Extending Graph U-Nets

Ayush Agarwal
Ddpt. of Computer Science
Stanford University
ayush@stanford.edu

Caleb Chiam
Dept. of Computer Science
Stanford University
calebc96@stanford.edu

Anthony Vento
Dept. of Electrical Engineering
Stanford University
avento@stanford.edu

Abstract

Graph U-Nets [1] employ an encoder-decoder architecture that encodes an input graph into feature representations at multiple levels through pooling and downsampling, thus reducing feature map sizes and enlarging receptive fields. Past work has shown that this gives rise to better generalization and performance, particularly in node and graph classification tasks. In this paper, we apply variants of the Graph U-Net architecture to the `ogbn-arxiv` dataset. We make changes to the Graph U-Net architecture to allow it to handle larger datasets, as well as other experimental modifications such as batch normalization, linear classification layers, and activation functions. We also present a new architecture that builds on the U-Net architecture by incorporating pseudo-labeling and graph attention networks (GATs), finding that such a model achieves a best performance of 72.67% test accuracy on the `ogbn-arxiv` node prediction task. This is an improvement from the existing U-Net model baseline accuracy of 68.04%, and a stronger performance than the other OGB baselines'.

1 Introduction

Background. Encoder-decoder architectures like U-Nets have been successfully applied to image pixel-wise prediction tasks, such as image segmentation. At a high level, the encoder applies convolutions on the input (image) with maxpool downsampling in order to encode the input into feature representations at multiple levels. This pooling downsampling step can reduce sizes of feature maps and enlarge receptive fields, thereby giving rise to better generalization and performance [2]. At the same time, it effectively augments the data and thus uses annotated samples more efficiently.

The decoding step then projects the lower resolution features learned by the encoder into a higher resolution spaces to get a dense classification. This is carried out by doing upsampling and concatenation followed by convolution operations repeatedly.

Gao et al. [1] generalized this architecture to the domain of graphs and found that such an architecture achieved consistently better performance than previous existing models in node and graph classification tasks. We will explain this method in greater detail in Section 2.

Dataset. The `ogbn-arxiv` dataset [3, 4, 5] is a directed graph, representing the citation network between all Computer Science arXiv papers indexed by MAG. Each node is an arXiv paper and each directed edge indicates that one paper cites another one. It consists of roughly 169K nodes, where each node has 128 features which are obtained from text embeddings extracted from the paper's title and abstract. The goal is to predict the primary categories of the arXiv papers (i.e. nodes), of which ther are 40 – making this a 40-class classification problem. For this dataset, training is done on papers published until 2017, validation on those published in 2018, and testing on those published since 2019.

Preprint. Under review.

Dataset motivation. Graph U-Nets have primarily been applied to node classification tasks, though the method can generally also be applied to graph or link prediction. The method was applied successfully to other citation network datasets in a transductive learning setting, where only a few nodes used for training were labeled but other nodes in the same graph remain unlabeled, and thus it would be interesting to examine how well it performs in an inductive learning setting.

The dataset is also significantly larger than datasets previously examined by Gao et al., which only had on the order of thousands of nodes. Moreover, those nodes only had 7 classes at most, thus it is an open question as to whether the graph U-Net architecture will generalize well to a larger dataset with a classification problem that is significantly more complex.

Final Model. Fig. 1 shows a high level overview of our final improved model based on U-Nets. We explain more details about our improved model in section 3

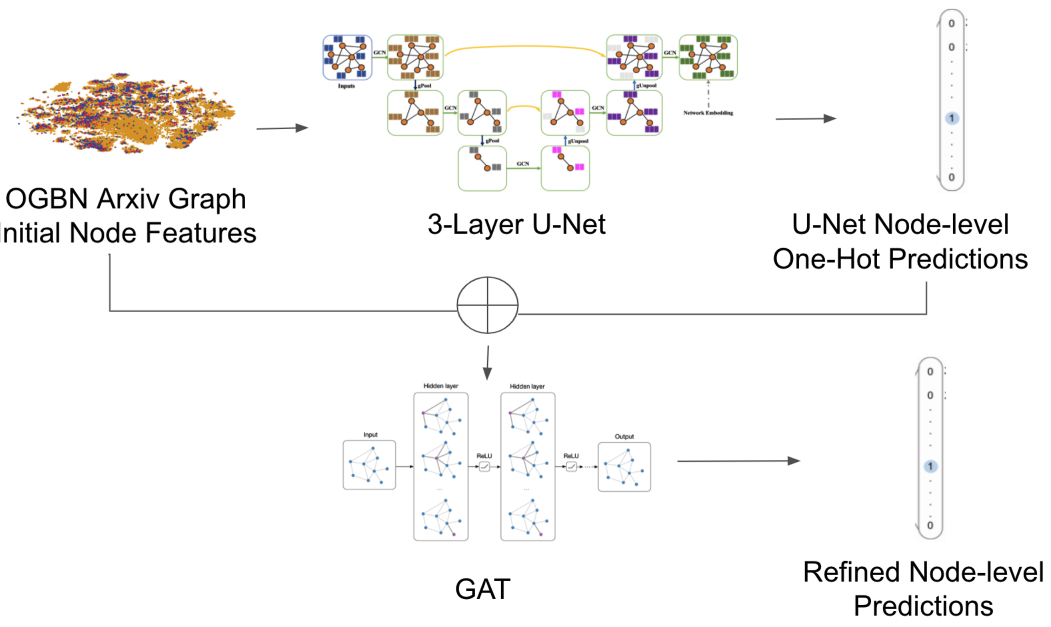

Figure 1: High level overview of our final improved model. First, we train an improved U-Net on the dataset. Next, we predict the classification for every node from that trained U-Net. Then, we concatenate the original node embeddings with the one-hot predictions and train a GAT-style architecture to get our final node predictions.

2 Existing Method

U-Nets for image segmentation (shown in Fig. 2) rely on the local information of pixels in order to downsample and upsample. However, graphs have no spatial locality or order information that would be required by typical pooling operations. Gao et al. thus developed a new framework that would extend downsampling and upsampling operations to graphs.

Firstly, Gao et al. developed a novel pooling layer, `gPool`, where node embeddings are projected down to a lower dimension and nodes with the highest projection values are retained. More formally, $A^{\ell+1} = A^{\ell}(\text{idx}, \text{idx})$ and $X^{\ell+1} = X^{\ell}(\text{idx}, :) \odot \text{sigmoid}(y(\text{idx}))\mathbf{1}^T$ where idx is the indices of the nodes to preserve, A is the adjacency matrix, X is the matrix of all the feature embeddings, y is the result of the projection, and $\odot$ means element wise multiplication. The scaling of the features allows for training the projection vector by backpropagation. We note that edge information is not incorporated when deciding which nodes to keep. Therefore, one possible improvement would be to investigate different approaches to incorporating edge information in this node preservation step.

Secondly, they developed an unpooling layer, `gUnpool`, which is the inverse of the pooling operation. This layer "restores" the original graph structure in the corresponding pooling layer by upsampling

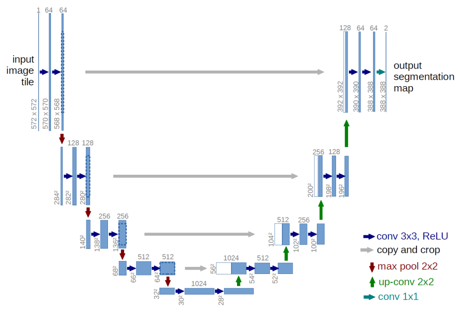

Figure 2: A sketch of the U-Net architecture from the original U-Net paper. [6]

the nodes and adding the missing edges back. The node feature matrix $X^{\ell+1}$ preserves the features of the nodes already presnt at layer l and contains vectors of zeroes at every row of $X^{\ell+1}$ corresponding to the nodes that were just added back.

Overall, the graph U-Net architecture allows the neural network to capture global context information, to be very data efficient (requiring few training examples), and to leverage information from a large receptive field. In doing so, it learns high-level feature encodings that enable better generalization and performance.

Finally, we note that **this method has not been featured in the leaderboard**.

3 Improvements to Existing Method

3.1 Scaling to larger datasets

Because our experiments are run on a 12GB GPU, there are limits to the size of models we can run. We found that the Graph U-Net architectures originally proposed by Gao et al. did not scale well to the 169K nodes in the `ogbn-arxiv` dataset.

In particular, the authors proposed taking the square of the adjacency matrix prior to each pooling layer due to the sparsity that gets introduced from pooling. For `ogbn-arxiv`, this matrix squaring step would result in roughly 59 million edges at the first layer, which can be represented as a sparse tensor. However, further operations on it were computationally unfeasible, as it would have to be converted into a “dense” format (i.e. tensor of edge indices) during the pooling step, which exceeded the computational limits of our machine. Due to these limitations, we removed the squaring of the adjacency matrix from the downsampling step for both the baseline method implementation and our variant implementations.

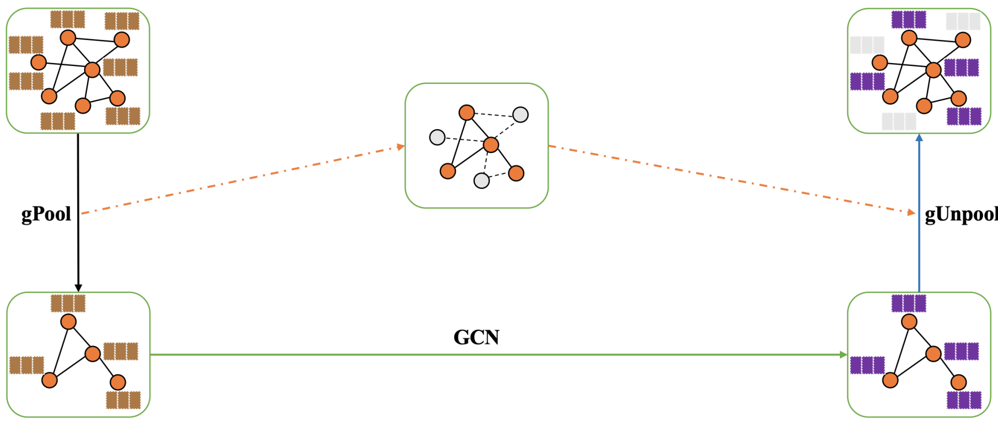

Figure 3: A brief sketch of the pooling-unpooling architecture from the paper [1]. We note that this is only one layer of pooling and unpooling and not the entire graph U-Net architecture.

3.2 Batch normalization, linear layers, and activation functions

The original Graph U-Net architecture described in the paper was based on stacking graph convolutions and pooling/unpooling layers, but used identity activations and no batch normalization. We find that using non-identity activation functions and instituting batch normalization is beneficial to generalization performance.

Batch normalization, originally proposed by Ioffe et al. [7], is a widely used mechanism in deep learning due to its stabilizing effect during training by normalizing inputs to each neural network that eliminates internal covariate shift as layers continually change. It has also been shown to act as a regularizer and reduce the need for dropout in some instances. Li et al. demonstrate that batch normalization is an effective tool to use in training Deep GCNs [8].

Next, activation functions such as Rectified Linear Units (ReLU) or Exponential Linear Units (ELU) [9] have several benefits when training deep neural networks by introducing non-linear relationships and mitigating the "vanishing gradient" problem. We experiment with these activation functions in lieu of the default identity activation used by Gao et al.

Finally, instead of a final graph convolution layer after the U-Net, we use two linear layers for classification, finding that this consistently achieves better performance.

3.3 "Pseudo-Labeling" using a U-Net

Pseudo-labeling [10] is a commonly used self-supervised learning technique that adds confidently predicted unlabeled data to the training dataset and then refits the model, which allows the model to have larger exposure to the input feature space with reasonably accurate labels. Motivated by this idea, we leverage our trained U-Net as a "pseudo-labeler" that generates one-hot predicted labels for each node in the training set. These predicted labels are then concatenated with the original node features and fed into a Graph Attention Network (GAT) style-architecture [11].

Our motivation for using Graph Attention Networks (GAT) is that these networks have been shown to outperform Graph Convolutional Networks (GCNs) in transductive node classification settings. This is due to the fact that GATs do not symmetrically aggregate the feature repesentations of the neighborhood of each node but instead use a self-attention mechanism to weight certain neighbors more than others during aggregation. Rather than stacking GAT layers sequentially with activations, we stack "GAT blocks" where each block consists of a GAT layer summed with a residual linear layer skip connection followed by Batch Normalization and ReLU activation [8] (see Fig. 4 for an overview of this pipeline). .

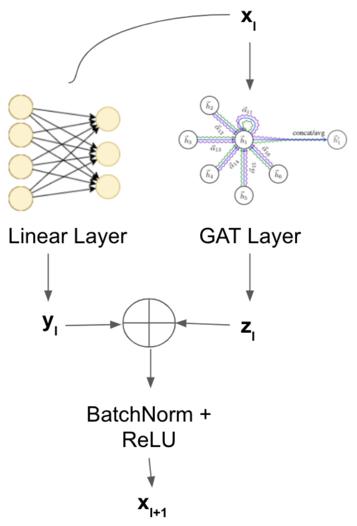

Figure 4: A visual depiction of one block of our GAT network where x_l denotes the input to block l.

Our intuition behind such a "pseudo-labeling" mechanism is that U-Nets tend to capture a more global representation around each node due to the pooling and unpooling sequences that increase the receptive fields of each node. This has been shown to lead to a high capacity for generalization. On the other hand, GATs capture a more local node representation by attending over the direct neighbors of each node at every layers and aggregating features from those neighbors. Hence, leveraging both representations when making node-level predictions can be beneficial.

In addition, this method allows us to train both a U-Net and a deep Graph Attention Network within the computational constraints of a single 12GB GPU.

4 Experiments

In this section, we detail our experiments with various architectures for the `ogbn-arxiv` prediction task. For each variant of the architecture, we explored various hyperparameter configurations using Weights & Biases [12] sweep configuration over the hyperparameter space.

Note: We compute confidence intervals only over 5 runs for the baseline existing model and our best performing model. For all other experiments, we report the scores from a single run.

4.1 U-Net base variants

We first implement and evaluate some basic variants of the baseline existing model, where only the improvements discussed in Section 3.1 and 3.2 are incorporated into the architecture.

For the U-Net base model and our implemented variants, we used the following hyperparameters: `learning rate`: [0.0001, 0.001*, 0.01], `depth`: [3*, 4], `hidden_channels`: [128*], `dropout`: [0.1*, 0.2, 0.3, 0.4, 0.5], `epochs`: [200, 300, 400, 500*], `activation`: [relu, elu*], where (*) denotes the hyperparameters we eventually selected based on validation performance. The number of pool factors is dependent on (i.e. equal to) the depth of the U-Net architecture. Given that each pool factor can take a value from 0 to 1, the feature space for the pool factors is computationally infeasible to cover. Instead, we experiment with arbitrarily chosen pool factors, and document these results in the next few sub-sections. Our intuition with these choices is that smaller pooling factors leads to a more global representation of each node whereas larger pooling factors lead to a more local representation that mostly considers node's direct k-hop neighborhoods.

Finally, given the range of different architectures we test, we lacked the time and resources to select optimal hyperparameters specific to each architecture. Instead, we chose 'universal' hyperparameters that achieved generally good results for all the U-Net-based architectures. Moreover, we note that the choice of some of these hyperparameters are limited by compute as well. Hidden channels, for example, was set to 128 because our other options (192, 256) resulted in out-of-memory errors.

	Accuracy (%)	
Method	Validation	Test
Baseline U-Net (4-layer, Pool Factors: 0.5, 0.5, 0.5, 0.5)	69.08±0.45	68.04±0.65
3-layer Improved U-Net (Pool Factors: 0.5, 0.2, 0.1)	**73.13**	**71.72**
3-layer Improved U-Net (Pool Factors: 0.5, 0.5, 0.5)	72.72	71.68
3-layer Improved U-Net (Pool Factors: 0.9, 0.8, 0.7)	72.86	71.03
4-layer Improved U-Net (Pool Factors: 0.7, 0.5, 0.2, 0.1)	72.36	71.08
4-layer Improved U-Net (Pool Factors: 0.5, 0.5, 0.5, 0.5)	72.19	71.14

Table 1: Comparing baseline U-Net model performance against U-Net base variants' on `ogbn-arxiv`

We implement the baseline U-Net model as described in Gao et al. and explained in 2. For transductive learning tasks, the authors recommend a 4-layer U-Net architectures with pool factors of 0.5. We compare this to our base variants in Table 1. We find that our basic modifications are already able to achieve significantly better performance than the baseline model. In particular, a 3-layer improved U-Net architecture with pool factors of 0.5, 0.2, 0.1 achieves the best accuracy.

4.2 GAT + Pseudo-labeling from U-Net

Next, we implement the pseudo-labeling mechanism described in Section 3.3 where we use the U-Net as a pseudo-labeler for the GAT model. The U-Net is trained with the hyperparameters of the best performing U-Net base variant in Table 1.

For the GAT model we explored the following hyperparameters: `learning rate`: [0.001, 0.002*, 0.003], `depth`: `hidden_channels`: [90*], `dropout`: [0.1, 0.2, 0.3, 0.4, 0.5*], `epochs`: [200, 300, 400, 500*], `activation`: [relu*] where (*) denotes the hyperparameters we eventually selected based on validation performance. We note that due to compute limitations, 90 was the maximum number of hidden channels we could use.

	Accuracy (%)	
Method	Validation	Test
3-layer U-Net (Pooling Factors: 0.5, 0.2, 0.1) + GAT (**best model**)	**73.55**±0.19	**72.67**±0.26
3-layer U-Net (Pooling Factors: 0.9, 0.8, 0.7) + GAT	73.53	72.28
4-layer U-Net (Pooling Factors: 0.7, 0.5, 0.2, 0.1) + GAT	73.51	72.55

Table 2: Comparing different depths and pooling factors for our U-Net pseudo-labeler for `ogbn-arxiv`

4.3 Ablation Study

In our ablation study, we systematically remove individual design choices and improvements we made one-by-one from our best model to demonstrate the importance of each of our contributions. Note that the best model contains GAT with a U-Net pseudo-labeler that contains both batch normalization and activation functions. From Table 4.3, we can see that pseudo-labeling from a U-Net was an essential component to our model as the GAT trained without pseudo-labeling loses 0.77% absolute performance. In addition, both batch normalization and activation functions were important components to our performance, though not as important as pseudo-labeling.

	Accuracy (%)	
Method	Validation	Test
GAT with Pseudo-Labeling (**best model**)	**73.55**±0.19	**72.67**±0.26
GAT with no Pseudo-Labeling from U-Net	72.51	71.79
GAT with Pseudo-Labeling from U-Net without BatchNorm	73.13	72.24
GAT with Pseudo-Labeling from U-Net without activation	73.01	71.94

Table 3: Ablation results for our best model for `ogbn-arxiv`

4.4 Final Results

Our best model consisted of training a U-Net with pooling factors of 0.5, 0.2, 0.1 which performed pseudo-labeling as input to a second GAT. We compare our best model to several models from the original OGB benchmarks paper such as MLP, Node2Vec, GCN, and GraphSAGE, outperforming each of them. The best model significantly improves upon the OGB baselines and our baseline model as well with an absolute performance gain of 4.63% over the baseline.

	Accuracy (%)	
Method	Validation	Test
MLP	57.65±0.12	55.50±0.23
Node2Vec	71.29±0.13	70.07±0.13
GCN	73.00±0.17	71.74±0.29
GraphSAGE	72.77±0.16	71.49±0.27
Baseline U-Net	69.08±0.45	68.04±0.65
Best model	**73.55**±0.19	**72.67**±0.26

Table 4: Results for OGB baselines, baseline U-Net, and our best performing model for `ogbn-arxiv`

5 Conclusion

Overall, we adapted the U-Net model such that it was able to scale to a significantly larger dataset than those used in the original paper. With various other smaller architecture modifications, we were also able to obtain significantly better results than the baseline model on the `ogbn-arxiv` node prediction task. Finally, we propose an extension to the U-Net architecture through pseudo-labeling combined with graph attention networks, which further improved our performance. Compared to the baseline U-Net model, our best performing model reduces absolute error by 4.63% and has superior performance to the OGB baselines.

That being said, our final model's performance does not outperform some other models on the OGB leaderboard. In the future, we would like to explore this pseudo-labeling approach to see if it might generalize to other state-of-the-art models. Furthermore, Graph-U-Nets are computationally intensive and require significantly more computational resources to run more expressive models, such as models with a higher number of hidden channels. Given more resources, we would scale up our model's architecture and fully test the limits of our proposed architecture.

References

[1] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[2] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[3] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[4] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[8] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276, 2019.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[10] Eric Arazo, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[11] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[12] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.