Generating super-resolution brain graph using DualSRAtt-Net

Dengi Yan 02381324 | Robert Maye 01736765 | Simon Rendon Arango 02458574 | Yijie Gong 01747721 | Hachem Ben Abdelbaki 02454779

***Index Terms*—graphs, deep learning, generative models, brain connectomes**

I. Introduction

An illustration of the neuronal connections inside the brain is called a connectome, or brain graph. The brain's regions of interest (ROIs) are represented by nodes in the graph, and the connections or pathways that connect these regions are shown by edges. The granularity of the ROIs might be referred to as a brain graph's resolution. Whereas low-resolution graphs feature fewer, bigger ROIs, high-resolution graphs have many small, finely split ROIs.

MRI-scanned brain connectomes are very helpful to diagnose several brain disorders and deepen our understanding of brain structure. Generating high-resolution MRI scans is a very costly operation and requires very expensive sensors, as opposed to lower-resolution scans where data is more readily available. We aim to exploit Graph Generative Models to attempt to extract a high-definition connectome from a low-resolution scan. We experimented with multiple publicly available architectures and made a few changes to try to improve them.

II. Problem Setting

We are given a dataset consisting of 167 MRI scans of patients, where low-resolution and high-resolution brain graphs are available. The data is given in the form of a vectorized upper half of a covariance matrix between activations of different brain zones, where negative covariances are clipped to zero. The low-resolution matrices are of size 160×160 while the high-resolution graphs are of size 284×284. We will try to build an architecture that predicts the high-resolution graph from the low-resolution one, which will be trained on these samples.

III. Previous Work

A. Graph Super Resolution Block

The most important component in the architectures we will be using is the Graph Super Resolution (GSR) block. It was first introduced in [IR20]. The block uses the input graph features to generate a matrix that spectrally resembles the high-resolution graph. The output of this block is later enhanced through several convolution blocks to output the final high-resolution graph. The block works by using the linear transformation $A_{out} = WSU_{in}^{*}Z_{in}$ where Z_{in} are the input features, U_{in} is a matrix containing the eigenvectors of the input adjacency matrix, S is a concatenation of identity matrices and W is a learnable approximation to the eigenvector matrix of the high-resolution graph's adjacency matrix. This is a permutation invariant transformation, as a permutation of the nodes in the input graph would be cancelled by the inverse of the orthogonal eigenvector matrix U_{in}^{*} of the permuted input graph. The output feature matrix is computed as $X_{out} = A_{out}A_{out}^{*}$. Since W attempts to approximate the eigenvectors of the high-resolution graph, the loss is set to the mean squared error between W and the eigenvectors of the ground truth high-resolution graph. To make the result positive, we take the absolute value of the output and force a symmetric result using the transformation $A \to (A + A^{*})/2$. The adjacency matrix is also symetrically normalized.

B. Adversarial Regularisation

In a follow-up work [IR21], the training process of the neural architecture used is improved by adding an adversarial discriminator at the output during the training process. Let p the ground truth high resolution graph distribution and q the distribution generated by our model. This technique attempts to make q as close as possible to p by training a multi-layer perceptron model to detect whether a given graph is sampled from p or q. A component related to the discriminator is later added to the generator loss in order to train the generator to produce graphs that are accepted by the discriminator. This acts as a regularization loss component and helps close the gap between the generated graph distribution and the ground truth distribution.

IV. Improvement Attempts

A. Skip Connection

The final convolution blocks are used as refinement layers for the result of the GSR block. It therefore makes sense to make them predict the difference between the GSR output and the final output. We achieve this by adding a skip connection between the GSR block output and the final output. This makes the model easier to train, as it reduces the effects of the vanishing gradients problem and makes the task for the convolution blocks much easier. We can also add a linear transformation before adding in the skip connection.

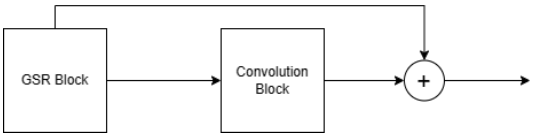

Fig. 1. Skip Connection

B. Different Convolution Types

The original model only utilized graph convolution layers for the final convolution block. We attempted several other types of layers. GINs (Graph Isomorphism Networks) have been introduced in [Xu+18] and are a more expressive alternative to graph convolution networks, as they're able to better differentiate between graphs. We replaced the final convolution layers by GIN layers. We also experimented with other combinations, such as interleaving GATs (Graph Attention Layer) between the convolutions to resemble a transformer network, but we eventually settled to just using GINs.

C. Self Attention Layer

We added classical self attention heads with residual connections after the GSR output. We found out this gives better results than using GATs. We believe these heads do exploit the graph structure, since the embeddings we're passing as input are actually equal to AA^*, where A is the adjacency matrix of the underlying graph. The output of this attention block is passed to the convolution block. Self Attention is permutation equivariant so using this block won't affect that aspect of the model.

D. Dual Paths

Instead of using one convolution block, we create two of them and pass in the input separately. The first path uses the original GSR block's output and the second path uses the attention layer's outputs. The final two results are concatenated and an MLP fuses the results back into one prediction.

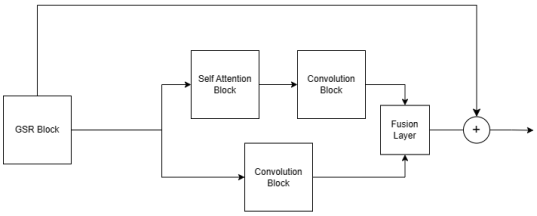

Fig. 2. Dual Path layout

E. Discriminator modifications

We made the discriminator model slightly larger and added some batch normalization layers in between the MLPs.

F. k-split Ensemble Model

During training, we split the train set into k different folds and train a model on each fold. A final model is later obtained, which averages the results of the k models trained previously.

G. Training Process

We train the model using an ADAM optimiser with weight decay. Several dropout layers are added into the network to reduce overfitting. We chose $k = 5$ for the k-split Ensemble Model and each submodel has been trained for 200 epochs. The k-split approach also allows us to do a k-fold cross validation for the underlying model architecture and evaluate its performance.

V. Results

We evaluate the models on the public test set. We will gradually incorporate Attention layers (Att), Weight Decay (WD), Double Path (DP) and Skip (S).

Model	MAE
Base AGSR	0.159135
AGSR+GIN	0.147896
AGSR+GAT&GIN	0.163586
AGSR+Att+WD	0.147321
AGSR+Att+WD+S	0.143079
Ensemble AGSR+Att+WD+S	0.135739
Ensemble AGSR+Att+WD+S+GIN+DP	**0.129105**

We tested the final version of our model using 3-fold cross validation. Each fold was allocated 200 epochs of training time, the process took about two hours and ten minutes to train on an M3 Pro GPU. The model and data took about 500 MB of VRAM during training. We used MAE (Mean Average Error) as a distance metric, two probability distribution error metrics : PCC (Pearson Correlation Coefficient) and JSD (Jenson-Shannon Divergence). We also looked at three different topological metrics : MAE-BC (Mean Average Error of Betweenness Centrality), MAE-PC (Mean Average Error of PageRank centrality) and MAE-EC (Mean Average Error of Eigenvector Centrality). The graphs can be found in the appendix.

VI. Conclusion

Through a range of incremental modifications, we have been able to get some improvements over the base AGSR model. Our work mainly affects what happens after the data exits the GSR Block, so it may be worth it to see if we could get further improvements by looking at the portion of the architecture before the super resolution block. The training process could also be further improved, by adding batching for example.

VII. Appendix

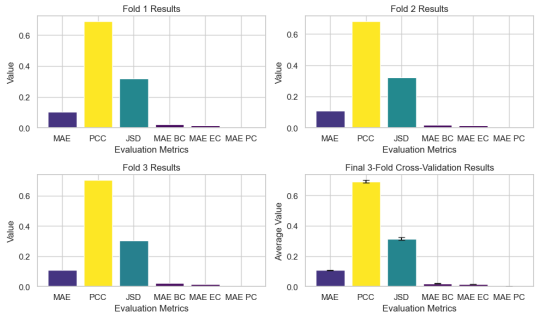

Fig. 3. All Metrics

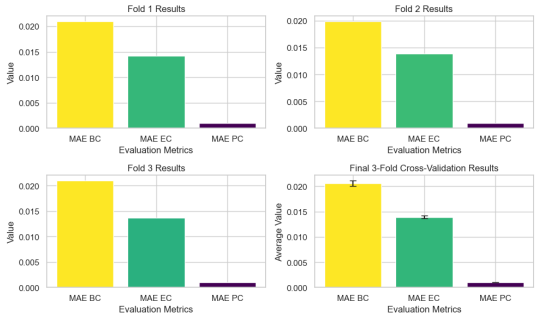

Fig. 4. Topological Metrics

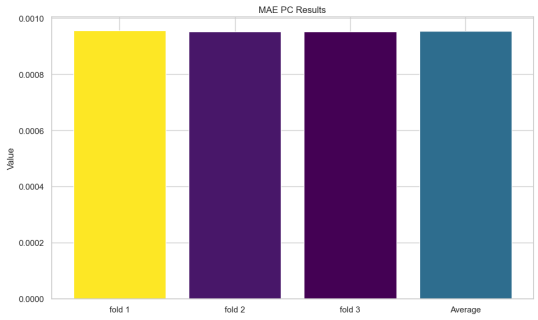

Fig. 5. PageRank Centrality

References

[Xu+18] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* 2018. eprint: arXiv:1810.00826.

[IR20] Megi Isallari and Islem Rekik. *GSR-Net: Graph Super-Resolution Network for Predicting High-Resolution from Low-Resolution Functional Brain Connectomes*. 2020. eprint: arXiv:2009.11080.

[IR21] Megi Isallari and Islem Rekik. *Brain Graph Super-Resolution Using Adversarial Graph Neural Network with Application to Functional Brain Connectivity*. 2021. eprint: arXiv:2105.00425.