models are based on the graph attentional operator (GATConv) proposed by (Veličković et al. 2017), which defines a self-supervised graph attentional operator as:

$$\mathbf{x}_i' = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \textstyle\sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j,$$

where the αs are attentions.

Whereas that paper applied GATConv to citation network data (Citeseer, Cora and Pubmed) and to protein interaction data (PPI), here we tested various GATConv models using our image datasets for image classification tasks.

Loss Function

We used Cross Entropy Loss across all our experiments, since our objective is to compare various GNN architectures for the task of image classification.

Hyperparameter Selection

Number of Layers For each of our three architectures, we experimented with 4, 8, 16, and 32 graph convolution layers. We also constructed a few models with 64 graph convolution layers.

Hidden Layer Dimension For each of our three architecture and each depth (number of convolutional layers), we experimented with both 128 and 256 hidden dimensions.

Learning Rate With our deeper models (16 and 32 convolutional layers), we tried lowering our learning rates after observing that our initial learning rate resulted in extremely poor performance, even though the same initial learning rate worked fine with shallower GCNs. After lowering learning rate from 1×10^{-2} to 1×10^{-3}, our deeper GCNs' performance returned to a level similar to those of our shallower models.

We also ran a few experiments with cyclical learning rates that cycled between 1×10^{-4} and 1×10^{-3}. While this cyclic schedule began to show convergence on a 64 layer, 4M parameter deep GCN model–due to slow training time and constraints of colab–we were not able to train this model for more than a handful of epochs.

Other Training Details

Epochs All of our models were trained for at least 100 epochs, to make sure that even the slowest-converging models got enough time to improve their performance. As it turns out, GeneralConv generally required more epochs to before validation loss bottoms out (Table 3). GATConv generally required fewer epochs, but each epoch takes longer.

	GENConv	GeneralConv	GATConv
avg min epoch#	54	67	39

Table 3: Average epoch at which validation loss bottoms out

Edge Attributes

Even though edge attributes are optional as inputs to GNNs, we included edge attributes in most of our models since doing so improved performance. In the context of image as graphs, we used edge attributes defined as the Cartesian distance between any two nodes.

Results

MNIST

Most of our models achieved 95% test accuracies on both versions of our MNIST superpixel data sets, namely the one by (Dwivedi et al. 2020) and by (Monti et al. 2016).

This reassures us that our various GCN models were constructed correctly and inline with the DeeperGCN design proposed by (Li et al. 2020). So our models passed the sanity check, in the words of the (Monti et al. 2016).

However, given our objective of comparing and contrasting various GCN designs, such lack of differentiation when using MNIST superpixel data sets means the true conclusions need to be drawn by using CIFAR10 superpixel data set.

CIFAR10

DeeperGCN While the DeeperGCN design aimed at reliably train very deep GCNS (Li et al. 2020), we found that in the domain of image classification, the DeeperGCN architecture alone was not sufficient to prevent a deep GCN model from deteriorating performance compared to a similarly constructed but shallower GCN. We discovered that lowering learning rate mitigates such performance deterioration (Table 4).

Once we customized our hyperparameters to rescue the performance of our deepest models, we found that GATConv's performance was near uniform across different depths, GeneralConv's accuracies improved as the model deepened, while GENConv's performance worsened with depth, despite our best effort in mitigating such worsening (Table 4).

GAT When compared to the results obtained by (Dwivedi et al. 2020), we recognized that our models performed poorly on the CIFAR-10 benchmark. For example, the authors were able to obtain a 64% accuracy with a GAT model containing 110,400 parameters. On the other hand, our 8-layer GAT model with 128 hidden dimensions (137,994 parameters) obtained a test accuracy of 50%. While there could be several factors that affected the performance of our model, we believe our training regimen would have significantly benefit from a dynamic learning schedule, similar to a decaying schedule employed by (Dwivedi et al. 2020).

However, it is important to note that after switching to a learning rate of 1×10^{-3}, our training loss continued to fall without significant oscillations (see Figure 1), but our validation loss began to rise, clearly implying overfitting. Thus, we believe a combination of a careful learning rate scheduling and further experimentation with dropout parameters or other layer hyperparameters could have put us at a similar performance as published benchmarks.

Comparison of Graph Convolution Layer Types Our best performing models were our 16-layer GeneralConv models with test accuracies at 54%. Overall, GeneralConv had the best performance, followed by GENConv, with GATConv being the worst.

We also tested the even deeper 64-layer models: our preliminary test showed loss curves continuing to go down after at least 70 epochs, but due to high training time per epoch, we were unable to train further despite repeated attempts.

Loss Curves All of our models managed to converge to a local minimal loss, such as Figure 1a, after ran with several different combinations of hyperparameters: various learning rates, optimizers, etc.. If we had more time, we think we could further improve convergence speed and reduce choppiness in some of our experiments such as Figure 1b.

	GENConv	GeneralConv	GATConv
4L 128H	0.52	0.46	0.44
4L 256H	0.48	0.48	0.54*
8L 128H	0.54	0.48	0.50*
8L 256H	0.54	0.45	0.50*
16L 128H	0.47	0.56*	0.50*
16L 256H	0.50	0.56*	0.52*
32L 128H	0.25*	na	0.49*
32L 256H	0.47*	na	0.52*

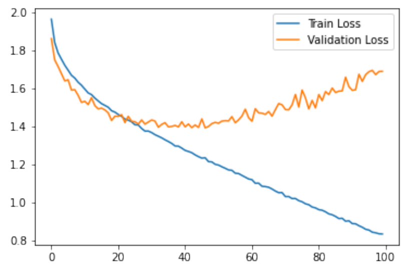
Table 4: Test accuracies of our GCNs on CIFAR10 superpixel data after at least 100 epochs, organized by graph convolution type (columns) and by the number of graph convolution layers (L) and the size of the hidden dimension (H) (rows). The symbol * denotes results based on 1×10^{-3} initial learning rate; the remaining results are based on 1×10^{-2} initial learning rate. The symbol na denotes experiments that we were not able to carry out due to limited computing resource.

Conclusion

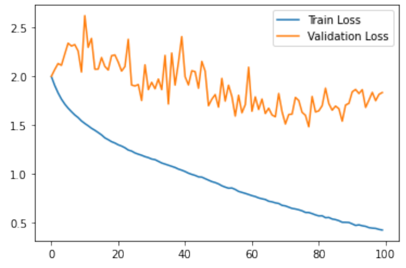
Our objective was to compare different GNN models architectures. As a result, we instantiated different GNN models as similarly as possible: identical depth, similar number of parameters, similar learning rates, etc. We found that when models got deeper, it was the case for *all* of our model architectures that we had to lower learning rate in order to maintain a reasonable level of accuracy ($\geq$50%). Such forced uniformity might have been unfair to certain architectures: perhaps one architecture did not perform well in our experiment simply because our predetermined set of designs missed an optimal design that would have enabled that architecture to perform. If we had more time, we would carry out more experiments and designs to be applied across all of our architectures.

We believe an interesting variation for future work might be hybrid architectures, where one could stack different graph convolutional layers to study how it might affect model performance. More specifically, it might be beneficial to understand whether certain modules, such as attention, are more beneficial at earlier or later stages.

The project's scope was narrower than our objective due to training time that significantly exceeded our expectation: it took an average model of ours 4 hours before loss curve flattened, and our deepest models required at least 8 hours, which timed out the GPUs available to us. If we had more time, we would have tested Vision GNN (ViG) (Han et al. 2022) and Vision Transformers (ViT) (Dosovitskiy et al. 2020).



(a) GATConv with 32 layers and hidden dimension of 256



(b) GENConv with 32 layers and hidden dimension of 128

Figure 1: Train and validation loss curves of two of our models

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

Dwivedi, V. P.; Joshi, C. K.; Luu, A. T.; Laurent, T.; Bengio, Y.; and Bresson, X. 2020. Benchmarking Graph Neural Networks.

Han, K.; Wang, Y.; Guo, J.; Tang, Y.; and Wu, E. 2022. Vision GNN: An Image is Worth Graph of Nodes.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

Javaloy, A.; Sanchez-Martin, P.; Levi, A.; and Valera, I. 2022. Learnable Graph Convolutional Attention Networks. *arXiv preprint arXiv:2211.11853*.

Li, G.; Xiong, C.; Thabet, A.; and Ghanem, B. 2020. DeeperGCN: All You Need to Train Deeper GCNs.

Lin, Q.; Zhong, W.; and Lu, J. 2021. Deep Superpixel Cut for Unsupervised Image Segmentation.

Monti, F.; Boscaini, D.; Masci, J.; Rodolà, E.; Svoboda, J.; and Bronstein, M. M. 2016. Geometric deep learning on graphs and manifolds using mixture model CNNs.

Vasudevan, V.; Bassenne, M.; Islam, M. T.; and Xing, L. 2022. Image Classification using Graph Neural Network and Multiscale Wavelet Superpixels. *arXiv preprint arXiv:2201.12633*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph Attention Networks.

You, J.; Ying, R.; and Leskovec, J. 2020. Design Space for Graph Neural Networks.